
Administering Platform LSF

Platform LSF
Version 8.0
June 2011



Copyright

© 1994-2011 Platform Computing Corporation.

Although the information in this document has been carefully reviewed, Platform Computing Corporation ("Platform") does not warrant it to be free of errors or omissions. Platform reserves the right to make corrections, updates, revisions or changes to the information in this document.

UNLESS OTHERWISE EXPRESSLY STATED BY PLATFORM, THE PROGRAM DESCRIBED IN THIS DOCUMENT IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL PLATFORM COMPUTING BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT LIMITATION ANY LOST PROFITS, DATA, OR SAVINGS, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS PROGRAM.

We'd like to hear from you

You can help us make this document better by telling us what you think of the content, organization, and usefulness of the information. If you find an error, or just want to make a suggestion for improving this document, please address your comments to doc@platform.com.

Your comments should pertain only to Platform documentation. For product support, contact support@platform.com.

Document redistribution and translation

This document is protected by copyright and you may not redistribute or translate it into another language, in part or in whole.

Internal redistribution

You may only redistribute this document internally within your organization (for example, on an intranet) provided that you continue to check the Platform Web site for updates and update your version of the documentation. You may not make it available to your organization over the Internet.

Trademarks

LSF is a registered trademark of Platform Computing Corporation in the United States and in other jurisdictions.

ACCELERATING INTELLIGENCE, PLATFORM COMPUTING, PLATFORM SYMPHONY, PLATFORM JOB SCHEDULER, PLATFORM ISF, PLATFORM ENTERPRISE GRID ORCHESTRATOR, PLATFORM EGO, and the PLATFORM and PLATFORM LSF logos are trademarks of Platform Computing Corporation in the United States and in other jurisdictions.

UNIX is a registered trademark of The Open Group in the United States and in other jurisdictions.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Intel, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Other products or services mentioned in this document are identified by the trademarks or service marks of their respective owners.

Third-party license agreements

<http://www.platform.com/Company/third.part.license.htm>

Third-party copyright notices

<http://www.platform.com/Company/Third.Party.Copyright.htm>

Contents

Part I: Managing Your Cluster ... 7

1	Working with Your Cluster	9
2	Platform LSF Daemon Startup Control	33
3	Working with Hosts	43
4	Managing Jobs	89
5	Working with Queues	117
6	Platform LSF Resources	129
7	External Load Indices	155
8	Managing Users and User Groups	171
9	External Host and User Groups	179
10	Between-Host User Account Mapping	185
11	Cross-Cluster User Account Mapping	191
12	UNIX/Windows User Account Mapping	199

Part II: Licensing and Upgrading Platform LSF Versions ... 209

13	Platform LSF Licensing	211
14	Cluster Version Management and Patching on UNIX and Linux	237
15	Enable Platform LSF HPC Features	251

Part III: Monitoring Your Cluster ... 257

16	Achieving Performance and Scalability	259
17	Event Generation	277
18	Tuning the Cluster	281
19	Authentication and Authorization	291
20	Submitting Jobs with SSH	301
21	External Authentication	309
22	Job Email and Job File Spooling	319
23	Non-Shared File Systems	327

24	Error and Event Logging	335
25	Troubleshooting and Error Messages	349

Part IV: Time-Based Configuration ... 367

26	Time Configuration	369
27	Advance Reservation	379

Part V: Job Scheduling Policies ... 399

28	Preemptive Scheduling	401
29	Specifying Resource Requirements	415
30	Fairshare Scheduling	453
31	Resource Preemption	493
32	Guaranteed Resource Pools	505
33	Goal-Oriented SLA-Driven Scheduling	513
34	Exclusive Scheduling	533

Part VI: Job Scheduling and Dispatch ... 537

35	Working with Application Profiles	539
36	Resource Allocation Limits	555
37	Reserving Resources	571
38	Job Dependency and Job Priority	589
39	Job Requeue and Job Rerun	607
40	Job Migration	615
41	Job Checkpoint and Restart	627
42	Resizable Jobs	641
43	Chunk Jobs and Job Arrays	655
44	Job Packs	669
45	Running Parallel Jobs	673

Part VII: Job Execution and Interactive Jobs ... 715

46	Runtime Resource Usage Limits	717
47	Load Thresholds	733
48	Pre-Execution and Post-Execution Processing	739
49	Job Starters	755
50	Job Controls	763
51	External Job Submission and Execution Controls	769

52	Interactive Jobs with bsub	789
53	Interactive and Remote Tasks	805

Part VIII: Appendices ... 813

Appendix A: Submitting Jobs Using JSDL	815
Appendix B: Using lstash	827
Appendix C: Using Session Scheduler	835
Appendix D: Using lsmake	853
Appendix E: Managing Platform LSF on EGO	859



Managing Your Cluster

Working with Your Cluster

Learn about Platform LSF

Before using Platform LSF for the first time, you should download and read LSF Foundations Guide for an overall understanding of how LSF works.

Basic concepts

Job states

LSF jobs have the following states:

- **PEND** : Waiting in a queue for scheduling and dispatch
- **RUN** : Dispatched to a host and running
- **DONE**: Finished normally with zero exit value
- **EXIT** : Finished with non-zero exit value
- **PSUSP**: Suspended while pending
- **USUSP**: Suspended by user
- **SSUSP** : Suspended by the LSF system
- **POST_DONE**: Post-processing completed without errors
- **POST_ERR** : Post-processing completed with errors
- **UNKWN**: mbat chd has lost contact with sbat chd on the host on which the job runs
- **WAIT**: For jobs submitted to a chunk job queue, members of a chunk job that are waiting to run
- **ZOMBI**: A job becomes ZOMBI if the execution host is unreachable when a non-rerunnable job is killed or a rerunnable job is requeued

Host

An individual computer in the cluster.

Each host may have more than 1 processor. Multiprocessor hosts are used to run parallel jobs. A multiprocessor host with a single process queue is considered a single machine, while a box full of processors that each have their own process queue is treated as a group of separate machines.

Tip:

The names of your hosts should be unique. They should not be the same as the cluster name or any queue defined for the cluster.

Job

A unit of work run in the LSF system. A job is a command submitted to LSF for execution, using the `bsub` command. LSF schedules, controls, and tracks the job according to configured policies.

Jobs can be complex problems, simulation scenarios, extensive calculations, anything that needs compute power.

Job files

When a job is submitted to a queue, LSF holds it in a job file until conditions are right for it to be executed. Then the job file is used to execute the job.

UNIX: The job file is a Bourne shell script run at execution time.

Windows: The job file is a batch file processed at execution time.

Interactive batch job

A batch job that allows you to interact with the application and still take advantage of LSF scheduling policies and fault tolerance. All input and output are through the terminal that you used to type the job submission command.

When you submit an interactive job, a message is displayed while the job is awaiting scheduling. A new job cannot be submitted until the interactive job is completed or terminated.

Interactive task

A command that is not submitted to a batch queue and scheduled by LSF, but is dispatched immediately. LSF locates the resources needed by the task and chooses the best host among the candidate hosts that has the required resources and is lightly loaded. Each command can be a single process, or it can be a group of cooperating processes.

Tasks are run without using the batch processing features of LSF but still with the advantage of resource requirements and selection of the best host to run the task based on load.

Local task

An application or command that does not make sense to run remotely. For example, the `ls` command on UNIX.

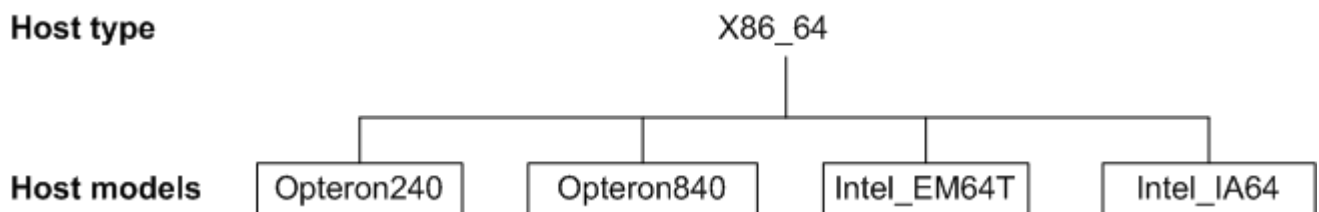
Remote task

An application or command that can be run on another machine in the cluster.

Host types and host models

Hosts in LSF are characterized by host type and host model.

The following example is a host with type `X86_64`, with host models `Opteron240`, `Opteron840`, `Intel_EM64T`, `Intel_IA64`, etc.



Host type

The combination of operating system and host CPU architecture.

All computers that run the same operating system on the same computer architecture are of the same type — in other words, binary-compatible with each other.

Each host type usually requires a different set of LSF binary files.

Host model

The host type of the computer, which determines the CPU speed scaling factor applied in load and placement calculations.

The CPU factor is taken into consideration when jobs are being dispatched.

Resources

Resource usage

The LSF system uses built-in and configured resources to track resource availability and usage. Jobs are scheduled according to the resources available on individual hosts.

Jobs submitted through the LSF system will have the resources they use monitored while they are running. This information is used to enforce resource limits and load thresholds as well as fairshare scheduling.

LSF collects information such as:

- Total CPU time consumed by all processes in the job
- Total resident memory usage in KB of all currently running processes in a job
- Total virtual memory usage in KB of all currently running processes in a job
- Currently active process group ID in a job
- Currently active processes in a job

On UNIX, job-level resource usage is collected through PIM.

Load indices

Load indices measure the availability of dynamic, non-shared resources on hosts in the cluster. Load indices built into the LIM are updated at fixed time intervals.

External load indices

Defined and configured by the LSF administrator and collected by an External Load Information Manager (ELIM) program. The ELIM also updates LIM when new values are received.

Static resources

Built-in resources that represent host information that does not change over time, such as the maximum RAM available to user processes or the number of processors in a machine. Most static resources are determined by the LIM at start-up time.

Static resources can be used to select appropriate hosts for particular jobs based on binary architecture, relative CPU speed, and system configuration.

Load thresholds

Two types of load thresholds can be configured by your LSF administrator to schedule jobs in queues. Each load threshold specifies a load index value:

- `loadSched` determines the load condition for dispatching pending jobs. If a host's load is beyond any defined `loadSched`, a job will not be started on the host. This threshold is also used as the condition for resuming suspended jobs.
- `loadStop` determines when running jobs should be suspended.

To schedule a job on a host, the load levels on that host must satisfy both the thresholds configured for that host and the thresholds for the queue from which the job is being dispatched.

The value of a load index may either increase or decrease with load, depending on the meaning of the specific load index. Therefore, when comparing the host load conditions with the threshold values, you need to use either greater than (>) or less than (<), depending on the load index.

Runtime resource usage limits

Limit the use of resources while a job is running. Jobs that consume more than the specified amount of a resource are signalled.

Hard and soft limits

Resource limits specified at the queue level are hard limits while those specified with job submission are soft limits. See `setrlimit(2)` man page for concepts of hard and soft limits.

Resource allocation limits

Restrict the amount of a given resource that must be available during job scheduling for different classes of jobs to start, and which resource consumers the limits apply to. If all of the resource has been consumed, no more jobs can be started until some of the resource is released.

Resource requirements (bsub -R)

Restrict which hosts the job can run on. Hosts that match the resource requirements are the candidate hosts. When LSF schedules a job, it collects the load index values of all the candidate hosts and compares them to the scheduling conditions. Jobs are only dispatched to a host if all load values are within the scheduling thresholds.

View cluster information

LSF provides commands for users to access information about the cluster.

- Cluster information includes the cluster master host, cluster name, cluster resource definitions, cluster administrator, and so on.

To view the ...	Run ...
Version of LSF	<code>l s i d</code>
Cluster name	<code>l s i d</code>
Current master host	<code>l s i d</code>
Cluster administrators	<code>l s c l u s t e r s</code>
Configuration parameters	<code>b p a r a m s</code>

View LSF version, cluster name, and current master host

- Run `l s i d` to display the version of LSF, the name of your cluster, and the current master host:

```
lsid
Platform LSF 8 Jan 10 2011
Copyright 1992-2011 Platform Computing Corporation

My cluster name is cluster1
My master name is hostA
```

View cluster administrators

- Run `l s c l u s t e r s` to find out who your cluster administrator is and see a summary of your cluster:

```
lsclusters
CLUSTER_NAME  STATUS  MASTER_HOST  ADMIN  HOSTS  SERVERS
cluster1      ok      hostA        lsfadmin  6      6
```

If you are using the LSF MultiCluster product, you will see one line for each of the clusters that your local cluster is connected to in the output of `l s c l u s t e r s`.

View configuration parameters

- Run `b p a r a m s` to display the generic configuration parameters of LSF. These include default queues, job dispatch interval, job checking interval, and job accepting interval.

```
bparams
Default Queues:  normal idle
MBD_SLEEP_TIME used for calculations: 20 seconds
Job Checking Interval: 15 seconds
Job Accepting Interval: 20 seconds
```

- Run `b p a r a m s -l` to display the information in long format, which gives a brief description of each parameter and the name of the parameter as it appears in `l s b. p a r a m s`.

```
bparams -l
System default queues for automatic queue selection:
  DEFAULT_QUEUE = normal idle
Amount of time in seconds used for calculating parameter values:
```

```

MBD_SLEEP_TIME = 20 (seconds)
The interval for checking jobs by slave batch daemon:
SBD_SLEEP_TIME = 15 (seconds)
The interval for a host to accept two batch jobs subsequently:
JOB_ACCEPT_INTERVAL = 1 (* MBD_SLEEP_TIME)
The idle time of a host for resuming pg suspended jobs:
PG_SUSP_IT = 180 (seconds)
The amount of time during which finished jobs are kept in core:
CLEAN_PERIOD = 3600 (seconds)
The maximum number of finished jobs that are logged in current event file:
MAX_JOB_NUM = 2000
The maximum number of retries for reaching a slave batch daemon:
MAX_SBD_FAIL = 3
The number of hours of resource consumption history:
HIST_HOURS = 5
The default project assigned to jobs.
DEFAULT_PROJECT = default
Sync up host status with master LIM is enabled:
LSB_SYNC_HOST_STAT_LIM = Y
MBD child query processes will only run on the following CPUs:
MBD_QUERY_CPUS=1 2 3

```

3. Run `bparams -a` to display all configuration parameters and their values in `lsb.params`.

For example:

```

bparams -a
lsb.params configuration at Fri Jun 8 10:27:52 CST 2011
  MBD_SLEEP_TIME = 20
  SBD_SLEEP_TIME = 15
  JOB_ACCEPT_INTERVAL = 1
  SUB_TRY_INTERVAL = 60
  LSB_SYNC_HOST_STAT_LIM = N
  MAX_JOBINFO_QUERY_PERIOD = 2147483647
  PEND_REASON_UPDATE_INTERVAL = 30

```

View daemon parameter configuration

Log on to a server host.

1. Display all configuration settings for running LSF daemons.
 - Run `lsadmin showconf` to display all configured parameters and their values in `lsf.conf` or `ego.conf` for LIM.
 - Run `badmin showconf` to display all configured parameters and their values in `lsf.conf` or `ego.conf` for `mbatchd` and `sbatchd`.

In a MultiCluster environment, the parameters apply to the local cluster only.

2. Display `mbatchd` and root `sbatchd` configuration.
 - Run `badmin showconf mbd` to display the parameters configured in `lsf.conf` or `ego.conf` that apply to `mbatchd`.
 - Run `badmin showconf sbd` to display the parameters configured in `lsf.conf` or `ego.conf` that apply to root `sbatchd`.

Examples

- Show `mbatchd` configuration:

```

badmin showconf mbd
MBD configuration at Fri Jun 8 10:27:52 CST 2011
  LSB_SHAREDIR=/scratch/dev/lsf/user1/0604/work
  LSF_CONFDIR=/scratch/dev/lsf/user1/0604/conf
  LSF_LOG_MASK=LOG_WARNING
  LSF_ENVDIR=/scratch/dev/lsf/user1/0604/conf
  LSF_EGO_DAEMON_CONTROL=N

```

- Show sbat chd configuration on a specific host:

```
badmin showconf sbd hosta
SBD configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2011
LSB_SHAREDIR=/scratch/dev/l sf/user1/0604/work
LSF_CONFDIR=/scratch/dev/l sf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/l sf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

- Show sbat chd configuration for all hosts:

```
badmin showconf sbd all
SBD configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2011
LSB_SHAREDIR=/scratch/dev/l sf/user1/0604/work
LSF_CONFDIR=/scratch/dev/l sf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/l sf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
...
SBD configuration for host <hostb> at Fri Jun 8 10:27:52 CST 2011
LSB_SHAREDIR=/scratch/dev/l sf/user1/0604/work
LSF_CONFDIR=/scratch/dev/l sf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/l sf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

- Show lim configuration:

```
lsadmin showconf lim
LIM configuration at Fri Jun 8 10:27:52 CST 2010
LSB_SHAREDIR=/scratch/dev/l sf/user1/0604/work
LSF_CONFDIR=/scratch/dev/l sf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/l sf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

- Show lim configuration for a specific host:

```
lsadmin showconf lim hosta
LIM configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2011
LSB_SHAREDIR=/scratch/dev/l sf/user1/0604/work
LSF_CONFDIR=/scratch/dev/l sf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/l sf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

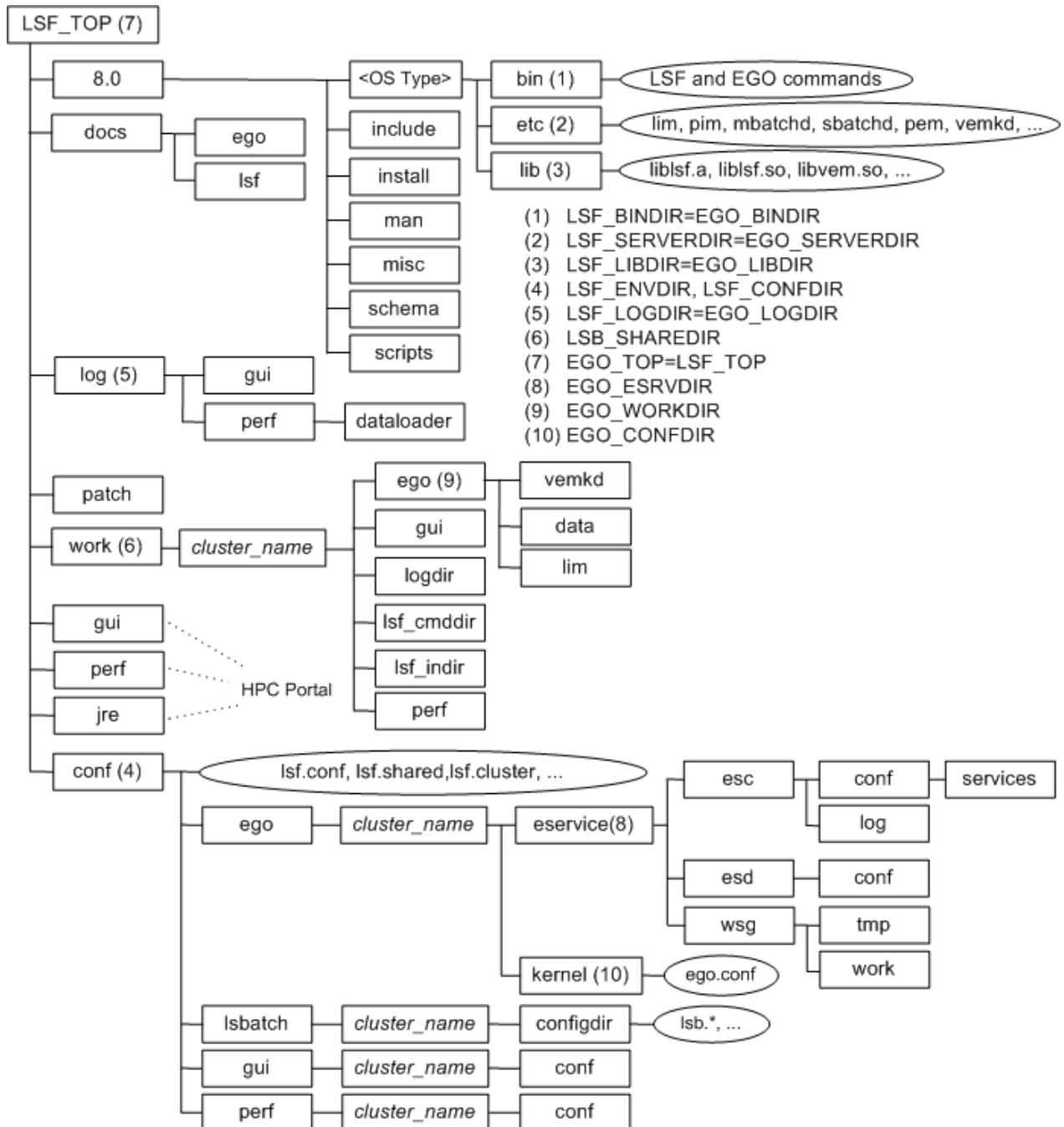
- Show lim configuration for all hosts:

```
lsadmin showconf lim all
LIM configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2011
LSB_SHAREDIR=/scratch/dev/l sf/user1/0604/work
LSF_CONFDIR=/scratch/dev/l sf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/l sf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
...
LIM configuration for host <hostb> at Fri Jun 8 10:27:52 CST 2011
LSB_SHAREDIR=/scratch/dev/l sf/user1/0604/work
LSF_CONFDIR=/scratch/dev/l sf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/l sf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

Example directory structures

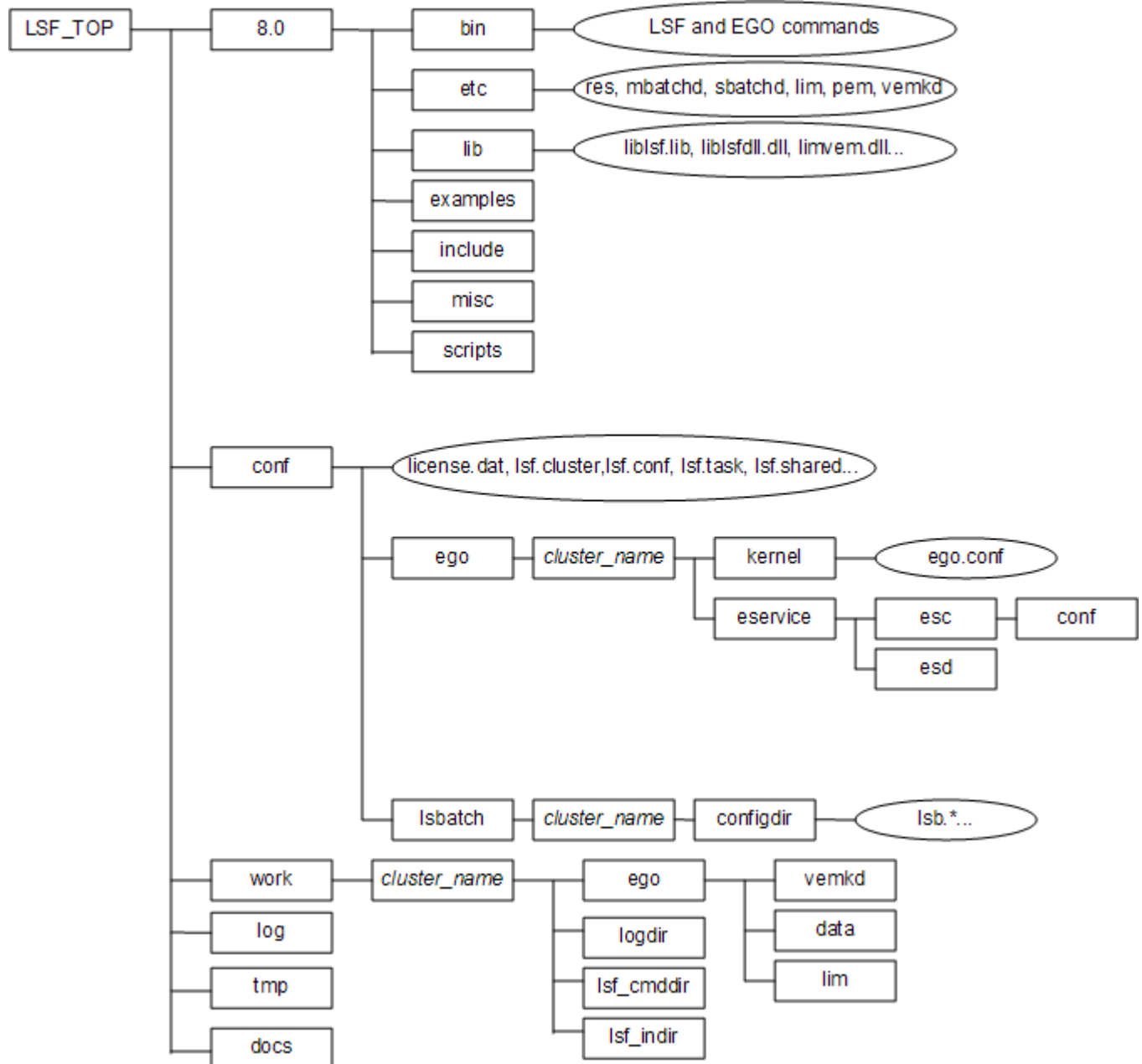
UNIX and Linux

The following figures show typical directory structures for a new UNIX or Linux installation with `lsfinstall`. Depending on which products you have installed and platforms you have selected, your directory structure may vary.



Microsoft Windows

The following diagram shows an example directory structure for a Windows installation.



Add cluster administrators

Primary Cluster Administrator

Required. The first cluster administrator, specified during installation. The primary LSF administrator account owns the configuration and log files. The primary LSF administrator has permission to perform clusterwide operations, change configuration files, reconfigure the cluster, and control jobs submitted by all users.

Other Cluster Administrators

Optional. May be configured during or after installation.

Cluster administrators can perform administrative operations on all jobs and queues in the cluster. Cluster administrators have the same cluster-wide operational privileges as the primary LSF administrator except that they do not have permission to change LSF configuration files.

1. In the `ClusterAdmins` section of `lsf.cluster.cluster_name`, specify the list of cluster administrators following `ADMINISTRATORS`, separated by spaces.

You can specify user names and group names.

The first administrator in the list is the primary LSF administrator. All others are cluster administrators.

For example:

```
Begin ClusterAdmins
ADMINISTRATORS = lsfadmin admin1 admin2
End ClusterAdmins
```

2. Save your changes.
3. Run `lsadmin reconfig` to restart LIM.
4. Run `badmin mbdrestart` to restart `mbatchd`.

Control daemons

Permissions required

To control all daemons in the cluster, you must

- Be logged on as root or as a user listed in the `/etc/l sf . sudoers` file. See the *LSF Configuration Reference* for configuration details of `l sf . sudoers`.
- Be able to run the `rsh` or `ssh` commands across all LSF hosts without having to enter a password. See your operating system documentation for information about configuring the `rsh` and `ssh` commands. The shell command specified by `LSF_RSH` in `l sf . conf` is used before `rsh` is tried.

Daemon commands

The following is an overview of commands you use to control LSF daemons.

Daemon	Action	Command	Permissions
All in cluster	Start	<code>lsfstartup</code>	Must be root or a user listed in <code>l sf . sudoers</code> for all these commands
	Shut down	<code>lsfshutdown</code>	
<code>sbat chd</code>	Start	<code>badadmin hstartup [host_name ... all]</code>	Must be root or a user listed in <code>l sf . sudoers</code> for the startup command
	Restart	<code>badadmin hrestart [host_name ... all]</code>	Must be root or the LSF administrator for other commands
	Shut down	<code>badadmin hshutdown [host_name ... all]</code>	
<code>mbat chd</code> <code>mbschd</code>	Restart	<code>badadmin mbdrestart</code>	Must be root or the LSF administrator for these commands
	Shut down	<ol style="list-style-type: none"> 1. <code>badadmin hshutdown</code> 2. <code>badadmin mbdrestart</code> 	
	Reconfigure	<code>badadmin reconfig</code>	
RES	Start	<code>lsadmin resstartup [host_name ... all]</code>	Must be root or a user listed in <code>l sf . sudoers</code> for the startup command
	Shut down	<code>lsadmin resshutdown [host_name ... all]</code>	Must be the LSF administrator for other commands
	Restart	<code>lsadmin resrestart [host_name ... all]</code>	
LIM	Start	<code>lsadmin limstartup [host_name ... all]</code>	Must be root or a user listed in <code>l sf . sudoers</code> for the startup command

Daemon	Action	Command	Permissions
	Shut down	lsadmin limshutdown [host_name ... all]	Must be the LSF administrator for other commands
	Restart	lsadmin limrestart [host_name ... all]	
	Restart all in cluster	lsadmin reconfig	

sbatchd

Restarting `sbatchd` on a host does not affect jobs that are running on that host.

If `sbatchd` is shut down, the host is not available to run new jobs. Existing jobs running on that host continue, but the results are not sent to the user until `sbatchd` is restarted.

LIM and RES

Jobs running on the host are not affected by restarting the daemons.

If a daemon is not responding to network connections, `lsadmin` displays an error message with the host name. In this case you must kill and restart the daemon manually.

If the LIM and the other daemons on the current master host shut down, another host automatically takes over as master.

If the RES is shut down while remote interactive tasks are running on the host, the running tasks continue but no new tasks are accepted.

Control mbatchd

- You use the `badmi n` command to control `mbatchd`.

Reconfigure mbatchd

If you add a host to a host group, a host to a queue, or change resource configuration in the Hosts section of `lsf.cluster.cluster_name`, the change is not recognized by jobs that were submitted before you reconfigured.

If you want the new host to be recognized, you must restart `mbatchd` (or add the host using the `bconf` command if you are using live reconfiguration).

1. Run `badmi n reconfi g`.

When you reconfigure the cluster, `mbatchd` is not restarted. Only configuration files are reloaded.

Restart mbatchd

1. Run `badmi n mbdrestart`.

LSF checks configuration files for errors and prints the results to `stderr`. If no errors are found, the following occurs:

- Configuration files are reloaded
- `mbatchd` is restarted
- Events in `lsb.events` are reread and replayed to recover the running state of the last `mbatchd`

Tip:

Whenever `mbatchd` is restarted, it is unavailable to service requests. In large clusters where there are many events in `lsb.events`, restarting `mbatchd` can take some time. To avoid replaying events in `lsb.events`, use the command `badmi n reconfi g`.

Log a comment when restarting mbatchd

1. Use the `-C` option of `badmi n mbdrestart` to log an administrator comment in `lsb.events`.

For example:

```
badmi n mbdrestart -C "Configuration change"
```

The comment text `Configuration change` is recorded in `lsb.events`.

2. Run `badmi n hi st` or `badmi n mbdhi st` to display administrator comments for `mbatchd` restart.

Shut down mbatchd

1. Run `badmi n hshutdown` to shut down `mbatchd` on the master host.

For example:

```
badmi n hshutdown hostD  
Shut down slave batch daemon on <hostD> .... done
```

2. Run `badmi n mbdrestart`:

```
badmi n mbdrestart
Checking configuration files ...
No errors found.
```

This causes `mbat chd` and `mbschd` to exit. `mbat chd` cannot be restarted, because `sbat chd` is shut down. All LSF services are temporarily unavailable, but existing jobs are not affected. When `mbat chd` is later started by `sbat chd`, its previous status is restored from the event log file and job scheduling continues.

Customize batch command messages

LSF displays error messages when a batch command cannot communicate with `mbat chd`. Users see these messages when the batch command retries the connection to `mbat chd`.

You can customize three of these messages to provide LSF users with more detailed information and instructions.

1. In the file `lsf . conf`, identify the parameter for the message that you want to customize.

The following lists the parameters you can use to customize messages when a batch command does not receive a response from `mbat chd`.

Reason for no response from mbatchd	Default message	Parameter used to customize the message
<code>mbat chd</code> is too busy to accept new connections or respond to client requests	LSF is processing your request. Please wait...	<code>LSB_MBD_BUSY_MSG</code>
internal system connections to <code>mbat chd</code> fail	Cannot connect to LSF. Please wait...	<code>LSB_MBD_CONNECT_FAIL_MSG</code>
<code>mbat chd</code> is down or there is no process listening at either the <code>LSB_MBD_PORT</code> or the <code>LSB_QUERY_PORT</code>	LSF is down. Please wait...	<code>LSB_MBD_DOWN_MSG</code>

2. Specify a message string, or specify an empty string:

- To specify a message string, enclose the message text in quotation marks (") as shown in the following example:

```
LSB_MBD_BUSY_MSG="The mbatchd daemon is busy. Your command will retry every 5 minutes. No action required."
```

- To specify an empty string, type quotation marks (") as shown in the following example:

```
LSB_MBD_BUSY_MSG=""
```

Whether you specify a message string or an empty string, or leave the parameter undefined, the batch command retries the connection to `mbat chd` at the intervals specified by the parameters `LSB_API_CONNTIMEOUT` and `LSB_API_RECVMTIMEOUT`.

Note:

Before Version 7.0, LSF displayed the following message for all three message types: "batch daemon not responding...still trying." To display the previous default message, you must define each of the three message parameters and specify "batch daemon not responding...still trying" as the message string.

3. Save and close the `lsf . conf` file.

Reconfigure your cluster

After changing LSF configuration files, you must tell LSF to reread the files to update the configuration. Use the following commands to reconfigure a cluster:

- `lsadmin reconfig`
- `badmin reconfig`
- `badmin mbdrestart`

The reconfiguration commands you use depend on which files you change in LSF. The following table is a quick reference.

After making changes to ...	Use ...	Which ...
<code>hosts</code>	<code>badmin reconfig</code>	reloads configuration files
<code>license.dat</code>	<code>lsadmin reconfig</code> AND <code>badmin mbdrestart</code>	restarts LIM, reloads configuration files, and restarts <code>mbatchd</code>
<code>lsb.applications</code>	<code>badmin reconfig</code>	reloads configuration files Pending jobs use new application profile definition. Running jobs are not affected.
<code>lsb.hosts</code>	<code>badmin reconfig</code>	reloads configuration files
<code>lsb.modules</code>	<code>badmin reconfig</code>	reloads configuration files
<code>lsb.nqsmaps</code>	<code>badmin reconfig</code>	reloads configuration files
<code>lsb.params</code>	<code>badmin reconfig</code>	reloads configuration files
<code>lsb.queues</code>	<code>badmin reconfig</code>	reloads configuration files
<code>lsb.resources</code>	<code>badmin reconfig</code>	reloads configuration files
<code>lsb.serviceclasses</code>	<code>badmin reconfig</code>	reloads configuration files
<code>lsb.users</code>	<code>badmin reconfig</code>	reloads configuration files
<code>lsf.cluster.cluster_name</code>	<code>lsadmin reconfig</code> AND <code>badmin mbdrestart</code>	restarts LIM, reloads configuration files, and restarts <code>mbatchd</code>
<code>lsf.conf</code>	<code>lsadmin reconfig</code> AND <code>badmin mbdrestart</code>	restarts LIM, reloads configuration files, and restarts <code>mbatchd</code>
<code>lsf.licensescheduler</code>	<code>bladmin reconfig</code> <code>lsadmin reconfig</code> <code>badmin mbdrestart</code>	restarts <code>bl d</code> , restarts LIM, reloads configuration files, and restarts <code>mbatchd</code>
<code>lsf.shared</code>	<code>lsadmin reconfig</code> AND <code>badmin mbdrestart</code>	restarts LIM, reloads configuration files, and restarts <code>mbatchd</code>
<code>lsf.sudoers</code>	<code>badmin reconfig</code>	reloads configuration files
<code>lsf.task</code>	<code>lsadmin reconfig</code> AND <code>badmin reconfig</code>	restarts LIM and reloads configuration files

Reconfigure the cluster with lsadmin and badmin

To make a configuration change take effect, use this method to reconfigure the cluster.

1. Log on to the host as root or the LSF administrator.
2. Run `lsadmin reconfig` to restart LIM:

lsadmin reconfig

The `lsadmin reconfig` command checks for configuration errors.

If no errors are found, you are prompted to either restart `lim` on master host candidates only, or to confirm that you want to restart `lim` on all hosts. If fatal errors are found, reconfiguration is aborted.

3. Run `badmin reconfig` to reconfigure `mbatchd`:

badmin reconfig

The `badmin reconfig` command checks for configuration errors.

If fatal errors are found, reconfiguration is aborted.

Reconfigure the cluster by restarting mbatchd

To replay and recover the running state of the cluster, use this method to reconfigure the cluster.

1. Run `badmin mbdrestart` to restart `mbatchd`:

badmin mbdrestart

The `badmin mbdrestart` command checks for configuration errors.

If no fatal errors are found, you are asked to confirm `mbatchd` restart. If fatal errors are found, the command exits without taking any action.

Tip:

If the `lsb.events` file is large, or many jobs are running, restarting `mbatchd` can take some time. In addition, `mbatchd` is not available to service requests while it is restarted.

View configuration errors

1. Run `lsadmin ckconfig -v`.
2. Run `badmin ckconfig -v`.

This reports all errors to your terminal.

How reconfiguring the cluster affects licenses

If the license server goes down, LSF can continue to operate for a period of time until it attempts to renew licenses.

Reconfiguring causes LSF to renew licenses. If no license server is available, LSF does not reconfigure the system because the system would lose all its licenses and stop working.

If you have multiple license servers, reconfiguration proceeds provided LSF can contact at least one license server. In this case, LSF still loses the licenses on servers that are down, so LSF may have fewer licenses available after reconfiguration.

Live reconfiguration

You can use live reconfiguration to make configuration changes in LSF active memory that take effect immediately. Live reconfiguration requests use the `bconf` command, and generate updated configuration files in the directory set by `LSF_LIVE_CONFDIR` in `lsf.conf`. By default, `LSF_LIVE_CONFDIR` is set to `$LSB_SHAREDIR/cluster_name/live_confdir`. This directory is created automatically during LSF installation but remains empty until live reconfiguration requests write working configuration files into it later.

Live configuration changes made using the `bconf` command are recorded in the history file `liveconf.hist` located under `$LSB_SHAREDIR/cluster_name/logdir`, and can be queried using the `bconf hist` command. Not all configuration changes are supported by `bconf` and substantial configuration changes made using `bconf` may affect system performance for a few seconds.

When files exist in the directory set by `LSF_LIVE_CONFDIR`, all LSF restart and reconfig commands read the files in this directory instead of configuration files in `LSF_CONFDIR`. Merge the configuration files generated by `bconf` into `LSF_CONFDIR` regularly to avoid confusion. Alternatively, if you want `bconf` changes to overwrite original configuration files directly, set `LSF_LIVE_CONFDIR` to `LSF_CONFDIR`.

For more information about the `bconf` command syntax and a complete list of configuration changes supported by live reconfiguration, see the `bconf` man page or `bconf` in the *LSF Command Reference*.

bconf authentication

All `bconf` requests must be made from static servers; `bconf` requests from dynamic hosts or client hosts are not accepted.

Regular users can run `bconf hist` queries. Only cluster administrators and root can run all `bconf` commands.

User group administrators can do the following:

- with `usershares` rights: adjust user shares using `bconf update`, `addmember`, or `rmmember`
- with `full` rights: adjust both user shares and group members using `bconf update`, delete the user group using `bconf delete`, and create new user groups using `bconf create`

Note:

User group admins with `full` rights can only add a user group member to the user group if they also have `full` rights for the member user group.

User group administrators adding a new user group with `bconf create` are automatically added to `GROUP_ADMIN` with `full` rights for the new user group.

For more information about user group administrators see [Platform LSF user groups](#) on page 176 and the `lsb.users` man page or `lsb.users` in the *LSF Configuration Reference*.

Enable live reconfiguration

All configuration files should be free from warning messages when running `badmi n reconfi g`, and multiple sections in configuration files should be merged where possible. Configuration files should follow the order and syntax given in the template files.

1. Define `LSF_LIVE_CONFDIR` in `lsf.conf` using an absolute path.

2. Run `lsadmin reconfig` and `badmin mbdrestart` to apply the new parameter setting.

Running `bconf` creates updated copies of changed configuration files in the directory specified by `LSF_LIVE_CONFDIR`.

Important:

When a file exist in the directory set by `LSF_LIVE_CONFDIR`, all LSF restart and reconfig commands read the file in this directory instead of the equivalent configuration file in `LSF_CONFDIR`.

Add a host to the cluster using bconf

You can add a new slave host with boolean resources to your cluster using live reconfiguration.

1. Run `bconf add host=hostname`

For example:

```
bconf add host=host24 "MXJ=21;RESOURCES=bigmem"
bconf: Request for host <host24> accepted
```

Restriction:

If default is already defined in `lsb.hosts` without a model or type line, no new line is added to the `lsb.hosts` file. (Applies to hosts added without batch parameters.)

When using MultiCluster you cannot add leased hosts or any hosts from another cluster.

Newly added hosts won't join an existing advance reservation, or run existing pending jobs submitted to a host group with `bsub -m` where more than one host or hostgroup is specified.

Adding a faster host to the cluster will not update the `RUNLIMIT` definition in the queue to normalize with the new cpu factor.

Create a user group using bconf

1. Run `bconf create usergroup=group_name`

For example:

```
bconf create usergroup=ug12 "GROUP_MEMBER=user1 user2 ; USER_SHARES=[user1, 5]
[user2, 2] ; GROUP_ADMIN=admin1"
bconf: Request for usergroup <ug12> accepted
```

Once accepted by `bconf`, the new usergroup appears in `bugroup` output:

```
bugroup -l ug12
GROUP_NAME:  ug12
USERS:      user2 user1
GROUP_ADMIN: admin1
SHARES:     [user1, 5] [user2, 2]
```

Remove a user group member using bconf

You can remove members from a usergroup using live reconfiguration.

As well as removing the specified group member, all references to the group member are updated as required.

1. Run `bconf rmmember usergroup=group_name "GROUP_MEMBER=user_name"`

For example:

```
bconf rmmember usergroup=ug12 "GROUP_MEMBER=user1"
bconf: Request for usergroup <ug12> accepted
```

Once accepted by bconf, the changed usergroup appears in bugroup output:

```
bugroup -l ug12
GROUP_NAME:   ug12
USERS:        user2
GROUP_ADMIN:  admin1
SHARES:       [user2, 2]
```

Notice the SHARES entry for user1 has also been removed.

Create a limit using bconf

You can create new limits using live reconfiguration.

1. Run `bconf create limit=limit_name`

For example, to create the limit X1 with a job limit of 23 per host:

```
bconf create limit=X1 "JOBS=23;PER_HOST=host12"
bconf: Request for limit <X1> accepted
```

Once accepted by bconf, the new limit appears in blimits output:

```
blimits -cn X1
Begin Limit
NAME           = X1
PER_HOST       = host12
JOBS           = 23
End Limit
```

Limits created using

```
bconf create
```

are written to the changed lsb. resources configuration file in horizontal format.

Update a limit using bconf

1. Run `bconf update limit=limit_name`. For example:

```
bconf update limit=Lim3 "JOBS=20; SLOTS=100"
```

Examples of changing a limit in two steps

Changing a limit using bconf may require two bconf calls if you have a dependent value or want to change from an integer to a percentage setting.

For example, given the limit L1 configured in lsb. resources, MEM is dependent on PER_HOST:

```
Begin Limit
NAME           = L1
PER_USER       = all
PER_QUEUE      = normal priority
PER_HOST       = all
MEM            = 40%
End Limit
```


One `bconf` update call cannot reset both the `PER_HOST` value and dependent `MEM` percentage value:

```
bconf update limit=L1 "MEM=-;PER_HOST=-"
```

```
bconf: Request for limit <L1> rejected
```

```
Error(s): PER_HOST cannot be replaced due to the dependent resource MEM
```

Instead, reset `MEM` and `PER_HOST` in two steps:

```
bconf update limit=L1 "MEM=-;"
```

```
bconf: Request for limit <L1> accepted
```

```
bconf update limit=L1 "PER_HOST=-"
```

```
bconf: Request for limit <L1> accepted
```

Similarly changing the value of `SWP` from a percentage to an integer requires two steps:

```
Begin Limit
```

```
NAME = L1
```

```
...
```

```
SWP = 40%
```

```
End Limit
```

```
bconf update limit=L1 "SWP=20"
```

```
bconf: Request for limit <L1> rejected
```

```
Error(s): Cannot change between integer and percentage directly; reset the resource first
```

First reset `SWP` and then set as an integer, calling `bconf` twice:

```
bconf update limit=L1 "SWP=-;"
```

```
bconf: Request for limit <L1> accepted
```

```
bconf update limit=L1 "SWP=20"
```

```
bconf: Request for limit <L1> accepted
```

Add a user share to a fairshare queue

You can add a member and share to a fairshare queue in `lsb.queues` using live reconfiguration.

1. Run `bconf addmember queue=queue_name "FAIRSHARE=USER_SHARES[[user_name, share]] "`

For example, for the existing `lsb.queues` configuration:

```
...
Begin queue
QUEUE_NAME=my_queue
FAIRSHARE=USER_SHARES[ [tina, 10] [default, 3]]
End Queue
...
```

Add a user group and share:

```
bconf addmember queue=my_queue "FAIRSHARE=USER_SHARES[[ug1, 10]]"
```

```
bconf: Request for queue <my_queue> accepted
```

Once accepted by `bconf`, the new share definition appears in `bqueue -l` output:

```
bqueues -l my_queue
```

```
...
USER_SHARES: [tina, 10] [ug1, 10] [default, 3]
```

```
...
```

Important:

If `USER_SHARES=[]` for the fairshare queue and a share value is added to `USER_SHARES`, the value `[default, 1]` is also added automatically.

For example, for the `lsb.queues` configuration:

```
...
Begin queue
QUEUE_NAME=queue16
FAIRSHARE=USER_SHARES[ ]
End Queue
...
```

Add a share value:

```
bconf addmember queue=queue16 "FAIRSHARE=USER_SHARES[[user3, 10]]"
bconf: Request for queue <queue16> accepted
```

Once accepted by `bconf`, the new share definition appears in `bqueue -l` output:

```
bqueues -l queue16
...
USER_SHARES: [user3, 10] [default, 1]
...
```

Add consumers to a guaranteed resource pool

Change the `DISTRIBUTION` of a guaranteed resource pool in `lsb.resources` using live reconfiguration.

1. Run `bconf addmember gpool=pool_name "DISTRIBUTION=([SLA, share])"`

For example, for the existing `lsb.resources` configuration:

```
...
Begin GuaranteedResourcePool
NAME=my_pool
DISTRIBUTION=( [ SLA1, 10 ] [ SLA2, 30 ] )
...
End GuaranteedResourcePool
...
```

Add another SLA and share:

```
bconf addmember gpool=my_pool "DISTRIBUTION=( [ SLA3, 10 ] )"
bconf: Request for gpool <my_pool> accepted
```

Once accepted by `bconf`, the new share definition appears in `bqueue -l` output:

```
bresources -gl my_pool
GUARANTEED RESOURCE POOL: my_pool
TYPE: slots
DISTRIBUTION: [ SLA1, 10 ] [ SLA2, 30 ] [ SLA3, 10 ]
...
```

Note:

An SLA is neither a user group nor a host group. Don't use `bconf` to update an SLA.

For more about guaranteed resource pools see [About guaranteed resource pools](#) on page 506

View bconf records

All successful and partially successful `bconf` requests are recorded in the history file `liveconf.hist` located under `$LSB_SHAREDIR/cluster_name/logdir`.

1. Run `bconf hist`.

All bconf requests made by the current user are displayed.

For example:

```
bconf hist
TIME          OBJECT    NAME    ACTION    USER    IMPACTED_OBJ
Nov 9 15:19:46 2009 limit    aaa     update    liam    limit=aaa
Nov 9 15:19:28 2009 queue    normal  update    liam    queue=normal
```

View bconf records for a specific configuration file

1. Run `bconf hist -f config_file`

where *config_file* is one of `lsb.resources`, `lsb.queues`, `lsb.users`, `lsb.hosts`, `lsf.cluster`, *clustername*, or `lsb.serviceclasses`.

All entries in the bconf history file which changed the specified configuration file are listed. This includes changes made directly, such as changing a limit, and indirectly, such as deleting the usergroup which must then be removed from the limit.

For example:

```
bconf hist -u all -f lsb.resources
TIME          OBJECT    NAME    ACTION    USER    IMPACTED_OBJ
Nov 9 15:19:50 2009 limit    aaa     create    robby    limit=aaa
Nov 9 15:19:46 2009 limit    aaa     update    liam     limit=aaa
Nov 9 15:19:37 2009 usergroup ug1     delete    robby    queue=normal owners*
                                         limit=bbb
                                         usergroup=ug1
```

View bconf records for a specific type of object

1. Run `bconf hist -o object_type`

where *object_type* is one of: `user`, `usergroup`, `host`, `hostgroup`, `queue`, `limit`, `gpool`

All entries in the bconf history file which changed the specified object are listed.

For example:

```
bconf hist -u all -o queue
TIME          OBJECT    NAME    ACTION    USER    IMPACTED_OBJ
Nov 9 15:19:28 2009 queue    normal  update    liam     queue=normal
Nov 9 15:19:37 2009 usergroup ug1     delete    robby    queue=normal owners*
                                         limit=bbb
                                         usergroup=ug1
```

Merge configuration files

Any changes made to configuration files using the bconf command result in changed configuration files written to the directory set by `LSF_LIVE_CONFDIR` in `lsf.conf`. LSF restart and reconfig uses configuration files in `LSF_LIVE_CONFDIR` if they exist.

Make live reconfiguration changes permanent by copying changed configuration files into the `LSF_CONFDIR` directory.

Important:

Remove `LSF_LIVE_CONFDIR` configuration files or merge files into `LSF_CONFDIR` before disabling bconf, upgrading LSF, applying patches to LSF, or adding server hosts.

1. Locate the live reconfiguration directory set in `LSF_LIVE_CONFDIR` in `lsf.conf`.

The bconf command can result in updated copies of the following configuration files:

- `lsb.resources`
 - `lsb.queues`
 - `lsb.users`
 - `lsb.hosts`
 - `lsf.cluster.clustername`
2. Copy any existing configuration files from `LSF_LIVE_CONFDIR` to the main configuration file directory set by `LSF_CONFDIR` in `lsf.conf`.
 3. Delete configuration files from `LSF_LIVE_CONFDIR`.

Running `badm n mbdrestart` or `lsadm n reconfi g now`, `LSF_LIVE_CONFDIR` is empty, and the configuration files found in `LSF_CONFDIR` are used.

Platform LSF Daemon Startup Control

The LSF daemon startup control feature allows you to specify a list of user accounts other than `root` that can start LSF daemons on UNIX hosts. This feature also enables UNIX and Windows users to bypass the additional login required to start `res` and `sbatchd` when the EGO Service Controller (EGOSC) is configured to control LSF daemons; bypassing the EGO administrator login enables the use of scripts to automate system startup.

About Platform LSF daemon startup control

Startup by users other than root (UNIX only)

On UNIX hosts, by default only root can manually start LSF daemons. To manually start LSF daemons, a user runs the commands `lsadmin` and `badmi`, which have been installed as `setuid root`. The LSF daemon startup control feature allows you to specify a list of user accounts that are allowed to run the commands `lsadmin` and `badmi` to start LSF daemons. The list is defined in the file `lsf.sudoers`.

On Windows hosts, the `Platform Services Admin` group identifies the user accounts that can start and shut down LSF daemons.

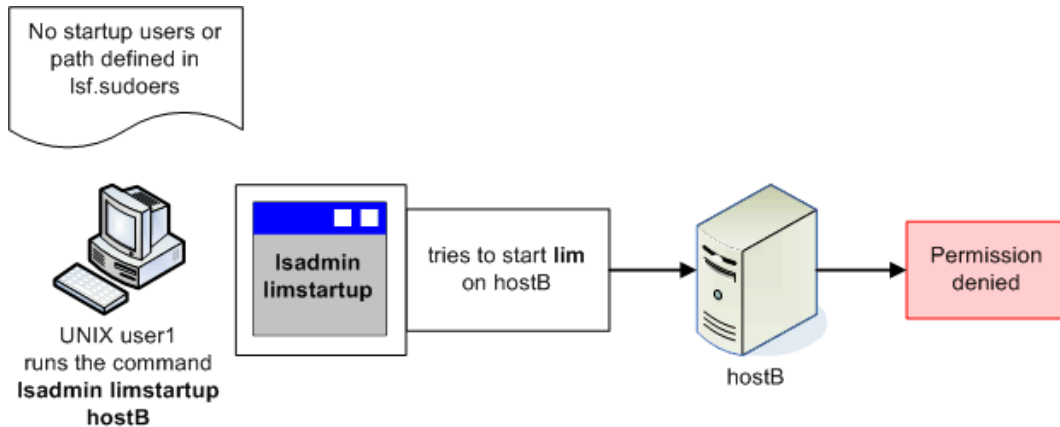


Figure 1: Default behavior (feature not enabled)

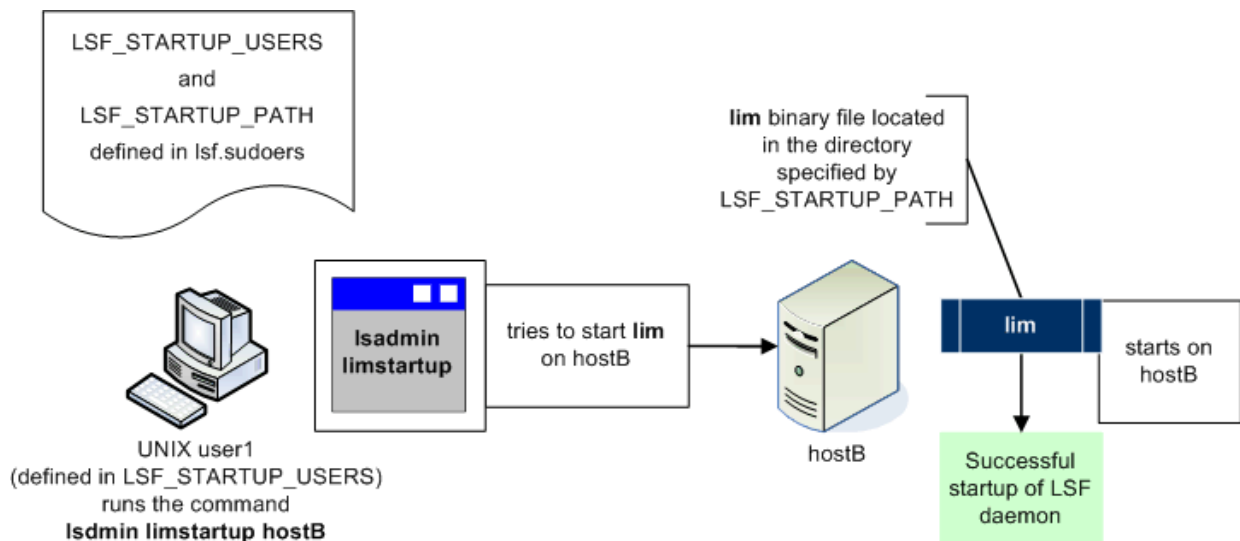


Figure 2: With Platform LSF daemon startup control enabled

EGO administrator login bypass

If the EGO Service Controller (EGOSC) is configured to control LSF daemons, EGO will automatically restart the `res` and `sbatchd` daemons unless a user has manually shut them down. When manually starting a `res` or `sbatchd` daemon that EGO has not yet started, the user who invokes `lsadmin` or

`badmin` is prompted to enter EGO administrator credentials. You can configure LSF to bypass this step by specifying the EGO administrator credentials in the file `lsf.sudoers`.

In the following illustrations, an authorized user is either a UNIX user listed in the `LSF_STARTUP_USERS` parameter or a Windows user with membership in the `Platform services admin` group.

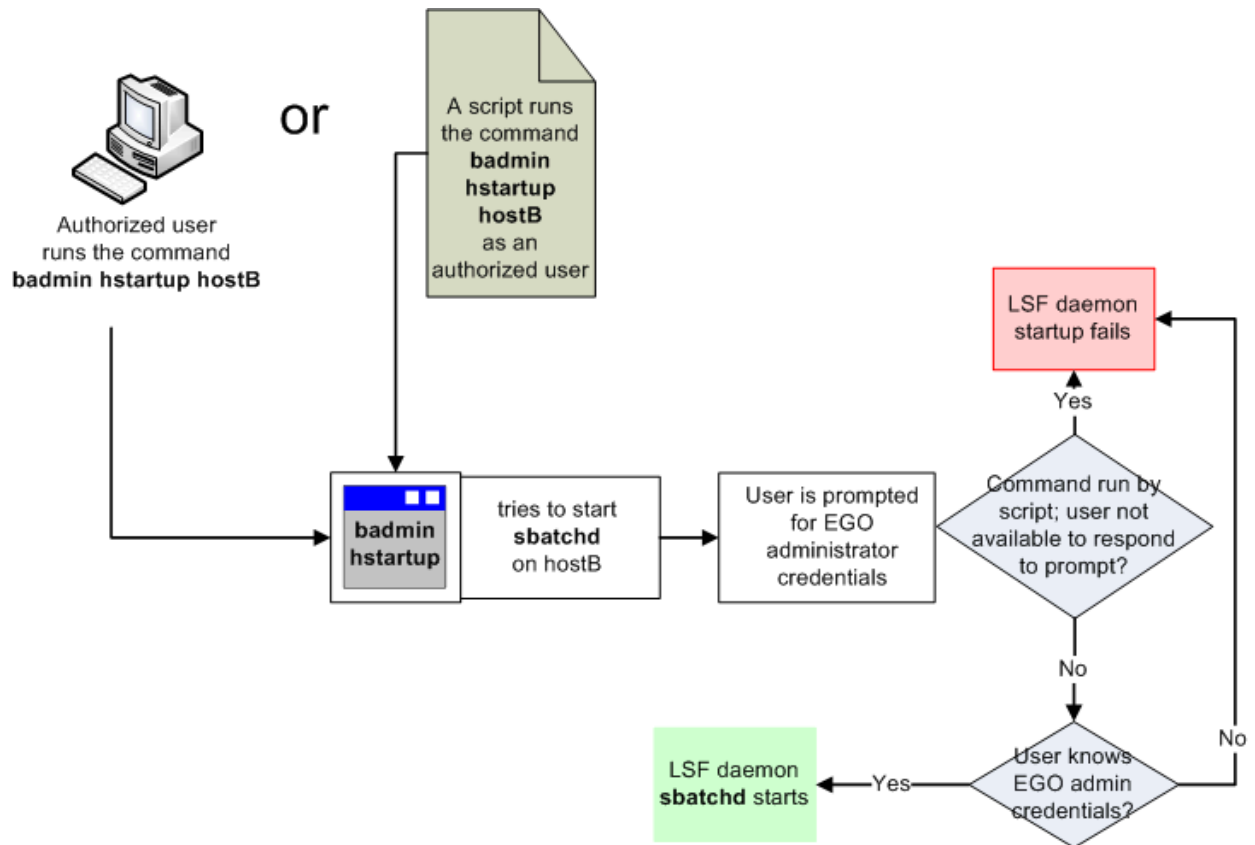


Figure 3: EGO administrator login bypass not enabled

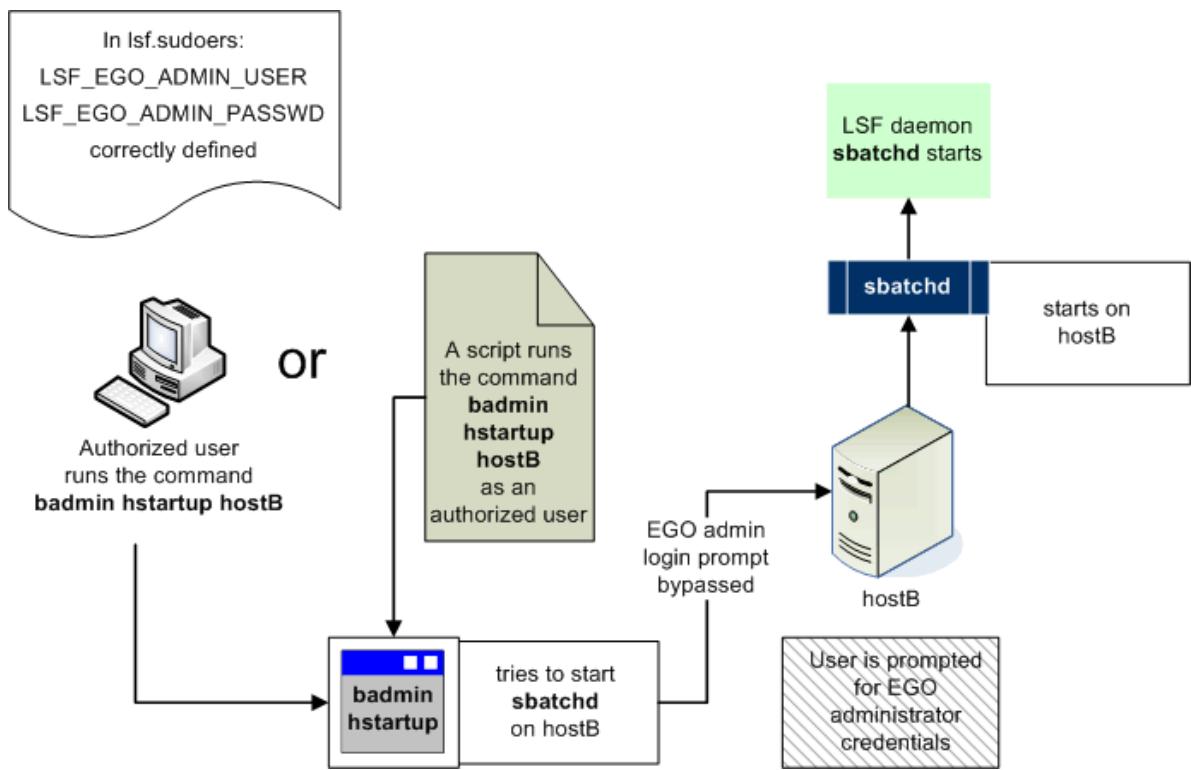


Figure 4: With EGO administrator login bypass enabled

Scope

Applicability	Details
Operating system	<ul style="list-style-type: none">• UNIX hosts only within a UNIX-only or mixed UNIX/Windows cluster: Startup of LSF daemons by users other than <code>root</code>.• UNIX and Windows: EGO administrator login bypass.
Dependencies	<ul style="list-style-type: none">• For startup of LSF daemons by users other than <code>root</code>:<ul style="list-style-type: none">• You must define both a list of users <i>and</i> the absolute path of the directory that contains the LSF daemon binary files.• The commands <code>lsadmi n</code> and <code>badmi n</code> must be installed as <code>setuid root</code>.• For EGO administrator login bypass, the default <code>Admi n</code> EGO cluster administrator account must be defined.
Limitations	<ul style="list-style-type: none">• Startup of LSF daemons by users other than <code>root</code> applies only to the following <code>lsadmi n</code> and <code>badmi n</code> subcommands:<ul style="list-style-type: none">• <code>badmi n hstartup</code>• <code>lsadmi n limstartup</code>• <code>lsadmi n resstartup</code>

Configuration to enable Platform LSF daemon startup control

Startup by users other than root (UNIX-only)

The LSF daemon startup control feature is enabled for UNIX hosts by defining the LSF_STARTUP_USERS and LSF_STARTUP_PATH parameters in the `lsf.sudoers` file. Permissions for `lsf.sudoers` must be set to 600. For Windows hosts, this feature is already enabled at installation when the `Platform services admin` group is defined.

Configuration file	Parameter and syntax	Default behavior
lsf.sudoers	LSF_STARTUP_USERS=all_admins	<ul style="list-style-type: none">Enables LSF daemon startup by users other than root when LSF_STARTUP_PATH is also defined.Allows all UNIX users defined as LSF administrators in the file <code>lsf.cluster.cluster_name</code> to start LSF daemons as root by running the <code>lsadmin</code> and <code>badmin</code> commands.Not recommended due to the security risk of a non-root LSF administrator adding to the list of administrators in the <code>lsf.cluster.cluster_name</code> file.Not required for Windows hosts because all users with membership in the <code>Platform services admin</code> group can start LSF daemons.
	<div>LSF_STARTUP_USERS="user_name1 user_name2 ..."</div> <div>LSF_STARTUP_USERS=user_name</div>	<ul style="list-style-type: none">Enables LSF daemon startup by users other than root when LSF_STARTUP_PATH is also defined.Allows the specified user accounts to start LSF daemons as root by running the <code>lsadmin</code> and <code>badmin</code> commands.Specify only cluster administrator accounts; if you add a non-administrative user, the user can start—but not shut down—LSF daemons.Separate multiple user names with a space.For a single user, do not use quotation marks.

Configuration file	Parameter and syntax	Default behavior
	<code>LSF_STARTUP_PATH=path</code>	<ul style="list-style-type: none"> Enables LSF daemon startup by users other than root when <code>LSF_STARTUP_USERS</code> is also defined. Specifies the directory that contains the LSF daemon binary files. LSF daemons are usually installed in the path specified by the <code>LSF_SERVERDIR</code> parameter defined in the <code>cskr.c.lsf</code>, <code>profile.lsf</code>, or <code>lsf.conf</code> files. <hr/> <p>Important:</p> <p>For security reasons, you should move the LSF daemon binary files to a directory other than <code>LSF_SERVERDIR</code> or <code>LSF_BINDIR</code>. The user accounts specified by <code>LSF_STARTUP_USERS</code> can start any binary in the <code>LSF_STARTUP_PATH</code>.</p>

EGO administrator login bypass

For both UNIX and Windows hosts, you can bypass the EGO administrator login for `res` and `sbatchd` by defining the parameters `LSF_EGO_ADMIN_USER` and `LSF_EGO_ADMIN_PASSWORD` in the `lsf.sudoers` file.

Configuration file	Parameter and syntax	Default behavior
lsf.sudoers	LSF_EGO_ADMIN_USER= <i>Admin</i>	<ul style="list-style-type: none"> Enables a user or script to bypass the EGO administrator login prompt when LSF_EGO_ADMIN_PASSWD is also defined. Applies only to startup of res or sbatchd. Specify the <i>Admin</i> EGO cluster administrator account.
	LSF_EGO_ADMIN_PASSWD= <i>password</i>	<ul style="list-style-type: none"> Enables a user or script to bypass the EGO administrator login prompt when LSF_EGO_ADMIN_USER is also defined. Applies only to startup of res or sbatchd. Specify the password for the <i>Admin</i> EGO cluster administrator account.

Platform LSF daemon startup control behavior

This example illustrates how LSF daemon startup control works when configured for UNIX hosts in a cluster with the following characteristics:

- The cluster contains both UNIX and Windows hosts
- The UNIX account user1 is mapped to the Windows account BUSINESS\user1 by enabling the UNIX/Windows user account mapping feature
- The account BUSINESS\user1 is a member of the Platform services admin group
- In the file `lsf.sudoers`:

```
LSF_STARTUP_USERS="user1 user2 user3"
LSF_STARTUP_PATH=LSF_TOP/8.0/linux2.4-glibc2.3-x86/etc
LSF_EGO_ADMIN_USER=Admin
LSF_EGO_ADMIN_PASSWD=Admin
```

Note:

You should change the Admin user password immediately after installation using the command `egosh user modify`.

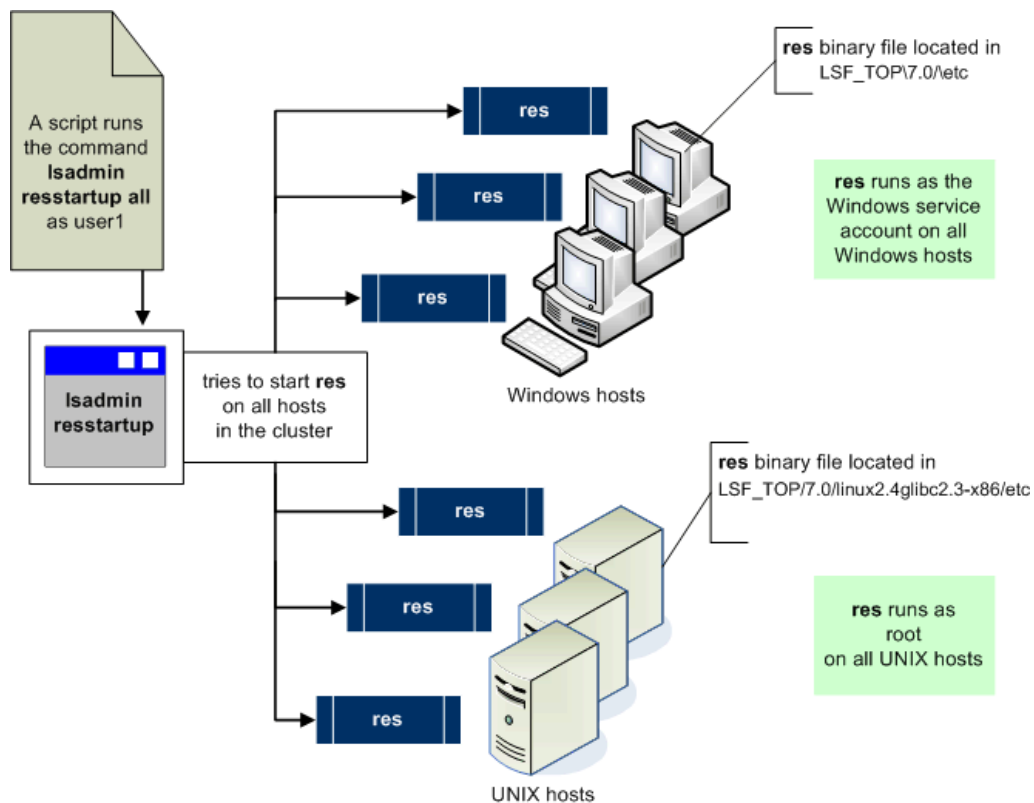


Figure 5: Example of Platform LSF daemon startup control

Configuration to modify Platform LSF daemon startup control

Not applicable: There are no parameters that modify the behavior of this feature.

Platform LSF daemon startup control commands

Commands for submission

Command	Description
N/A	<ul style="list-style-type: none"> This feature does not directly relate to job submission.

Commands to monitor

Command	Description
bhost s	<ul style="list-style-type: none"> Displays the host status of all hosts, specific hosts, or specific host groups.
lsl oad	<ul style="list-style-type: none"> Displays host status and load information.

Commands to control

Command	Description
badmi n hstartup	<ul style="list-style-type: none"> Starts the sbat chd daemon on specific hosts or all hosts. Only root and users listed in the LSF_STARTUP_USERS parameter can successfully run this command.
lsadmi n limstartup	<ul style="list-style-type: none"> Starts the l i m daemon on specific hosts or all hosts in the cluster. Only root and users listed in the LSF_STARTUP_USERS parameter can successfully run this command.
lsadmi n resstartup	<ul style="list-style-type: none"> Starts the res daemon on specific hosts or all hosts in the cluster. Only root and users listed in the LSF_STARTUP_USERS parameter can successfully run this command.

Commands to display configuration

Command	Description
badmi n showconf	<ul style="list-style-type: none"> Displays all configured parameters and their values set in l sf . conf or ego . conf that affect mbat chd and sbat chd. Use a text editor to view other parameters in the l sf . conf or ego . conf configuration files. In a MultiCluster environment, displays the parameters of daemons on the local cluster.

Use a text editor to view the l sf . sudoers configuration file.

Working with Hosts

Host status

Host status describes the ability of a host to accept and run batch jobs in terms of daemon states, load levels, and administrative controls. The `bhosts` and `lsload` commands display host status.

bhosts

Displays the current status of the host:

STATUS	Description
ok	Host is available to accept and run new batch jobs.
unavail	Host is down, or LIM and sbatchd are unreachable.
unreach	LIM is running but sbatchd is unreachable.
closed	Host will not accept new jobs. Use <code>bhosts -l</code> to display the reasons.
unlicensed	Host does not have a valid license.

bhosts -l

Displays the closed reasons (for details, see the `bhosts` command reference). A closed host does not accept new batch jobs:

```

bhosts
HOST_NAME      STATUS      JL/U      MAX  NJOBS  RUN  SSUSP  USUSP  RSV
hostA          ok          -         55    2      2     0      0      0
hostB          closed     -         20   16     16     0      0      0
...

bhosts -l hostB
HOST hostB
STATUS      CPUF  JL/U      MAX  NJOBS  RUN  SSUSP  USUSP  RSV  DISPATCH_WINDOW
closed_Adm  23.10 -         55    2      2     0      0      0      -
CURRENT LOAD USED FOR SCHEDULING:
          r15s  r1m  r15m  ut    pg    io  ls    it    tmp  swp  mem
Total      1.0 -0.0 -0.0  4%   9.4  148  2     3  4231M 698M 233M
Reserved    0.0  0.0  0.0  0%   0.0   0    0     0    0M    0M    0M
LOAD THRESHOLD USED FOR SCHEDULING:
          r15s  r1m  r15m  ut    pg    io  ls    it    tmp  swp  mem
loadSched  -    -    -    -    -    -    -    -    -    -    -
loadStop   -    -    -    -    -    -    -    -    -    -    -
          cpuspeed  bandwidth
loadSched   -        -
loadStop    -        -

```

lsload

Displays the current state of the host:

Status	Description
ok	Host is available to accept and run batch jobs and remote tasks.
- ok	LIM is running but RES is unreachable.
busy	Does not affect batch jobs, only used for remote task placement (i.e., <code>lsrun</code>). The value of a load index exceeded a threshold (configured in <code>lsf.cluster.cluster_name</code> , displayed by <code>lshosts -l</code>). Indices that exceed thresholds are identified with an asterisk (*).

Status	Description
lockW	Does not affect batch jobs, only used for remote task placement (i.e., lsrunc). Host is locked by a run window (configured in lsf.cluster.cluster_name, displayed by lshosts -l).
lockU	Will not accept new batch jobs or remote tasks. An LSF administrator or root explicitly locked the host using lsadmin limlock, or an exclusive batch job (bsub -x) is running on the host. Running jobs are not affected. Use lsadmin limunlock to unlock LIM on the local host.
unavail	Host is down, or LIM is unavailable.
unlicensed	The host does not have a valid license.

lsload

HOST_NAME	status	r15s	r1m	r15m	ut	pg	ls	it	tmp	swp	mem
hostA	ok	0.0	0.0	0.0	4%	0.4	0	4316	10G	302M	252M
hostB	ok	1.0	0.0	0.0	4%	8.2	2	14	4231M	698M	232M
...											

How LIM determines host models and types

The LIM (load information manager) daemon/service automatically collects information about hosts in an LSF cluster, and accurately determines running host models and types. At most, 1024 model types can be manually defined in `lsf.shared`.

If `lsf.shared` is not fully defined with all known host models and types found in the cluster, LIM attempts to match an unrecognized running host to one of the models and types that is defined.

LIM supports both exact matching of host models and types, and "fuzzy" matching, where an entered host model name or type is slightly different from what is defined in `lsf.shared` (or in `ego.shared` if EGO is enabled in the LSF cluster).

How does "fuzzy" matching work?

LIM reads host models and types that have been manually configured in `lsf.shared`. The format for entering host models and types is *model_bogomips_architecture* (for example, **x15_4604_Opteron****tmProcessor142**, **IA64_2793**, or **SUNWUltra510_360_sparc**). Names can be up to 64 characters long.

When LIM attempts to match running host model with what is entered in `lsf.shared`, it first attempts an exact match, then proceeds to make a fuzzy match.

How LIM attempts to make matches

Architecture name of running host	What the lim reports	Additional information about the lim process
Same as definition in <code>lsf.shared</code> (exact match)	Reports the reference index of exact match	LIM detects an exact match between model and input architecture string

Architecture name of running host	What the lim reports	Additional information about the lim process
Similar to what is defined in lsf.shared (fuzzy match)	Reports fuzzy match based on detection of 1 or 2 fields in the input architecture string	<ul style="list-style-type: none"> For input architecture strings with only one field, if LIM cannot detect an exact match for the input string, then it reports the <i>best match</i>. A best match is a model field with the most characters shared by the input string. For input architecture strings with two fields: <ol style="list-style-type: none"> If LIM cannot detect an exact match, it attempts to find a best match by identifying the <i>model</i> field with the most characters that match the input string LIM then attempts to find the best match on the <i>bogomips</i> field For architecture strings with three fields: <ol style="list-style-type: none"> If LIM cannot detect an exact match, it attempts to find a best match by identifying the <i>model</i> field with the most characters that match the input string After finding the best match for the model field, LIM attempts to find the best match on the <i>architecture</i> field LIM then attempts to find the closest match on the <i>bogomips</i> field, with wildcards supported (where the <i>bogomips</i> field is a wildcard)
Has an illegal name	Reports default host model	An illegal name is one that does not follow the permitted format for entering an architecture string where the first character of the string is not an English-language character.

View host information

LSF uses some or all of the hosts in a cluster as execution hosts. The host list is configured by the LSF administrator.

- Use the `bhosts` command to view host information.
- Use the `lsl load` command to view host load information.

To view...	Run...
All hosts in the cluster and their status	<code>bhosts</code>
Condensed host groups in an uncondensed format	<code>bhosts -X</code>
Detailed server host information	<code>bhosts -l</code> and <code>lshosts -l</code>
Host load by host	<code>lsl load</code>
Host architecture information	<code>lshosts</code>
Host history	<code>badmi n hhi st</code>
Host model and type information	<code>lsi nfo</code>
Job exit rate and load for hosts	<code>bhosts -l</code> and <code>bhosts -x</code>
Dynamic host information	<code>lshosts</code>

View all hosts in the cluster and their status

1. Run `bhosts` to display information about all hosts and their status.

`bhosts` displays condensed information for hosts that belong to condensed host groups. When displaying members of a condensed host group, `bhosts` lists the host group name instead of the name of the individual host. For example, in a cluster with a condensed host group (groupA), an uncondensed host group (groupB containing hostC and hostE), and a host that is not in any host group (hostF), `bhosts` displays the following:

```
bhosts
HOST_NAME      STATUS      JL/U      MAX  NJOBS      RUN  SSUSP  USUSP      RSV
groupA          ok           5         8         4         2        0        1         1
hostC           ok           -         3         0         0        0        0         0
hostE           ok           2         4         2         1        0        0         1
hostF           ok           -         2         2         1        0        1         0
```

Define condensed host groups in the Host Groups section of `lsb. hosts`.

View uncondensed host information

1. Run `bhosts -X` to display all hosts in an uncondensed format, including those belonging to condensed host groups:

```
bhosts -X
HOST_NAME      STATUS      JL/U      MAX  NJOBS      RUN  SSUSP  USUSP      RSV
hostA          ok           2         2         0         0        0        0         0
hostD          ok           2         4         2         1        0        0         1
hostB          ok           1         2         2         1        0        1         0
hostC          ok           -         3         0         0        0        0         0
hostE          ok           2         4         2         1        0        0         1
```

hostF	ok	-	2	2	1	0	1	0
-------	----	---	---	---	---	---	---	---

View detailed server host information

1. Run `bhosts -l host_name` and `lshosts -l host_name` to display all information about each server host such as the CPU factor and the load thresholds to start, suspend, and resume jobs:

bhosts -l hostB

```
HOST hostB
STATUS CPUF JL/U MAX NJOBS RUN SSUSP USUSP RSV DISPATCH_WINDOWS
ok 20.20 - - 0 0 0 0 -
CURRENT LOAD USED FOR SCHEDULING:
r15s r1m r15m ut pg io ls it tmp swp mem
Total 0.1 0.1 0.1 9% 0.7 24 17 0 394M 396M 12M
Reserved 0.0 0.0 0.0 0% 0.0 0 0 0 0M 0M 0M
LOAD THRESHOLD USED FOR SCHEDULING:
r15s r1m r15m ut pg io ls it tmp swp mem
loadSched - - - - - - - - - - -
loadStop - - - - - - - - - - -

cpuspeed bandwidth
loadSched - -
loadStop - -
```

lshosts -l hostB

```
HOST_NAME: hostB
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 PC6000 116.1 2 1 2016M 1983M 72917M 0 Yes 1 2 2

RESOURCES: Not defined
RUN_WINDOWS: (always open)

LICENSES_ENABLED: (LSF_Base LSF_Manager LSF_MultiCluster)
LICENSE_NEEDED: Class(E)

LOAD_THRESHOLDS:
r15s r1m r15m ut pg io ls it tmp swp mem
- 1.0 - - - - - - - - 4M
```

View host load by host

The `lsl load` command reports the current status and load levels of hosts in a cluster. The `lshosts -l` command shows the load thresholds.

The `lsmom` command provides a dynamic display of the load information. The LSF administrator can find unavailable or overloaded hosts with these tools.

1. Run `lsl load` to see load levels for each host:

lsl load

HOST_NAME	status	r15s	r1m	r15m	ut	pg	ls	it	tmp	swp	mem
hostD	ok	1.3	1.2	0.9	92%	0.0	2	20	5M	148M	88M
hostB	-ok	0.1	0.3	0.7	0%	0.0	1	67	45M	25M	34M
hostA	busy	8.0	*7.0	4.9	84%	4.6	6	17	1M	81M	27M

The first line lists the load index names, and each following line gives the load levels for one host.

View host architecture (type and model) information

The `lshosts` command displays configuration information about hosts. All these parameters are defined by the LSF administrator in the LSF configuration files, or determined by the LIM directly from the system.

Host types represent binary compatible hosts; all hosts of the same type can run the same executable. Host models give the relative CPU performance of different processors.

1. Run `lshosts` to see configuration information about hosts:

```

lshosts
HOST_NAME  type      model    cpuf  ncpus  maxmem  maxswp  server  RESOURCES
hostD      SUNSOL   SunSparc 6.0   1      64M     112M    Yes     (solaris cserver)
hostM      RS6K     IBM350   7.0   1      64M     124M    Yes     (cserver aix)
hostC      SGI 6    R10K     14.0  16     1024M   1896M   Yes     (irix cserver)
hostA      HPPA     HP715    6.0   1      98M     200M    Yes     (hpux fserver)

```

In the above example, the host type `SUNSOL` represents Sun SPARC systems running Solaris, and `SGI 6` represents an SGI server running IRIX 6. The `lshosts` command also displays the resources available on each host.

type

The host CPU architecture. Hosts that can run the same binary programs should have the same type.

An `UNKNOWN` type or model indicates the host is down, or LIM on the host is down.

When automatic detection of host type or model fails (the host type configured in `lsf.shared` cannot be found), the type or model is set to `DEFAULT`. LSF will work on the host, but a `DEFAULT` model may be inefficient because of incorrect CPU factors. A `DEFAULT` type may also cause binary incompatibility because a job from a `DEFAULT` host type can be migrated to another `DEFAULT` host type. automatic detection of host type or model has failed, and the host type configured in `lsf.shared` cannot be found.

View host history

1. Run `badmin hhist` to view the history of a host such as when it is opened or closed:

```

badmin hhist hostB
Wed Nov 20 14:41:58: Host <hostB> closed by administrator <lsf>.
Wed Nov 20 15:23:39: Host <hostB> opened by administrator <lsf>.

```

View host model and type information

1. Run `lsinfo -m` to display information about host models that exist in the cluster:

```

lsinfo -m
MODEL_NAME      CPU_FACTOR      ARCHITECTURE
PC1133          23.10           x6_1189_PentiumIII Coppermine
HP9K735         4.50            HP9000735_125
HP9K778         5.50            HP9000778
Ultra5S         10.30           SUNWUltra510_270_sparcv9
Ultra2          20.20           SUNWUltra2_300_sparc
Enterprise3000  20.00           SUNWUltraEnterprise_167_sparc

```

2. Run `lsinfo -M` to display all host models defined in `lsf.shared`:

```

lsinfo -M
MODEL_NAME      CPU_FACTOR      ARCHITECTURE
UNKNOWN_AUTO_DETECT 1.00           UNKNOWN_AUTO_DETECT
DEFAULT         1.00
LINUX133        2.50           x586_53_Pentium75
PC200           4.50           i86pc_200
Intel_IA64      12.00          ia64
Ultra5S         10.30          SUNWUltra5_270_sparcv9
PowerPC_G4      12.00          x7400G4
HP300           1.00

```

```
SunSparc          12.00
```

- Run `lim -t` to display the type, model, and matched type of the current host. You must be the LSF administrator to use this command:

```
lim -t
Host Type           : NTX64
Host Architecture   : EM64T_1596
Physical Processors : 2
Cores per Processor : 4
Threads per Core    : 2
License Needed      : Class(B), Multi-cores
Matched Type        : NTX64
Matched Architecture : EM64T_3000
Matched Model       : Intel_EM64T
CPU Factor          : 60.0
```

View job exit rate and load for hosts

- Run `bhosts` to display the exception threshold for job exit rate and the current load value for hosts.:

In the following example, `EXIT_RATE` for hostA is configured as 4 jobs per minute. hostA does not currently exceed this rate

bhosts -l hostA

```
HOST hostA
STATUS      CPUF  JL/U    MAX  NJOBS  RUN  SSUSP  USUSP  RSV  DISPATCH_WINDOW
ok          18.60  -      1    0      0    0      0    0    -

CURRENT LOAD USED FOR SCHEDULING:
r15s  r1m  r15m  ut    pg    io    ls    it    tmp    swp    mem
Total  0.0  0.0  0.0  0%   0.0  0    1    2   646M  648M  115M
Reserved 0.0  0.0  0.0  0%   0.0  0    0    0    0M    0M    0M

share_rsrc host_rsrc
Total      3.0    2.0
Reserved   0.0    0.0

LOAD THRESHOLD USED FOR SCHEDULING:
r15s  r1m  r15m  ut    pg    io    ls    it    tmp    swp    mem
loadSched -    -    -    -    -    -    -    -    -    -    -
loadStop  -    -    -    -    -    -    -    -    -    -    -

cpuspeed  bandwidth
loadSched -    -
loadStop  -    -

THRESHOLD AND LOAD USED FOR EXCEPTIONS:
JOB_EXIT_RATE
Threshold  4.00
Load       0.00
```

- Use `bhosts -x` to see hosts whose job exit rate has exceeded the threshold for longer than `JOB_EXIT_RATE_DURATION`, and are still high. By default, these hosts are closed the next time LSF checks host exceptions and invokes `eadmin`.

If no hosts exceed the job exit rate, `bhosts -x` displays:

```
There is no exceptional host found
```

View dynamic host information

- Use `lshosts` to display information on dynamically added hosts.

An LSF cluster may consist of static and dynamic hosts. The `lshosts` command displays configuration information about hosts. All these parameters are defined by the LSF administrator in the LSF configuration files, or determined by the LIM directly from the system.

Host types represent binary compatible hosts; all hosts of the same type can run the same executable. Host models give the relative CPU performance of different processors. Server represents the type of host in the cluster. “Yes” is displayed for LSF servers, “No” is displayed for LSF clients, and “Dyn” is displayed for dynamic hosts.

For example:

```
lshosts
HOST_NAME  type    model  cpuf  ncpus  maxmem  maxswp  server  RESOURCES
hostA      SOL64  Ultra60F 23.5    1    64M    112M    Yes    ()
hostB      LINUX86 Opteron8 60.0    1    94M    168M    Dyn    ()
```

In the above example, host A is a static host while host B is a dynamic host.

Control hosts

Hosts are opened and closed by:

- an LSF Administrator or root issuing a command
- configured dispatch windows

Close a host

1. Run `badmi n hcl ose`:

```
badmin hclose hostB
Close <hostB> ..... done
```

If the command fails, it may be because the host is unreachable through network problems, or because the daemons on the host are not running.

Open a host

1. Run `badmi n hopen`:

```
badmin hopen hostB
Open <hostB> ..... done
```

Configure dispatch windows

A dispatch window specifies one or more time periods during which a host will receive new jobs. The host will not receive jobs outside of the configured windows. Dispatch windows do not affect job submission and running jobs (they are allowed to run until completion). By default, dispatch windows are not configured.

To configure dispatch windows:

1. Edit `lsb. hosts`.
2. Specify one or more time windows in the `DISPATCH_WINDOW` column:

Begin Host	HOST_NAME	rlm	pg	ls	tmp	DISPATCH_WINDOW
...	hostB	3.5/4.5	15/	12/15	0	(4:30-12:00)
...						
End Host						

3. Reconfigure the cluster:
 - a) Run `lsadmi n reconfi g` to reconfigure LIM.
 - b) Run `badmi n reconfi g` to reconfigure `mbat chd`.
4. Run `bhosts -l` to display the dispatch windows.

Log a comment when closing or opening a host

1. Use the `-C` option of `badmi n hcl ose` and `badmi n hopen` to log an administrator comment in `lsb. events`:

```
badmin hclose -C "Weekly backup" hostB
```

The comment text Weekly backup is recorded in lsb.events. If you close or open a host group, each host group member displays with the same comment string.

A new event record is recorded for each host open or host close event. For example:

```
badmin hclose -C "backup" hostA
```

followed by

```
badmin hclose -C "Weekly backup" hostA
```

generates the following records in lsb.events:

```
"HOST_CTRL" "7.0 1050082346 1 "hostA" 32185 "lsfadmin" "backup"
"HOST_CTRL" "7.0 1050082373 1 "hostA" 32185 "lsfadmin" "Weekly backup"
```

2. Use badmin hist or badmin hhist to display administrator comments for closing and opening hosts:

```
badmin hhist
```

```
Fri Apr 4 10:35:31: Host <hostB> closed by administrator
<lsfadmin> Weekly backup.
```

bhosts -l also displays the comment text:

```
bhosts -l
```

```
HOST hostA
STATUS CPUF JL/U MAX NJOBS RUN SSUSP USUSP RSV DISPATCH_WINDOW
closed_Adm 1.00 - - 0 0 0 0 0 -

CURRENT LOAD USED FOR SCHEDULING:
          r15s r1m r15m ut pg io ls it tmp swp mem
Total      0.0 0.0 0.0 2% 0.0 64 2 11 7117M 512M 432M
Reserved   0.0 0.0 0.0 0% 0.0 0 0 0 0M 0M 0M

LOAD THRESHOLD USED FOR SCHEDULING:
          r15s r1m r15m ut pg io ls it tmp swp mem
loadSched - - - - - - - - - - -
loadStop - - - - - - - - - - -

          cpuspeed bandwidth
loadSched - -
loadStop - -

THRESHOLD AND LOAD USED FOR EXCEPTIONS:
          JOB_EXIT_RATE
Threshold 2.00
Load      0.00
ADMIN ACTION COMMENT: "Weekly backup"
```

How events are displayed and recorded in MultiCluster lease model

In the MultiCluster resource lease model, host control administrator comments are recorded only in the lsb.events file on the local cluster. badmin hist and badmin hhist display only events that are recorded locally. Host control messages are not passed between clusters in the MultiCluster lease model. For example, if you close an exported host in both the consumer and the provider cluster, the host close events are recorded separately in their local lsb.events.

Add a host

- You use the `lsfinstall` command to add a host to an LSF cluster.

Add a host of an existing type using `lsinstall`

Restriction:

`lsinstall` is not compatible with clusters installed with `lsfsetup`. To add a host to a cluster originally installed with `lsfsetup`, you must upgrade your cluster.

- Verify that the host type already exists in your cluster:
 - Log on to any host in the cluster. You do not need to be root.
 - List the contents of the `LSF_TOP/8.0` directory and confirm there is already a subdirectory with the name of the host type.

The default `LSF_TOP/8.0` directory is `/usr/share/lsf/8.0`.

- Add the host information to `lsf.cluster.cluster_name`:
 - Log on to the LSF master host as root.
 - Edit `LSF_CONFDIR/lsf.cluster.cluster_name`, and specify the following in the Host section:
 - The name of the host.
 - The model and type, or specify `!` to automatically detect the type or model.
 - Specify **1** for LSF server or **0** for LSF client.

```
Begin Host
HOSTNAME  model  type      server  rlm  mem  RESOURCES  REXPRI
hosta     !      SUNSOL6   1       1.0  4    ()          0
hostb     !      SUNSOL6   0       1.0  4    ()          0
hostc     !      HPPA1132  1       1.0  4    ()          0
hostd     !      HPPA1164  1       1.0  4    ()          0
End Host
```

- Save your changes.
- Run `lsadmin reconfig` to reconfigure LIM.
 - Run `badmin mbdrestart` to restart `mbatchd`.
 - Run `hostsetup` to set up the new host and configure the daemons to start automatically at boot from `/usr/share/lsf/8.0/install`:

```
./hostsetup --top="/usr/share/lsf" --boot="y"
```

- Start LSF on the new host:

```
lsadmin limstartup
lsadmin resstartup
badmin hstartup
```

- Run `bhosts` and `lshosts` to verify your changes.

Add a host of a new type using `lsinstall`

Restriction:

lsfinstall is not compatible with clusters installed with lssetup. To add a host to a cluster originally installed with lssetup, you must upgrade your cluster.

1. Verify that the host type does not already exist in your cluster:
 - a) Log on to any host in the cluster. You do not need to be root.
 - b) List the contents of the LSF_TOP/8.0 directory. The default is `/usr/share/lsf/8.0`. If the host type currently exists, there will be a subdirectory with the name of the host type.
2. Get the LSF distribution tar file for the host type you want to add.
3. Log on as root to any host that can access the LSF install directory.
4. Change to the LSF install directory. The default is

```
/usr/share/lsf/8.0/install
```

5. Edit `install.config`:
 - a) For LSF_TARDIR, specify the path to the tar file. For example:

```
LSF_TARDIR="/usr/share/lsf_distribution/8.0"
```
 - b) For LSF_ADD_SERVERS, list the new host names enclosed in quotes and separated by spaces. For example:

```
LSF_ADD_SERVERS="hosta hostb"
```
 - c) Run `./lsfinstall -f install.config`. This automatically creates the host information in `lsf.cluster.cluster_name`.
6. Run `lsadmin reconfig` to reconfigure LIM.
7. Run `badmin reconfig` to reconfigure mbatchd.
8. Run `hostsetup` to set up the new host and configure the daemons to start automatically at boot from `/usr/share/lsf/8.0/install`:

```
./hostsetup --top="/usr/share/lsf" --boot="y"
```

9. Start LSF on the new host:

```
lsadmin limstartup
lsadmin resstartup
badmin hstartup
```

10. Run `bhosts` and `lshosts` to verify your changes.

Remove a host

Removing a host from LSF involves preventing any additional jobs from running on the host, removing the host from LSF, and removing the host from the cluster.

Caution:

Never remove the master host from LSF. If you want to remove your current default master from LSF, change `lsf.cluster.cluster_name` to assign a different default master host. Then remove the host that was once the master host.

1. Log on to the LSF host as root.
2. Run `badmi n hcl ose` to close the host. This prevents jobs from being dispatched to the host and allows running jobs to finish.
3. Stop all running daemons manually.
4. Remove any references to the host in the Host section of `LSF_CONFDIR/lsf.cluster.cluster_name`.
5. Remove any other references to the host, if applicable, from the following LSF configuration files:
 - `LSF_CONFDIR/lsf.shared`
 - `LSB_CONFDIR/cluster_name/configdir/l sb. hosts`
 - `LSB_CONFDIR/cluster_name/configdir/l sb. queues`
 - `LSB_CONFDIR/cluster_name/configdir/l sb. resources`
6. Log off the host to be removed, and log on as root or the primary LSF administrator to any other host in the cluster.
7. Run `lsadmi n reconfi g` to reconfigure LIM.
8. Run `badmi n mbdrestart` to restart `mbatchd`.
9. If you configured LSF daemons to start automatically at system startup, remove the LSF section from the host's system startup files.
10. If any users of the host use `l st csh` as their login shell, change their login shell to `tcsh` or `csh`. Remove `l st csh` from the `/etc/shell s` file.

Remove a host from master candidate list

You can remove a host from the master candidate list so that it can no longer be the master should failover occur. You can choose to either keep it as part of the cluster or remove it.

1. Shut down the current LIM:

```
limshutdown host_name
```

If the host was the current master, failover occurs.

2. In `lsf.conf`, remove the host name from `LSF_MASTER_LIST`.
3. Run `lsadmin reconfig` for the remaining master candidates.
4. If the host you removed as a master candidate still belongs to the cluster, start up the LIM again:

```
limstartup host_name
```

Add hosts dynamically

By default, all configuration changes made to LSF are static. To add or remove hosts within the cluster, you must manually change the configuration and restart all master candidates.

Dynamic host configuration allows you to add and remove hosts without manual reconfiguration. To enable dynamic host configuration, all of the parameters described in the following table must be defined.

Parameter	Defined in ...	Description
LSF_MASTER_LIST	<code>lsf.conf</code>	Defines a list of master host candidates. These hosts receive information when a dynamic host is added to or removed from the cluster. Do not add dynamic hosts to this list, because dynamic hosts cannot be master hosts.
LSF_DYNAMIC_HOST_WAIT_TIME	<code>lsf.conf</code>	Defines the length of time a dynamic host waits before sending a request to the master LIM to add the host to the cluster.
LSF_HOST_ADDR_RANGE	<code>lsf.cluster.cluster_name</code>	Identifies the range of IP addresses for hosts that can dynamically join or leave the cluster.

Important:

If you choose to enable dynamic hosts when you install LSF, the installer adds the parameter `LSF_HOST_ADDR_RANGE` to `lsf.cluster.cluster_name` using a default value that allows any host to join the cluster. To enable security, configure `LSF_HOST_ADDR_RANGE` in `lsf.cluster.cluster_name` after installation to restrict the hosts that can join your cluster.

How dynamic host configuration works

Master LIM The master LIM runs on the master host for the cluster. The master LIM receives requests to add hosts, and tells the master host candidates defined by the parameter `LSF_MASTER_LIST` to update their configuration information when a host is dynamically added or removed.

Upon startup, both static and dynamic hosts wait to receive an acknowledgement from the master LIM. This acknowledgement indicates that the master LIM has added the host to the cluster. Static hosts normally receive an acknowledgement because the master LIM has access to static host information in the LSF configuration files. Dynamic hosts do not receive an acknowledgement, however, until they announce themselves to the master LIM. The parameter `LSF_DYNAMIC_HOST_WAIT_TIME` in `lsf.conf` determines how long a dynamic host waits before sending a request to the master LIM to add the host to the cluster.

Master candidate LIMs The parameter `LSF_MASTER_LIST` defines the list of master host candidates. These hosts receive updated host information from the master LIM so that any master host candidate can take over as master host for the cluster.

Important:

Master candidate hosts should share LSF configuration and binaries.

Dynamic hosts cannot be master host candidates. By defining the parameter `LSF_MASTER_LIST`, you ensure that LSF limits the list of master host candidates to specific, static hosts.

mbatchd `mbat chd` gets host information from the master LIM; when it detects the addition or removal of a dynamic host within the cluster, `mbat chd` automatically reconfigures itself.

Tip:

After adding a host dynamically, you might have to wait for `mbat chd` to detect the host and reconfigure. Depending on system load, `mbat chd` might wait up to a maximum of 10 minutes before reconfiguring.

lsadmin command Use the command `lsadmin limstart up` to start the LIM on a newly added dynamic host.

Allow only certain hosts to join the cluster

By default, any host can be dynamically added to the cluster. To enable security, define `LSF_HOST_ADDR_RANGE` in `lsf.cluster.cluster_name` to identify a range of IP addresses for hosts that are allowed to dynamically join the cluster as LSF hosts. IP addresses can have either a dotted quad notation (IPv4) or IP Next Generation (IPv6) format. You can use IPv6 addresses if you define the parameter `LSF_ENABLE_SUPPORT_IPV6` in `lsf.conf`; you do not have to map IPv4 addresses to an IPv6 format.

Configure LSF to run batch jobs on dynamic hosts

Before you run batch jobs on a dynamic host, complete any or all of the following steps, depending on your cluster configuration.

1. Configure queues to accept all hosts by defining the `HOSTS` parameter in `lsb.queues` using the keyword `all`.
2. Define host groups that will accept wild cards in the Host Group section of `lsb.hosts`.

For example, define `linuxrack*` as a `GROUP_MEMBER` within a host group definition.

3. Add a dynamic host to a host group using the command `badmin hghost add`.

Change a dynamic host to a static host

If you want to change a dynamic host to a static host, first use the command `badmin hghost del` to remove the dynamic host from any host group that it belongs to, and then configure the host as a static host in `lsf.cluster.cluster_name`.

Add a dynamic host in a shared file system environment

In a shared file system environment, you do not need to install LSF on each dynamic host. The master host will recognize a dynamic host as an LSF host when you start the daemons on the dynamic host.

1. In `lsf.conf` on the master host, define the parameter `LSF_DYNAMIC_HOST_WAIT_TIME`, in seconds, and assign a value greater than zero.

LSF_DYNAMIC_HOST_WAIT_TIME specifies the length of time a dynamic host waits before sending a request to the master LIM to add the host to the cluster.

For example:

```
LSF_DYNAMIC_HOST_WAIT_TIME=60
```

2. In `lsf.conf` on the master host, define the parameter `LSF_DYNAMIC_HOST_TIMEOUT`.

`LSF_DYNAMIC_HOST_TIMEOUT` specifies the length of time (minimum 10 minutes) a dynamic host is unavailable before the master host removes it from the cluster. Each time LSF removes a dynamic host, `mbatchd` automatically reconfigures itself.

Note:

For very large clusters, defining this parameter could decrease system performance.

For example:

```
LSF_DYNAMIC_HOST_TIMEOUT=60m
```

3. In `lsf.cluster.cluster_name` on the master host, define the parameter `LSF_HOST_ADDR_RANGE`.

`LSF_HOST_ADDR_RANGE` enables security by defining a list of hosts that can join the cluster. Specify IP addresses or address ranges for hosts that you want to allow in the cluster.

Note:

If you define the parameter `LSF_ENABLE_SUPPORT_IPV6` in `lsf.conf`, IP addresses can have either a dotted quad notation (IPv4) or IP Next Generation (IPv6) format; you do not have to map IPv4 addresses to an IPv6 format.

For example:

```
LSF_HOST_ADDR_RANGE=100-110.34.1-10.4-56
```

All hosts belonging to a domain with an address having the first number between 100 and 110, then 34, then a number between 1 and 10, then, a number between 4 and 56 will be allowed access. In this example, no IPv6 hosts are allowed.

4. Log on as root to each host you want to join the cluster.
5. Source the LSF environment:
 - For `csh` or `tcsh`:


```
source LSF_TOP/conf/cshrc.lsf
```
 - For `sh`, `ksh`, or `bash`:


```
. LSF_TOP/conf/profile.lsf
```
6. Do you want LSF to start automatically when the host reboots?

- If no, go to the next step.
- If yes, run the `hostsetup` command. For example:

```
cd /usr/share/lsf/8.0/install
./hostsetup --top="/usr/share/lsf" --boot="y"
```

For complete `hostsetup` usage, enter `hostsetup -h`.

7. Use the following commands to start LSF:

```
lsadmin limstartup
lsadmin resstartup
badmin hstartup
```

Add a dynamic host in a non-shared file system environment

In a non-shared file system environment, you must install LSF binaries, a localized `lsf.conf` file, and shell environment scripts (`cskr.c.lsf` and `profile.lsf`) on each dynamic host.

Specify installation options in the `slave.config` file

All dynamic hosts are slave hosts, because they cannot serve as master host candidates. The `slave.config` file contains parameters for configuring all slave hosts.

1. Define the required parameters.

```
LSF_SERVER_HOSTS="host_name [host_name ...]"
```

```
LSF_ADMINS="user_name [ user_name ... ]"
```

```
LSF_TOP="/path"
```

2. Define the optional parameters.

```
LSF_LIM_PORT=port_number
```

Important:

If the master host does not use the default `LSF_LIM_PORT`, you must specify the same `LSF_LIM_PORT` defined in `lsf.conf` on the master host.

Add local resources on a dynamic host to the cluster

Ensure that the resource name and type are defined in `lsf.shared`, and that the `ResourceMap` section of `lsf.cluster.cluster_name` contains at least one resource mapped to at least one static host. LSF can add local resources as long as the `ResourceMap` section is defined; you do not need to map the local resources.

1. In the `slave.config` file, define the parameter `LSF_LOCAL_RESOURCES`.

For numeric resources, define name-value pairs:

```
"[resourcemap value*resource_name]"
```

For Boolean resources, the value is the resource name in the following format:

```
"[resource resource_name]"
```

For example:

```
LSF_LOCAL_RESOURCES="[resourcemap 1*verilog] [resource linux]"
```

Tip:

If `LSF_LOCAL_RESOURCES` are already defined in a local `lsf.conf` on the dynamic host, `lsfinstall` does not add resources you define in `LSF_LOCAL_RESOURCES` in `slave.config`.

When the dynamic host sends a request to the master host to add it to the cluster, the dynamic host also reports its local resources. If the local resource is already defined in `lsf.cluster.cluster_name` as default or all, it cannot be added as a local resource.

Install LSF on a dynamic host

1. Run `lsfinstall -s -f slave.config`.

`lsfinstall` creates a local `lsf.conf` for the dynamic host, which sets the following parameters:

```
LSF_CONFDIR="/path"
```

```
LSF_GET_CONF=lim
```

```
LSF_LIM_PORT=port_number (same as the master LIM port number)
```

```
LSF_LOCAL_RESOURCES="resource ..."
```

Tip:

Do not duplicate LSF_LOCAL_RESOURCES entries in `lsf.conf`. If local resources are defined more than once, only the last definition is valid.

```
LSF_SERVER_HOSTS="host_name [host_name ...]"
```

```
LSF_VERSION=8.0
```

Important:

If LSF_STRICT_CHECKING is defined in `lsf.conf` to protect your cluster in untrusted environments, and your cluster has dynamic hosts, LSF_STRICT_CHECKING must be configured in the local `lsf.conf` on all dynamic hosts.

Configure dynamic host parameters

1. In `lsf.conf` on the master host, define the parameter LSF_DYNAMIC_HOST_WAIT_TIME, in seconds, and assign a value greater than zero.

LSF_DYNAMIC_HOST_WAIT_TIME specifies the length of time a dynamic host waits before sending a request to the master LIM to add the host to the cluster.

For example:

```
LSF_DYNAMIC_HOST_WAIT_TIME=60
```

2. In `lsf.conf` on the master host, define the parameter LSF_DYNAMIC_HOST_TIMEOUT.

LSF_DYNAMIC_HOST_TIMEOUT specifies the length of time (minimum 10 minutes) a dynamic host is unavailable before the master host removes it from the cluster. Each time LSF removes a dynamic host, `mbatchd` automatically reconfigures itself.

Note:

For very large clusters, defining this parameter could decrease system performance.

For example:

```
LSF_DYNAMIC_HOST_TIMEOUT=60m
```

3. In `lsf.cluster.cluster_name` on the master host, define the parameter LSF_HOST_ADDR_RANGE.

LSF_HOST_ADDR_RANGE enables security by defining a list of hosts that can join the cluster. Specify IP addresses or address ranges for hosts that you want to allow in the cluster.

Tip:

If you define the parameter LSF_ENABLE_SUPPORT_IPV6 in `lsf.conf`, IP addresses can have either a dotted quad notation (IPv4) or IP Next Generation (IPv6) format; you do not have to map IPv4 addresses to an IPv6 format.

For example:

```
LSF_HOST_ADDR_RANGE=100-110.34.1-10.4-56
```

All hosts belonging to a domain with an address having the first number between 100 and 110, then 34, then a number between 1 and 10, then, a number between 4 and 56 will be allowed access. No IPv6 hosts are allowed.

Start LSF daemons

1. Log on as root to each host you want to join the cluster.
2. Source the LSF environment:
 - For `csh` or `tcsh`:
`source LSF_TOP/conf/cshrc.lsf`
 - For `sh`, `ksh`, or `bash`:
`. LSF_TOP/conf/profile.lsf`
3. Do you want LSF to start automatically when the host reboots?

- If no, go to the next step.
- If yes, run the `hostsetup` command. For example:
`cd /usr/share/lsf/8.0/install`
`./hostsetup --top="/usr/share/lsf" --boot="y"`

For complete `hostsetup` usage, enter `hostsetup -h`.

4. Is this the first time the host is joining the cluster?
 - If no, use the following commands to start LSF:
`lsadmin limstartup`
`lsadmin resstartup`
`badmin hstartup`
 - If yes, you must start the daemons from the local host. For example, if you want to start the daemons on `hostB` from `hostA`, use the following commands:
`rsh hostB lsadmin limstartup`
`rsh hostB lsadmin resstartup`
`rsh hostB badmin hstartup`

Remove dynamic hosts

To remove a dynamic host from the cluster:

- Set a timeout value
- Edit the `hostcache` file

Remove a host by setting a timeout value

`LSF_DYNAMIC_HOST_TIMEOUT` specifies the length of time (minimum 10 minutes) a dynamic host is unavailable before the master host removes it from the cluster. Each time LSF removes a dynamic host, `mbatchd` automatically reconfigures itself.

Note:

For very large clusters, defining this parameter could decrease system performance. If you want to use this parameter to remove dynamic hosts from a very large cluster, disable the parameter after LSF has removed the unwanted hosts.

1. In `lsf.conf` on the master host, define the parameter `LSF_DYNAMIC_HOST_TIMEOUT`.
To specify minutes rather than hours, append `m` or `M` to the value.

For example:

```
LSF_DYNAMIC_HOST_TIMEOUT=60m
```

Remove a host by editing the hostcache file

Dynamic hosts remain in the cluster unless you intentionally remove them. Only the cluster administrator can modify the hostcache file.

1. Shut down the cluster.

lsfshutdown

This shuts down LSF on all hosts in the cluster and prevents LIMs from trying to write to the hostcache file while you edit it.

2. In the hostcache file `$EGO_WORKDIR/lim/hostcache`, delete the line for the dynamic host that you want to remove.
 - If EGO is enabled, the hostcache file is in `$EGO_WORKDIR/lim/hostcache`.
 - If EGO is not enabled, the hostcache file is in `$LSB_SHAREDIR`.
3. Close the hostcache file, and then start up the cluster.

lsfrestart

Automatically detect operating system types and versions

LSF can automatically detect most operating system types and versions so that you do not need to add them to the `lsf.shared` file manually. The list of automatically detected operating systems is updated regularly.

1. Edit `lsf.shared`.
2. In the Resource section, remove the comment from the following line:

```
ostype String () () () (Operating system and version)
```
3. In `$LSF_SERVERDIR`, rename `tmp.eslim.ostype` to `eslim.ostype`.
4. Run the following commands to restart the LIM and master batch daemon:
 1. `lsadmin reconfig`
 2. `badmin mbdrestart`
5. To view operating system types and versions, run `lshosts -l` or `lshosts -s`.

LSF displays the operating system types and versions in your cluster, including any that LSF automatically detects as well as those you have defined manually in the `HostType` section of `lsf.shared`.

You can specify `ostype` in your resource requirement strings. For example, when submitting a job you can specify the following resource requirement: **-R "select[ostype=RHEL2.6]"**.

Modify how long Platform LSF waits for new operating system types and versions

You must enable LSF to automatically detect operating system types and versions.

You can configure how long LSF waits for OS type and version detection.

1. In `lsf.conf`, modify the value for `EGO_ESLIM_TIMEOUT`.

The value is time in seconds.

Add a custom host type or model

The `lsf.shared` file contains a list of host type and host model names for most operating systems. You can add to this list or customize the host type and host model names. A host type and host model name can be any alphanumeric string up to 39 characters long.

1. Log on as the LSF administrator on any host in the cluster.
2. Edit `lsf.shared`:
 - a) For a new host type, modify the Host Type section:

```
Begin HostType
TYPENAME                # Keyword
DEFAULT
IBM Aix564
LINUX86
LINUX64
NTX64
NTIA64
SUNSOL
SOL732
SOL64
SGI 658
SOLX86
HPPA11
HPUX IA64
MACOSX
End HostType
```

- b) For a new host model, modify the Host Model section:

Add the new model and its CPU speed factor relative to other models.

```
Begin HostModel
MODELNAME CPUFACTOR ARCHITECTURE # keyword
# x86 (Solaris, Windows, Linux): approximate values, based on SpecBench results
# for Intel processors (Sparc/Win) and Bogomips results (Linux).
PC75      1.5 (i86pc_75 i586_75 x586_30)
PC90      1.7 (i86pc_90 i586_90 x586_34 x586_35 x586_36)
HP9K715   4.2 (HP9000715_100)
SunSparc  12.0 ()
CRAYJ90   18.0 ()
IBM350    18.0 ()
End HostModel
```

3. Save the changes to `lsf.shared`.
4. Run `lsadmin reconfig` to reconfigure LIM.
5. Run `badmadmin reconfig` to reconfigure `mbatchd`.

Register service ports

LSF uses dedicated UDP and TCP ports for communication. All hosts in the cluster must use the same port numbers to communicate with each other.

The service port numbers can be any numbers ranging from 1024 to 65535 that are not already used by other services.

- Make sure that the port numbers you supply are not already used by applications registered in your service database by checking `/etc/services` or using the command `yycat services`

lsf.conf

By default, port numbers for LSF services are defined in the `lsf.conf` file. You can also configure ports by modifying `/etc/services` or the NIS or NIS+ database. If you define port numbers in `lsf.conf`, port numbers defined in the service database are ignored.

1. Log on to any host as root.
2. Edit `lsf.conf` and add the following lines:

```
LSF_RES_PORT=3878
LSB_MBD_PORT=3881
LSB_SBD_PORT=3882
```

3. Add the same entries to `lsf.conf` on every host.
4. Save `lsf.conf`.
5. Run `lsadmin reconfig` to reconfigure LIM.
6. Run `badmin mbdrestart` to restart `mbatchd`.
7. Run `lsfstartup` to restart all daemons in the cluster.

/etc/services

Configure services manually

Tip:

During installation, use the `hostsetup --boot="y"` option to set up the LSF port numbers in the service database.

1. Use the file `LSF_TOP/version/install/instdlib/example.services` file as a guide for adding LSF entries to the services database.

If any other service listed in your services database has the same port number as one of the LSF services, you must change the port number for the LSF service. You must use the same port numbers on every LSF host.

2. Log on to any host as root.
3. Edit the `/etc/services` file by adding the contents of the `LSF_TOP/version/install/instdlib/example.services` file:

```
# /etc/services entries for LSF daemons
#
res      3878/tcp # remote execution server
lim      3879/udp # load information manager
mbatchd 3881/tcp # master lsbatch daemon
```



```

sbatchd 3882/tcp # slave lsbatch daemon
#
# Add this if ident is not already defined
# in your /etc/services file
ident 113/tcp auth tap # identd

```

4. Run `lsadmin reconfig` to reconfigure LIM.
5. Run `badmin reconfig` to reconfigure `mbatchd`.
6. Run `lsfstart` to restart all daemons in the cluster.

NIS or NIS+ database

If you are running NIS, you only need to modify the services database once per NIS master. On some hosts the NIS database and commands are in the `/var/yp` directory; on others, NIS is found in `/etc/yp`.

1. Log on to any host as root.
2. Run `lsfshut` to shut down all the daemons in the cluster
3. To find the name of the NIS master host, use the command:

```
ypwhich -m services
```

4. Log on to the NIS master host as root.
5. Edit the `/var/yp/src/services` or `/etc/yp/src/services` file on the NIS master host adding the contents of the `LSF_TOP/version/install/lib/example.services` file:

```

# /etc/services entries for LSF daemons.
#
res      3878/tcp # remote execution server
lim      3879/udp # load information manager
mbatchd  3881/tcp # master lsbatch daemon
sbatchd  3882/tcp # slave lsbatch daemon
#
# Add this if ident is not already defined
# in your /etc/services file
ident 113/tcp auth tap # identd

```

Make sure that all the lines you add either contain valid service entries or begin with a comment character (`#`). Blank lines are not allowed.

6. Change the directory to `/var/yp` or `/etc/yp`.
7. Use the following command:

```
ypmake services
```

On some hosts the master copy of the services database is stored in a different location.

On systems running NIS+ the procedure is similar. Refer to your system documentation for more information.

8. Run `lsadmin reconfig` to reconfigure LIM.
9. Run `badmin reconfig` to reconfigure `mbatchd`.
10. Run `lsfstart` to restart all daemons in the cluster.

Host names

LSF needs to match host names with the corresponding Internet host addresses.

LSF looks up host names and addresses the following ways:

- In the `/etc/hosts` file
- Sun Network Information Service/Yellow Pages (NIS or YP)
- Internet Domain Name Service (DNS).

DNS is also known as the Berkeley Internet Name Domain (BIND) or `named`, which is the name of the BIND daemon.

Each host is configured to use one or more of these mechanisms.

Network addresses

Each host has one or more network addresses; usually one for each network to which the host is directly connected. Each host can also have more than one name.

Official host name	The first name configured for each address is called the official name.
Host name aliases	Other names for the same host are called aliases. LSF uses the configured host naming system on each host to look up the official host name for any alias or host address. This means that you can use aliases as input to LSF, but LSF always displays the official name.

Use host name ranges as aliases

The default host file syntax

```
ip_address official_name [ alias [ alias ... ] ]
```

is powerful and flexible, but it is difficult to configure in systems where a single host name has many aliases, and in multihomed host environments.

In these cases, the `hosts` file can become very large and unmanageable, and configuration is prone to error.

The syntax of the LSF `hosts` file supports host name ranges as aliases for an IP address. This simplifies the host name alias specification.

To use host name ranges as aliases, the host names must consist of a fixed node group name prefix and node indices, specified in a form like:

```
host_name[ index_x- index_y, index_m, index_a- index_b ]
```

For example:

```
atl asD0[ 0- 3, 4, 5- 6, ... ]
```

is equivalent to:

```
atl asD0[ 0- 6, ... ]
```

The node list does not need to be a continuous range (some nodes can be configured out). Node indices can be numbers or letters (both upper case and lower case).

Example Some systems map internal compute nodes to single LSF host names. A host file might contain 64 lines, each specifying an LSF host name and 32 node names that correspond to each LSF host:

```
...
177.16.1.1 atlasD0 atlas0 atlas1 atlas2 atlas3 atlas4 ... atlas31
177.16.1.2 atlasD1 atlas32 atlas33 atlas34 atlas35 atlas36 ... atlas63
...
```

In the new format, you still map the nodes to the LSF hosts, so the number of lines remains the same, but the format is simplified because you only have to specify ranges for the nodes, not each node individually as an alias:

```
...
177.16.1.1 atlasD0 atlas[0-31]
177.16.1.2 atlasD1 atlas[32-63]
...
```

You can use either an IPv4 or an IPv6 format for the IP address (if you define the parameter `LSF_ENABLE_SUPPORT_IPV6` in `lsf.conf`).

Host name services

Solaris

On Solaris systems, the `/etc/nsswitch.conf` file controls the name service.

Other UNIX platforms

On other UNIX platforms, the following rules apply:

- If your host has an `/etc/resolv.conf` file, your host is using DNS for name lookups
- If the command `ypcat hosts` prints out a list of host addresses and names, your system is looking up names in NIS
- Otherwise, host names are looked up in the `/etc/hosts` file

For more information

The man pages for the `gethostbyname` function, the `ypbind` and `named` daemons, the `resolver` functions, and the `hosts.svc.conf`, `nsswitch.conf`, and `resolv.conf` files explain host name lookups in more detail.

Hosts with multiple addresses

Multi-homed hosts

Hosts that have more than one network interface usually have one Internet address for each interface. Such hosts are called *multi-homed hosts*. For example, dual-stack hosts are multi-homed because they have both an IPv4 and an IPv6 network address.

LSF identifies hosts by name, so it needs to match each of these addresses with a single host name. To do this, the host name information must be configured so that all of the Internet addresses for a host resolve to the same name.

There are two ways to do it:

- Modify the system hosts file (`/etc/hosts`) and the changes will affect the whole system
- Create an LSF hosts file (`LSF_CONFDIR/hosts`) and LSF will be the only application that resolves the addresses to the same host

Multiple network interfaces

Some system manufacturers recommend that each network interface, and therefore, each Internet address, be assigned a different host name. Each interface can then be directly accessed by name. This setup is often used to make sure NFS requests go to the nearest network interface on the file server, rather than going through a router to some other interface. Configuring this way can confuse LSF, because there is no way to determine that the two different names (or addresses) mean the same host. LSF provides a workaround for this problem.

All host naming systems can be configured so that host address lookups always return the same name, while still allowing access to network interfaces by different names. Each host has an official name and a number of aliases, which are other names for the same host. By configuring all interfaces with the same official name but different aliases, you can refer to each interface by a different alias name while still providing a single official name for the host.

Configure the Platform LSF hosts file

If your LSF clusters include hosts that have more than one interface and are configured with more than one official host name, you must either modify the host name configuration, or create a private `hosts` file for LSF to use.

The LSF `hosts` file is stored in `LSF_CONFDIR`. The format of `LSF_CONFDIR/hosts` is the same as for `/etc/hosts`.

In the LSF `hosts` file, duplicate the system `hosts` database information, except make all entries for the host use the same official name. Configure all the other names for the host as aliases so that you can still refer to the host by any name.

Example

For example, if your `/etc/hosts` file contains:

```
AA. AA. AA. AA  host-AA host # first interface
BB. BB. BB. BB  host-BB      # second interface
```

then the `LSF_CONFDIR/hosts` file should contain:

```
AA. AA. AA. AA  host host-AA # first interface
BB. BB. BB. BB  host host-BB # second interface
```

Example /etc/hosts entries

No unique official name

The following example is for a host with two interfaces, where the host does not have a unique official name.

# Address	Official name	Aliases
# Interface on network A		
AA. AA. AA. AA	host-AA. domain	host. domain host-AA host
# Interface on network B		
BB. BB. BB. BB	host-BB. domain	host-BB host

Looking up the address AA. AA. AA. AA finds the official name host-AA. domain. Looking up address BB. BB. BB. BB finds the name host-BB. domain. No information connects the two names, so there is no way for LSF to determine that both names, and both addresses, refer to the same host.

To resolve this case, you must configure these addresses using a unique host name. If you cannot make this change to the system file, you must create an LSF hosts file and configure these addresses using a unique host name in that file.

Both addresses have the same official name

Here is the same example, with both addresses configured for the same official name.

# Address	Official name	Aliases
# Interface on network A		
AA. AA. AA. AA	host. domain	host-AA. domain host-AA host
# Interface on network B		
BB. BB. BB. BB	host. domain	host-BB. domain host-BB host

With this configuration, looking up either address returns host. domain as the official name for the host. LSF (and all other applications) can determine that all the addresses and host names refer to the same host. Individual interfaces can still be specified by using the host-AA and host-BB aliases.

Example for a dual-stack host

Dual-stack hosts have more than one IP address. You must associate the host name with both addresses, as shown in the following example:

# Address	Official name	Aliases
# Interface IPv4		
AA. AA. AA. AA	host. domain	host-AA. domain
# Interface IPv6		
BBBB: BBBB: BBBB: BBBB: : BBBB	host. domain	host-BB. domain

With this configuration, looking up either address returns host. domain as the official name for the host. LSF (and all other applications) can determine that all the addresses and host names refer to the same host. Individual interfaces can still be specified by using the host-AA and host-BB aliases.

Sun Solaris example

For example, Sun NIS uses the /etc/hosts file on the NIS master host as input, so the format for NIS entries is the same as for the /etc/hosts file. Since LSF can resolve this case, you do not need to create an LSF hosts file.

DNS configuration

The configuration format is different for DNS. The same result can be produced by configuring two address (A) records for each Internet address. Following the previous example:

# name	class	type	address
--------	-------	------	---------

host. domain	IN	A	AA. AA. AA. AA
host. domain	IN	A	BB. BB. BB. BB
host-AA. domain	IN	A	AA. AA. AA. AA
host-BB. domain	IN	A	BB. BB. BB. BB

Looking up the official host name can return either address. Looking up the interface-specific names returns the correct address for each interface.

For a dual-stack host:

#	name	class	type	address
	host. domain	IN	A	AA. AA. AA. AA
	host. domain	IN	A	BBBB: BBBB: BBBB: BBBB: BBBB: BBBB: BBBB: BBBB
	host-AA. domain	IN	A	AA. AA. AA. AA
	host-BB. domain	IN	A	BBBB: BBBB: BBBB: BBBB: BBBB: BBBB: BBBB: BBBB

PTR records in DNS

Address-to-name lookups in DNS are handled using PTR records. The PTR records for both addresses should be configured to return the official name:

#	address	class	type	name
	AA. AA. AA. AA. i n- addr. arpa	IN	PTR	host. domain
	BB. BB. BB. BB. i n- addr. arpa	IN	PTR	host. domain

For a dual-stack host:

#	address	class	type	name
	AA. AA. AA. AA. i n- addr. arpa	IN	PTR	host. domain
	BBBB: BBBB: BBBB: BBBB: BBBB: BBBB: BBBB: BBBB: BBBB: BBBB: i n- addr. arpa	IN	PTR	host. domain

If it is not possible to change the system host name database, create the `host s` file local to the LSF system, and configure entries for the multi-homed hosts only. Host names and addresses not found in the `host s` file are looked up in the standard name system on your host.

Use IPv6 addresses

IP addresses can have either a dotted quad notation (IPv4) or IP Next Generation (IPv6) format. You can use IPv6 addresses if you define the parameter `LSF_ENABLE_SUPPORT_IPV6` in `lsf.conf`; you do not have to map IPv4 addresses to an IPv6 format.

LSF supports IPv6 addresses for the following platforms:

- Linux 2.4
- Linux 2.6
- Solaris 10
- Windows
 - XP
 - 2003
 - 2000 with Service Pack 1 or higher
- AIX 5
- HP-UX
 - 11i
 - 11iv1
 - 11iv2
 - 11.11
- SGI Altix ProPack 3, 4, and 5
- IRIX 6.5.19 and higher, Trusted IRIX 6.5.19 and higher
- Mac OS 10.2 and higher
- Cray XT3
- IBM Power 5 Series

Enable both IPv4 and IPv6 support

1. Configure the parameter `LSF_ENABLE_SUPPORT_IPV6=Y` in `lsf.conf`.

Configure hosts for IPv6

Follow the steps in this procedure if you do not have an IPv6-enabled DNS server or an IPv6-enabled router. IPv6 is supported on some linux2.4 kernels and on all linux2.6 kernels.

1. Configure the kernel.
 - a) Check that the entry `/proc/net/if_inet6` exists.
 - b) If it does not exist, as root run: **`modprobe ipv6`**
 - c) To check that the module loaded correctly, execute the command **`lsmod | grep -w 'ipv6'`**
2. Add an IPv6 address to the host by executing the following command as root: **`/sbin/ifconfig eth0 inet6 add 3ffe:ffff:0:f101::2/64`**
3. Display the IPv6 address using `ifconfig`.
4. Repeat all steps for other hosts in the cluster.
5. Add the addresses for all IPv6 hosts to `/etc/hosts` on each host.

Note:

For IPv6 networking, hosts must be on the same subnet.

6. Test IPv6 communication between hosts using the command `ping6`.

Specify host names with condensed notation

A number of commands often require you to specify host names. You can now specify host name ranges instead. You can use condensed notation with the following commands:

- `bacct`
- `bhist`
- `bjobs`
- `bming`
- `bmod`
- `bpeek`
- `brestart`
- `brsvadd`
- `brsvmod`
- `brsvs`
- `brun`
- `bsub`
- `bswitch`

You must specify a valid range of hosts, where the start number is smaller than the end number.

- Run the command you want and specify the host names as a range.

For example:

`bsub -m "host[1-100].corp.com"`

The job is submitted to host 1. corp. com, host 2. corp. com, host 3. corp. com, all the way to host 100. corp. com.

- Run the command you want and specify host names as a combination of ranges and individuals.

For example:

`bsub -m "host[1-10,12,20-25].corp.com"`

The job is submitted to host 1. corp. com, host 2. corp. com, host 3. corp. com, up to and including host 10. corp. com. It is also submitted to host 12. corp. com and the hosts between and including host 20. corp. com and host 25. corp. com.

Host groups

You can define a host group within LSF or use an external executable to retrieve host group members.

Use `bhosts` to view a list of existing hosts. Use `bmgroup` to view host group membership.

Where to use host groups

LSF host groups can be used in defining the following parameters in LSF configuration files:

- `HOSTS` in `lsb.queues` for authorized hosts for the queue
- `HOSTS` in `lsb.hosts` in the `HostPartiti on` section to list host groups that are members of the host partition

Configure host groups

1. Log in as the LSF administrator to any host in the cluster.
2. Open `lsb.hosts`.
3. Add the `HostGroup` section if it does not exist.

```
Begin HostGroup
GROUP_NAME      GROUP_MEMBER
groupA           (all)
groupB           (groupA ~hostA ~hostB)
groupC           (hostX hostY hostZ)
groupD           (groupC ~hostX)
groupE           (all ~groupC ~hostB)
groupF           (hostF groupC hostK)
desk_tops        (hostD hostE hostF hostG)
Big_servers      (!)
End HostGroup
```

4. Enter a group name under the `GROUP_NAME` column.
External host groups must be defined in the `egroup` executable.
5. Specify hosts in the `GROUP_MEMBER` column.
(Optional) To tell LSF that the group members should be retrieved using `egroup`, put an exclamation mark (!) in the `GROUP_MEMBER` column.
6. Save your changes.
7. Run `badmi n ckconfi g` to check the group definition. If any errors are reported, fix the problem and check the configuration again.
8. Run `badmi n mbdrestart` to apply the new configuration.

Wildcards and special characters to define host names

You can use special characters when defining host group members under the `GROUP_MEMBER` column to specify hosts. These are useful to define several hosts in a single entry, such as for a range of hosts, or for all host names with a certain text string.

If a host matches more than one host group, that host is a member of all groups. If any host group is a condensed host group, the status and other details of the hosts are counted towards all of the matching host groups.

When defining host group members, you can use string literals and the following special characters:

- Tilde (~) excludes specified hosts or host groups from the list. The tilde can be used in conjunction with the other special characters listed below. The following example matches all hosts in the cluster except for hostA, hostB, and all members of the groupA host group:

```
... (all ~hostA ~hostB ~groupA)
```

- Asterisk (*) represent any number of characters. The following example matches all hosts beginning with the text string “hostC” (such as hostCa, hostC1, or hostCZ1):

```
... (hostC*)
```

- Square brackets with a hyphen ([*integer1* - *integer2*]) define a range of non-negative integers at the end of a host name. The first integer must be less than the second integer. The following example matches all hosts from hostD51 to hostD100:

```
... (hostD[ 51- 100])
```

- Square brackets with commas ([*integer1*, *integer2* ...]) define individual non-negative integers at the end of a host name. The following example matches hostD101, hostD123, and hostD321:

```
... (hostD[ 101, 123, 321])
```

- Square brackets with commas and hyphens (such as [*integer1* - *integer2*, *integer3*, *integer4* - *integer5*]) define different ranges of non-negative integers at the end of a host name. The following example matches all hosts from hostD1 to hostD100, hostD102, all hosts from hostD201 to hostD300, and hostD320):

```
... (hostD[ 1- 100, 102, 201- 300, 320])
```

Restrictions

You cannot use more than one set of square brackets in a single host group definition.

The following example is *not* correct:

```
... (hostA[ 1- 10]B[ 1- 20] hostC[ 101- 120])
```

The following example is correct:

```
... (hostA[ 1- 20] hostC[ 101- 120])
```

You cannot define subgroups that contain wildcards and special characters. The following definition for groupB is not correct because groupA defines hosts with a wildcard:

```
Begin HostGroup
GROUP_NAME    GROUP_MEMBER
groupA        (hostA*)
groupB        (groupA)
End HostGroup
```

Define condensed host groups

You can define condensed host groups to display information for its hosts as a summary for the entire group. This is useful because it allows you to see the total statistics of the host group as a whole instead of having to add up the data yourself. This allows you to better plan the distribution of jobs submitted to the hosts and host groups in your cluster.

To define condensed host groups, add a CONDENSE column to the Host Group section. Under this column, enter Y to define a condensed host group or N to define an uncondensed host group, as shown in the following:

```
Begin HostGroup
GROUP_NAME    CONDENSE    GROUP_MEMBER
groupA        Y            (hostA hostB hostD)
groupB        N            (hostC hostE)
End HostGroup
```

The following commands display condensed host group information:

- `bhosts`
- `bhosts -w`
- `bj obs`
- `bj obs -w`

Use `bmgroup -l` to see whether host groups are condensed or not.

Hosts belonging to multiple condensed host groups

If you configure a host to belong to more than one condensed host group using wildcards, `bj obs` can display any of the host groups as execution host name.

For example, host groups `hg1` and `hg2` include the same hosts:

```
Begin HostGroup
GROUP_NAME      CONDENSE  GROUP_MEMBER      # Key words
hg1              Y          (host*)
hg2              Y          (hos*)
End HostGroup
```

Submit jobs using `bsub -m`:

```
bsub -m "hg2" sleep 1001
```

`bj obs` displays `hg1` as the execution host instead of `hg2`:

bjobs							
JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
520	user1	RUN	normal	host5	hg1	sleep 1001	Apr 15 13:50
521	user1	RUN	normal	host5	hg1	sleep 1001	Apr 15 13:50
522	user1	PEND	normal	host5		sleep 1001	Apr 15 13:51

Import external host groups (egroup)

When the membership of a host group changes frequently, or when the group contains a large number of members, you can use an external executable called `egroup` to retrieve a list of members rather than having to configure the group membership manually. You can write a site-specific `egroup` executable that retrieves host group names and the hosts that belong to each group. For information about how to use the external host and user groups feature, see [External Host and User Groups](#) on page 179.

Compute units

Compute units are similar to host groups, with the added feature of granularity allowing the construction of clusterwide structures that mimic network architecture. Job scheduling using compute unit resource requirements optimizes job placement based on the underlying system architecture, minimizing communications bottlenecks. Compute units are especially useful when running communication-intensive parallel jobs spanning several hosts.

Resource requirement strings can specify compute units requirements such as running a job exclusively (`excl`), spreading a job evenly over multiple compute units (`balance`), or choosing compute units based on other criteria.

Compute unit configuration

To enforce consistency, compute unit configuration has the following requirements:

- Hosts and host groups appear in the finest granularity compute unit type, and nowhere else.
- Hosts appear in the membership list of at most one compute unit of the finest granularity.
- All compute units of the same type have the same type of compute units (or hosts) as members.

Tip:

Configure each individual host as a compute unit to use the compute unit features for host level job allocation.

Where to use compute units

LSF compute units can be used in defining the following parameters in LSF configuration files:

- `EXCLUSIVE` in `lsb. queues` for the compute unit type allowed for the queue.
- `HOSTS` in `lsb. queues` for the hosts on which jobs from this queue can be run.
- `RES_REQ` in `lsb. queues` for queue compute unit resource requirements.
- `RES_REQ` in `lsb. applications` for application profile compute unit resource requirements.

Configure compute units

1. Log in as the LSF administrator to any host in the cluster.
2. Open `lsb.params`.
3. Add the `COMPUTE_UNIT_TYPES` parameter if it does not already exist and list your compute unit types in order of granularity (finest first).

```
COMPUTE_UNIT_TYPES=enclosure rack cabinet
```

4. Save your changes.
5. Open `lsb.hosts`.
6. Add the `ComputeUnit` section if it does not exist.

```
Begin ComputeUnit
NAME      MEMBER      TYPE
encl 1    (hostA hg1)         enclosure
encl 2    (hostC hostD) enclosure
encl 3    (hostE hostF) enclosure
encl 4    (hostG hg2)  enclosure
rack1     (encl 1 encl 2) rack
rack2     (encl 3 encl 4) rack
cabin     (rack1 rack2) cabinet
End ComputeUnit
```

7. Enter a compute unit name under the NAME column.

External compute units must be defined in the `egroup` executable.

8. Specify hosts or host groups in the MEMBER column of the finest granularity compute unit type. Specify compute units in the MEMBER column of coarser compute unit types.

(Optional) To tell LSF that the compute unit members of a finest granularity compute unit should be retrieved using `egroup`, put an exclamation mark (!) in the MEMBER column.

9. Specify the type of compute unit in the TYPE column.

10. Save your changes.

11. Run `badmi n ckconfi g` to check the compute unit definition. If any errors are reported, fix the problem and check the configuration again.

12. Run `badmi n mbdrestart` to apply the new configuration.

To view configured compute units, run `bmgroup - cu`.

Use wildcards and special characters to define names in compute units

You can use special characters when defining compute unit members under the MEMBER column to specify hosts, host groups, and compute units. These are useful to define several names in a single entry such as a range of hosts, or for all names with a certain text string.

When defining host, host group, and compute unit members of compute units, you can use string literals and the following special characters:

- Use a tilde (~) to exclude specified hosts, host groups, or compute units from the list. The tilde can be used in conjunction with the other special characters listed below. The following example matches all hosts in `group12` except for `hostA`, and `hostB`:

```
... (group12 ~hostA ~hostB)
```
- Use an asterisk (*) as a wildcard character to represent any number of characters. The following example matches all hosts beginning with the text string "hostC" (such as `hostCa`, `hostC1`, or `hostCZ1`):

```
... (hostC*)
```
- Use square brackets with a hyphen ([*integer1* - *integer2*]) to define a range of non-negative integers at the end of a name. The first integer must be less than the second integer. The following example matches all hosts from `hostD51` to `hostD100`:

```
... (hostD[ 51- 100])
```
- Use square brackets with commas ([*integer1*, *integer2* ...]) to define individual non-negative integers at the end of a name. The following example matches `hostD101`, `hostD123`, and `hostD321`:

```
... (hostD[ 101, 123, 321])
```
- Use square brackets with commas and hyphens (such as [*integer1* - *integer2*, *integer3*, *integer4* - *integer5*]) to define different ranges of non-negative integers at the end of a name. The following example matches all hosts from `hostD1` to `hostD100`, `hostD102`, all hosts from `hostD201` to `hostD300`, and `hostD320`:

```
... (hostD[ 1- 100, 102, 201- 300, 320])
```

Restrictions

You cannot use more than one set of square brackets in a single compute unit definition.

The following example is *not* correct:

```
... (hostA[ 1- 10]B[ 1- 20] hostC[ 101- 120])
```

The following example is correct:

```
... (hostA[ 1- 20] hostC[ 101- 120])
```

The keywords `all`, `all remote`, `all cluster`, `other` and `default` cannot be used when defining compute units.

Define condensed compute units

You can define condensed compute units to display information for its hosts as a summary for the entire group, including the slot usage for each compute unit. This is useful because it allows you to see statistics of the compute unit as a whole instead of having to add up the data yourself. This allows you to better plan the distribution of jobs submitted to the hosts and compute units in your cluster.

To define condensed compute units, add a `CONDENSE` column to the `ComputeUnit` section. Under this column, enter `Y` to define a condensed host group or `N` to define an uncondensed host group, as shown in the following:

```
Begin ComputeUnit
NAME      CONDENSE  MEMBER                TYPE
enclA     Y          (hostA hostB hostD)  enclosure
enclB     N          (hostC hostE)        enclosure
End HostGroup
```

The following commands display condensed host information:

- `bhosts`
- `bhosts -w`
- `bjobs`
- `bjobs -w`

Use `bmgroup -l` to see whether host groups are condensed or not.

Import external host groups (egroup)

When the membership of a compute unit changes frequently, or when the compute unit contains a large number of members, you can use an external executable called `egroup` to retrieve a list of members rather than having to configure the membership manually. You can write a site-specific `egroup` executable that retrieves compute unit names and the hosts that belong to each group, and compute units of the finest granularity can contain `egroups` as members. For information about how to use the external host and user groups feature, see [External Host and User Groups](#) on page 179.

Use compute units with advance reservation

When running exclusive compute unit jobs (with the resource requirement `cu[excl]`), the advance reservation can affect hosts outside the advance reservation but in the same compute unit as follows:

- An exclusive compute unit job dispatched to a host inside the advance reservation will lock the entire compute unit, including any hosts outside the advance reservation.
- An exclusive compute unit job dispatched to a host outside the advance reservation will lock the entire compute unit, including any hosts inside the advance reservation.

Ideally all hosts belonging to a compute unit should be inside or outside of an advance reservation.

Tune CPU factors

CPU factors are used to differentiate the relative speed of different machines. LSF runs jobs on the best possible machines so that response time is minimized.

To achieve this, it is important that you define correct CPU factors for each machine model in your cluster.

How CPU factors affect performance

Incorrect CPU factors can reduce performance the following ways.

- If the CPU factor for a host is too low, that host may not be selected for job placement when a slower host is available. This means that jobs would not always run on the fastest available host.
- If the CPU factor is too high, jobs are run on the fast host even when they would finish sooner on a slower but lightly loaded host. This causes the faster host to be overused while the slower hosts are underused.

Both of these conditions are somewhat self-correcting. If the CPU factor for a host is too high, jobs are sent to that host until the CPU load threshold is reached. LSF then marks that host as busy, and no further jobs will be sent there. If the CPU factor is too low, jobs may be sent to slower hosts. This increases the load on the slower hosts, making LSF more likely to schedule future jobs on the faster host.

Guidelines for setting CPU factors

CPU factors should be set based on a benchmark that reflects your workload. If there is no such benchmark, CPU factors can be set based on raw CPU power.

The CPU factor of the slowest hosts should be set to 1, and faster hosts should be proportional to the slowest.

Example

Consider a cluster with two hosts: host A and host B. In this cluster, host A takes 30 seconds to run a benchmark and host B takes 15 seconds to run the same test. The CPU factor for host A should be 1, and the CPU factor of host B should be 2 because it is twice as fast as host A.

View normalized ratings

1. Run `lsl load -N` to display normalized ratings.

LSF uses a normalized CPU performance rating to decide which host has the most available CPU power. Hosts in your cluster are displayed in order from best to worst. Normalized CPU run queue length values are based on an estimate of the time it would take each host to run one additional unit of work, given that an unloaded host with CPU factor 1 runs one unit of work in one unit of time.

Tune CPU factors

1. Log in as the LSF administrator on any host in the cluster.
2. Edit `lsf.shared`, and change the `HostModel` section:

```
Begin HostModel
MODELNAME  CPUFACTOR  ARCHITECTURE # keyword
#HPUX (HPPA)
HP9K712S   2.5  (HP9000712_60)
HP9K712M   2.5  (HP9000712_80)
HP9K712F   4.0  (HP9000712_100)
```


See the *LSF Configuration Reference* for information about the `lsf.shared` file.

3. Save the changes to `lsf.shared`.
4. Run `lsadmin reconfig` to reconfigure LIM.
5. Run `badmi n reconfi g` to reconfigure `mbatchd`.

Handle host-level job exceptions

You can configure hosts so that LSF detects exceptional conditions while jobs are running, and take appropriate action automatically. You can customize what exceptions are detected, and the corresponding actions. By default, LSF does not detect any exceptions.

Host exceptions LSF can detect

If you configure host exception handling, LSF can detect jobs that exit repeatedly on a host. The host can still be available to accept jobs, but some other problem prevents the jobs from running. Typically jobs dispatched to such “black hole”, or “job-eating” hosts exit abnormally. LSF monitors the job exit rate for hosts, and closes the host if the rate exceeds a threshold you configure (EXIT_RATE in `lsb.hosts`).

If EXIT_RATE is specified for the host, LSF invokes `eadmin` if the job exit rate for a host remains above the configured threshold for longer than 5 minutes. Use JOB_EXIT_RATE_DURATION in `lsb.params` to change how frequently LSF checks the job exit rate.

Use GLOBAL_EXIT_RATE in `lsb.params` to set a cluster-wide threshold in minutes for exited jobs. If EXIT_RATE is not specified for the host in `lsb.hosts`, GLOBAL_EXIT_RATE defines a default exit rate for all hosts in the cluster. Host-level EXIT_RATE overrides the GLOBAL_EXIT_RATE value.

Configure host exception handling (`lsb.hosts`)

EXIT_RATE

Specify a threshold for exited jobs. If the job exit rate is exceeded for 5 minutes or the period specified by JOB_EXIT_RATE_DURATION in `lsb.params`, LSF invokes `eadmin` to trigger a host exception.

Example

The following Host section defines a job exit rate of 20 jobs for all hosts, and an exit rate of 10 jobs on hostA.

```
Begin Host
HOST_NAME      MXJ      EXIT_RATE  # Keywords
Default       !         20
hostA         !         10
End Host
```

Configure thresholds for host exception handling

By default, LSF checks the number of exited jobs every 5 minutes. Use JOB_EXIT_RATE_DURATION in `lsb.params` to change this default.

Tuning

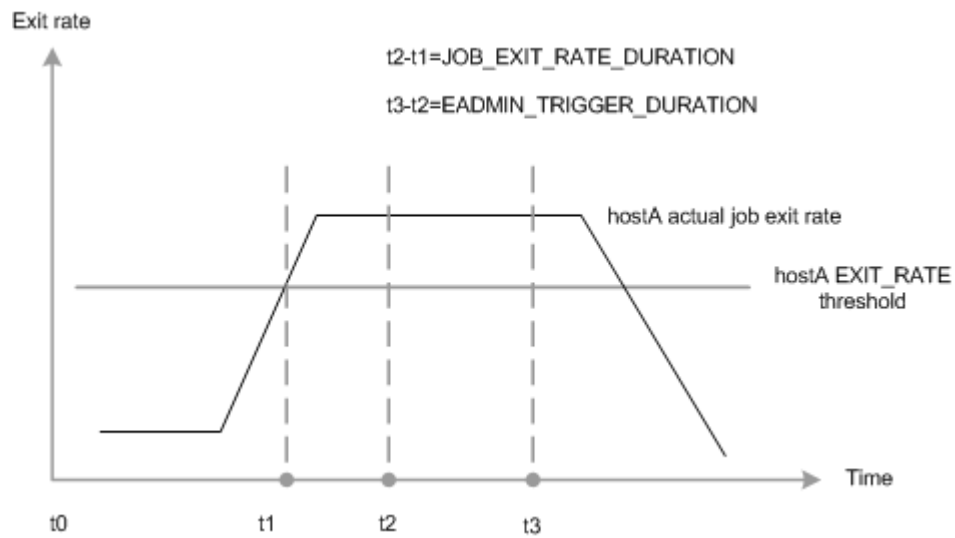
Tip:

Tune JOB_EXIT_RATE_DURATION carefully. Shorter values may raise false alarms, longer values may not trigger exceptions frequently enough.

Example

In the following diagram, the job exit rate of hostA exceeds the configured threshold (EXIT_RATE for hostA in `lsb.hosts`) LSF monitors hostA from time t_1 to time t_2 ($t_2 = t_1 + \text{JOB_EXIT_RATE_DURATION}$ in `lsb.params`). At t_2 , the exit rate is still high, and a host exception is detected. At t_3 (EADMIN_TRIGGER_DURATION in `lsb.params`), LSF invokes `eadmin` and the

host exception is handled. By default, LSF closes host A and sends email to the LSF administrator. Since host A is closed and cannot accept any new jobs, the exit rate drops quickly.



4

Managing Jobs

About job states

The `bj obs` command displays the current state of the job.

Normal job states

Most jobs enter only three states:

Job state	Description
PEND	Waiting in a queue for scheduling and dispatch
RUN	Dispatched to a host and running
DONE	Finished normally with a zero exit value

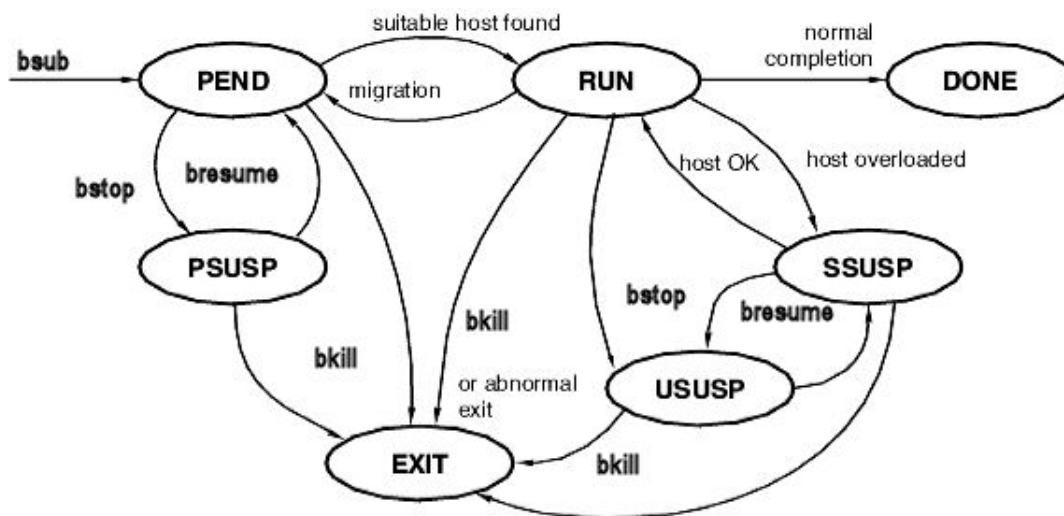
Suspended job states

If a job is suspended, it has three states:

Job state	Description
PSUSP	Suspended by its owner or the LSF administrator while in PEND state
USUSP	Suspended by its owner or the LSF administrator after being dispatched
SSUSP	Suspended by the LSF system after being dispatched

State transitions

A job goes through a series of state transitions until it eventually completes its task, fails, or is terminated. The possible states of a job during its life cycle are shown in the diagram.



Pending jobs

A job remains pending until all conditions for its execution are met. Some of the conditions are:

- Start time specified by the user when the job is submitted

- Load conditions on qualified hosts
- Dispatch windows during which the queue can dispatch and qualified hosts can accept jobs
- Run windows during which jobs from the queue can run
- Limits on the number of job slots configured for a queue, a host, or a user
- Relative priority to other users and jobs
- Availability of the specified resources
- Job dependency and pre-execution conditions

Maximum pending job threshold

If the user or user group submitting the job has reached the pending job threshold as specified by `MAX_PEND_JOBS` (either in the User section of `lsb.users`, or cluster-wide in `lsb.params`), LSF will reject any further job submission requests sent by that user or user group. The system will continue to send the job submission requests with the interval specified by `SUB_TRY_INTERVAL` in `lsb.params` until it has made a number of attempts equal to the `LSB_NTRIES` environment variable. If `LSB_NTRIES` is undefined and LSF rejects the job submission request, the system will continue to send the job submission requests indefinitely as the default behavior.

Suspended jobs

A job can be suspended at any time. A job can be suspended by its owner, by the LSF administrator, by the root user (superuser), or by LSF.

After a job has been dispatched and started on a host, it can be suspended by LSF. When a job is running, LSF periodically checks the load level on the execution host. If any load index is beyond either its per-host or its per-queue suspending conditions, the lowest priority batch job on that host is suspended.

If the load on the execution host or hosts becomes too high, batch jobs could be interfering among themselves or could be interfering with interactive jobs. In either case, some jobs should be suspended to maximize host performance or to guarantee interactive response time.

LSF suspends jobs according to the priority of the job's queue. When a host is busy, LSF suspends lower priority jobs first unless the scheduling policy associated with the job dictates otherwise.

Jobs are also suspended by the system if the job queue has a run window and the current time goes outside the run window.

A system-suspended job can later be resumed by LSF if the load condition on the execution hosts falls low enough or when the closed run window of the queue opens again.

WAIT state (chunk jobs)

If you have configured chunk job queues, members of a chunk job that are waiting to run are displayed as `WAIT` by `bjobs`. Any jobs in `WAIT` status are included in the count of pending jobs by `bqueues` and `busers`, even though the entire chunk job has been dispatched and occupies a job slot. The `bhosts` command shows the single job slot occupied by the entire chunk job in the number of jobs shown in the `NJOBS` column.

You can switch (`bswitch`) or migrate (`bmig`) a chunk job member in `WAIT` state to another queue.

Exited jobs

An exited job ended with a non-zero exit status.

A job might terminate abnormally for various reasons. Job termination can happen from any state. An abnormally terminated job goes into `EXIT` state. The situations where a job terminates abnormally include:

- The job is cancelled by its owner or the LSF administrator while pending, or after being dispatched to a host.
- The job is not able to be dispatched before it reaches its termination deadline set by `bsub -t`, and thus is terminated by LSF.
- The job fails to start successfully. For example, the wrong executable is specified by the user when the job is submitted.
- The application exits with a non-zero exit code.

You can configure hosts so that LSF detects an abnormally high rate of job exit from a host.

Post-execution states

Some jobs may not be considered complete until some post-job processing is performed. For example, a job may need to exit from a post-execution job script, clean up job files, or transfer job output after the job completes.

The DONE or EXIT job states do not indicate whether post-processing is complete, so jobs that depend on processing may start prematurely. Use the `post_done` and `post_err` keywords on the `bsub -w` command to specify job dependency conditions for job post-processing. The corresponding job states POST_DONE and POST_ERR indicate the state of the post-processing.

After the job completes, you cannot perform any job control on the post-processing. Post-processing exit codes are not reported to LSF.

The post-processing of a repetitive job cannot be longer than the repetition period.

View job information

The `bj obs` command is used to display job information. By default, `bj obs` displays information for the user who invoked the command. For more information about `bj obs`, see the *LSF Reference* and the `bj obs (1) man` page.

View all jobs for all users

1. Run `bj obs -u all` to display all jobs for all users.

Job information is displayed in the following order:

- Running jobs
- Pending jobs in the order in which they are scheduled
- Jobs in high-priority queues are listed before those in lower-priority queues

For example:

```
bjobs -u all
JOBID  USER   STAT   QUEUE   FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
1004   user1   RUN    short   hostA       hostA       job0       Dec 16 09:23
1235   user3   PEND   priority hostM                job1       Dec 11 13:55
1234   user2   SSUSP  normal  hostD       hostM       job3       Dec 11 10:09
1250   user1   PEND   short   hostA                job4       Dec 11 13:59
```

View jobs for specific users

1. Run `bj obs -u user_name` to display jobs for a specific user:

```
bjobs -u user1
JOBID  USER   STAT   QUEUE   FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
2225   user1   USUSP  normal  hostA                job1       Nov 16 11:55
2226   user1   PSUSP  normal  hostA                job2       Nov 16 12:30
2227   user1   PSUSP  normal  hostA                job3       Nov 16 12:31
```

View running jobs

1. Run `bj obs -r` to display running jobs.

View done jobs

1. Run `bj obs -d` to display recently completed jobs.

View pending job information

1. Run `bj obs -p` to display the reason why a job is pending.
2. Run `busers -w all` to see the maximum pending job threshold for all users.

View suspension reasons

1. Run `bj obs -s` to display the reason why a job was suspended.

View chunk job wait status and wait reason

1. Run `bhist -l` to display jobs in WAIT status. Jobs are shown as Waiting . . .

The `bjobs -l` command does not display a WAIT reason in the list of pending jobs.

View post-execution states

1. Run `bhist` to display the `POST_DONE` and `POST_ERR` states.

The resource usage of post-processing is not included in the job resource usage.

View exception status for jobs (bjobs)

1. Run `bjobs` to display job exceptions. `bjobs -l` shows exception information for unfinished jobs, and `bjobs -x -l` shows finished as well as unfinished jobs.

For example, the following `bjobs` command shows that job 2 is running longer than the configured `JOB_OVERRUN` threshold, and is consuming no CPU time. `bjobs` displays the job idle factor, and both job overrun and job idle exceptions. Job 1 finished before the configured `JOB_UNDERRUN` threshold, so `bjobs` shows exception status of underrun:

bjobs -x -l -a

```
Job <2>, User <user1>, Project <default>, Status <RUN>, Queue <normal>, Command
<sleep 600>
Wed Aug 13 14:23:35 2009: Submitted from host <hostA>, CWD <$HOME>, Output File
</dev/null>, Specified Hosts <hostB>;
Wed Aug 13 14:23:43 2009: Started on <hostB>, Execution Home </home/user1>, Execution
CWD </home/user1>;
Resource usage collected.
IDLE_FACTOR(cputime/runtime): 0.00
MEM: 3 Mbytes; SWAP: 4 Mbytes; NTHREAD: 3
PGID: 5027; PIDs: 5027 5028 5029
```

SCHEDULING PARAMETERS:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

	cpuspeed	bandwidth
loadSched	-	-
loadStop	-	-

EXCEPTION STATUS: overrun idle

```
Job <1>, User <user1>, Project <default>, Status <DONE>, Queue <normal>, Command
<sleep 20>
Wed Aug 13 14:18:00 2009: Submitted from host <hostA>, CWD <$HOME>,
Output File </dev/null>, Specified Hosts <
hostB>;
Wed Aug 13 14:18:10 2009: Started on <hostB>, Execution Home </home/user1>, Execution
CWD </home/user1>;
Wed Aug 13 14:18:50 2009: Done successfully. The CPU time used is 0.2 seconds.
```

SCHEDULING PARAMETERS:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

	cpuspeed	bandwidth
loadSched	-	-
loadStop	-	-

EXCEPTION STATUS: underrun

Use `bacct -l -x` to trace the history of job exceptions.

Change job order within queues

By default, LSF dispatches jobs in a queue in the order of arrival (that is, first-come, first-served), subject to availability of suitable server hosts.

Use the `bt op` and `bbot` commands to change the position of pending jobs, or of pending job array elements, to affect the order in which jobs are considered for dispatch. Users can only change the relative position of their own jobs, and LSF administrators can change the position of any users' jobs.

bbot

Moves jobs relative to your last job in the queue.

If invoked by a regular user, `bbot` moves the selected job after the last job with the same priority submitted by the user to the queue.

If invoked by the LSF administrator, `bbot` moves the selected job after the last job with the same priority submitted to the queue.

btop

Moves jobs relative to your first job in the queue.

If invoked by a regular user, `btop` moves the selected job before the first job with the same priority submitted by the user to the queue.

If invoked by the LSF administrator, `btop` moves the selected job before the first job with the same priority submitted to the queue.

Move a job to the top of the queue

In the following example, job 5311 is moved to the top of the queue. Since job 5308 is already running, job 5311 is placed in the queue after job 5308.

Note that user 1's job is still in the same position on the queue. user2 cannot use `btop` to get extra jobs at the top of the queue; when one of his jobs moves up the queue, the rest of his jobs move down.

bjobs -u all

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
5308	user2	RUN	normal	hostA	hostD	/s500	Oct 23 10: 16
5309	user2	PEND	ni ght	hostA		/s200	Oct 23 11: 04
5310	user1	PEND	ni ght	hostB		/myj ob	Oct 23 13: 45
5311	user2	PEND	ni ght	hostA		/s700	Oct 23 18: 17

btop 5311

Job <5311> has been moved to position 1 from top.

bjobs -u all

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
5308	user2	RUN	normal	hostA	hostD	/s500	Oct 23 10: 16
5311	user2	PEND	ni ght	hostA		/s200	Oct 23 18: 17
5310	user1	PEND	ni ght	hostB		/myj ob	Oct 23 13: 45
5309	user2	PEND	ni ght	hostA		/s700	Oct 23 11: 04

Switch jobs from one queue to another

You can use the commands `bswi tch` and `bmod` to change jobs from one queue to another. This is useful if you submit a job to the wrong queue, or if the job is suspended because of queue thresholds or run windows and you would like to resume the job.

Switch a single job to a different queue

1. Run `bswi tch` or `bmod` to move pending and running jobs from queue to queue. By default, LSF dispatches jobs in a queue in order of arrival, so a pending job goes to the last position of the new queue, no matter what its position was in the original queue.

In the following example, job 5309 is switched to the `priority` queue:

bswitch priority 5309

Job <5309> is switched to queue <priority>

bjobs -u all

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
5308	user2	RUN	normal	hostA	hostD	/j ob500	Oct 23 10: 16
5309	user2	RUN	priority	hostA	hostB	/j ob200	Oct 23 11: 04
5311	user2	PEND	ni ght	hostA		/j ob700	Oct 23 18: 17
5310	user1	PEND	ni ght	hostB		/myj ob	Oct 23 13: 45

Switch all jobs to a different queue

1. Run `bswitch -q from_queue to_queue 0` to switch all the jobs in a queue to another queue.

The `-q` option is used to operate on all jobs in a queue. The job ID number 0 specifies that all jobs from the `night` queue should be switched to the `idle` queue:

The example below selects jobs from the `ni ght` queue and switches them to the `i dl e` queue.

bswitch -q night idle 0

Job <5308> is switched to queue <idle>

Job <5310> is switched to queue <idle>

Force job execution

A pending job can be forced to run with the `brun` command. This operation can only be performed by an LSF administrator.

You can force a job to run on a particular host, to run until completion, and other restrictions. For more information, see the `brun` command.

When a job is forced to run, any other constraints associated with the job such as resource requirements or dependency conditions are ignored.

In this situation you may see some job slot limits, such as the maximum number of jobs that can run on a host, being violated. A job that is forced to run cannot be preempted.

Force a pending job to run

1. Run `brun -m hostname job_ID` to force a pending job to run.

You must specify the host on which the job will run.

For example, the following command will force the sequential job 104 to run on host A:

```
brun -m hostA 104
```

Suspend and resume jobs

A job can be suspended by its owner or the LSF administrator. These jobs are considered user-suspended and are displayed by `bj obs` as `USUSP`.

If a user suspends a high priority job from a non-preemptive queue, the load may become low enough for LSF to start a lower priority job in its place. The load created by the low priority job can prevent the high priority job from resuming. This can be avoided by configuring preemptive queues.

Suspend a job

1. Run `bstop job_ID`.

Your job goes into `USUSP` state if the job is already started, or into `PSUSP` state if it is pending.

```
bstop 3421
Job <3421> is being stopped
```

The above example suspends job 3421.

UNIX

`bstop` sends the following signals to the job:

- `SI GTSTP` for parallel or interactive jobs—`SI GTSTP` is caught by the master process and passed to all the slave processes running on other hosts.
- `SI GSTOP` for sequential jobs—`SI GSTOP` cannot be caught by user programs. The `SI GSTOP` signal can be configured with the `LSB_SIGSTOP` parameter in `lsf.conf`.

Windows

`bstop` causes the job to be suspended.

Resume a job

1. Run `bresume job_ID`.

```
bresume 3421
Job <3421> is being resumed
```

resumes job 3421.

Resuming a user-suspended job does not put your job into `RUN` state immediately. If your job was running before the suspension, `bresume` first puts your job into `SSUSP` state and then waits for `sbatchd` to schedule it according to the load conditions.

Kill jobs

The `bkill` command cancels pending batch jobs and sends signals to running jobs. By default, on UNIX, `bkill` sends the `SIGKILL` signal to running jobs.

Before `SIGKILL` is sent, `SIGINT` and `SIGTERM` are sent to give the job a chance to catch the signals and clean up. The signals are forwarded from `mbatchd` to `sbatchd`. `sbatchd` waits for the job to exit before reporting the status. Because of these delays, for a short period of time after the `bkill` command has been issued, `bjobs` may still report that the job is running.

On Windows, job control messages replace the `SIGINT` and `SIGTERM` signals, and termination is implemented by the `TerminateProcess()` system call.

Kill a job

1. Run `bkill job_ID`. For example, the following command kills job 3421:

```
bkill 3421
Job <3421> is being terminated
```

Kill multiple jobs

1. Run `bkill 0` to kill all pending jobs in the cluster or use `bkill 0` with the `-g`, `-J`, `-m`, `-q`, or `-u` options to kill all jobs that satisfy these options.

The following command kills all jobs dispatched to the `hostA` host:

```
bkill -m hostA 0
Job <267> is being terminated
Job <268> is being terminated
Job <271> is being terminated
```

The following command kills all jobs in the `groupA` job group:

```
bkill -g groupA 0
Job <2083> is being terminated
Job <2085> is being terminated
```

Kill a large number of jobs rapidly

Killing multiple jobs with `bkill 0` and other commands is usually sufficient for moderate numbers of jobs. However, killing a large number of jobs (approximately greater than 1000 jobs) can take a long time to finish.

1. Run `bkill -b` to kill a large number of jobs faster than with normal means. However, jobs killed in this manner are not logged to `lsb. acct`.

Local pending jobs are killed immediately and cleaned up as soon as possible, ignoring the time interval specified by `CLEAN_PERIOD` in `lsb. params`. Other jobs are killed as soon as possible but cleaned up normally (after the `CLEAN_PERIOD` time interval).

If the `-b` option is used with `bkill 0`, it kills all applicable jobs and silently skips the jobs that cannot be killed.

The `-b` option is ignored if used with `-r` or `-s`.

Force removal of a job from Platform LSF

1. Run `bkill -r` to force the removal of the job from LSF. Use this option when a job cannot be killed in the operating system.

The `bkill -r` command removes a job from the LSF system without waiting for the job to terminate in the operating system. This sends the same series of signals as `bkill` without `-r`, except that the job is removed from the system immediately, the job is marked as EXIT, and job resources that LSF monitors are released as soon as LSF receives the first signal.

Send a signal to a job

LSF uses signals to control jobs, to enforce scheduling policies, or in response to user requests. The principal signals LSF uses are `SI GSTOP` to suspend a job, `SI GCONT` to resume a job, and `SI GKI LL` to terminate a job.

Occasionally, you may want to override the default actions. For example, instead of suspending a job, you might want to kill or checkpoint it. You can override the default job control actions by defining the `JOB_CONTROLS` parameter in your queue configuration. Each queue can have its separate job control actions.

You can also send a signal directly to a job. You cannot send arbitrary signals to a pending job; most signals are only valid for running jobs. However, LSF does allow you to kill, suspend and resume pending jobs.

You must be the owner of a job or an LSF administrator to send signals to a job.

You use the `bkill -s` command to send a signal to a job. If you issue `bkill` without the `-s` option, a `SI GKI LL` signal is sent to the specified jobs to kill them. Twenty seconds before `SI GKI LL` is sent, `SI GTERM` and `SI GI NT` are sent to give the job a chance to catch the signals and clean up.

On Windows, job control messages replace the `SI GI NT` and `SI GTERM` signals, but only customized applications are able to process them. Termination is implemented by the `TerminateProcess()` system call.

Signals on different platforms

LSF translates signal numbers across different platforms because different host types may have different signal numbering. The real meaning of a specific signal is interpreted by the machine from which the `bkill` command is issued.

For example, if you send signal 18 from a SunOS 4.x host, it means `SI GTSTP`. If the job is running on HP-UX and `SI GTSTP` is defined as signal number 25, LSF sends signal 25 to the job.

Send a signal to a job

On most versions of UNIX, signal names and numbers are listed in the `kill (1)` or `signal (2)` man pages. On Windows, only customized applications are able to process job control messages specified with the `-s` option.

1. Run `bkill -s signal job_id`, where *signal* is either the signal name or the signal number:

```
bkill -s TSTP 3421
Job <3421> is being signaled
```

The above example sends the TSTP signal to job 3421.

Job groups

A collection of jobs can be organized into job groups for easy management. A job group is a container for jobs in much the same way that a directory in a file system is a container for files. For example, a payroll application may have one group of jobs that calculates weekly payments, another job group for calculating monthly salaries, and a third job group that handles the salaries of part-time or contract employees. Users can submit, view, and control jobs according to their groups rather than looking at individual jobs.

How job groups are created

Job groups can be created *explicitly* or *implicitly*:

- A job group is created *explicitly* with the `bgadd` command.
- A job group is created *implicitly* by the `bsub -g` or `bmod -g` command when the specified group does not exist. Job groups are also created implicitly when a default job group is configured (DEFAULT_JOBGROUP in `lsb.params` or `LSB_DEFAULT_JOBGROUP` environment variable).

Job groups created when jobs are attached to an SLA service class at submission are implicit job groups (`bsub -sl a service_class_name -g job_group_name`). Job groups attached to an SLA service class with `bgadd` are explicit job groups (`bgadd -sl a service_class_name job_group_name`).

The `GRP_ADD` event in `lsb.events` indicates how the job group was created:

- 0x01 - job group was created explicitly
- 0x02 - job group was created implicitly

For example:

```
GRP_ADD "7.02" 1193032735 1285 1193032735 0 "/Z" "" "user1" "" "" 2 0 "" -1 1
```

means job group `/Z` is an explicitly created job group.

Child groups can be created explicitly or implicitly under any job group. Only an implicitly created job group which has no job group limit (`bgadd -L`) and is not attached to any SLA can be automatically deleted once it becomes empty. An empty job group is a job group that has no jobs associated with it (including finished jobs). `NJOBS` displayed by `bj group` is 0.

Job group hierarchy

Jobs in job groups are organized into a hierarchical tree similar to the directory structure of a file system. Like a file system, the tree contains groups (which are like directories) and jobs (which are like files). Each group can contain other groups or individual jobs. Job groups are created independently of jobs, and can have dependency conditions which control when jobs within the group are considered for scheduling.

Job group path

The *job group path* is the name and location of a job group within the job group hierarchy. Multiple levels of job groups can be defined to form a hierarchical tree. A job group can contain jobs and sub-groups.

Root job group

LSF maintains a single tree under which all jobs in the system are organized. The top-most level of the tree is represented by a top-level “root” job group, named `/`. The root group is owned by the primary LSF Administrator and cannot be removed. Users and administrators create new groups under the root group. By default, if you do not specify a job group path name when submitting a job, the job is created under the top-level “root” job group, named `/`.

The root job group is not displayed by job group query commands, and you cannot specify the root job in commands.

Job group owner

Each group is owned by the user who created it. The login name of the user who creates the job group is the job group owner. Users can add job groups into a groups that are owned by other users, and they can submit jobs to groups owned by other users. Child job groups are owned by the creator of the job group and the creators of any parent groups.

Job control under job groups

Job owners can control their own jobs attached to job groups as usual. Job group owners can also control any job under the groups they own and below.

For example:

- Job group /A is created by user 1
- Job group /A/B is created by user 2
- Job group /A/B/C is created by user 3

All users can submit jobs to any job group, and control the jobs they own in all job groups. For jobs submitted by other users:

- user 1 can control jobs submitted by other users in all 3 job groups: /A, /A/B, and /A/B/C
- user 2 can control jobs submitted by other users only in 2 job groups: /A/B and /A/B/C
- user 3 can control jobs submitted by other users only in job group /A/B/C

The LSF administrator can control jobs in any job group.

Default job group

You can specify a default job group for jobs submitted without explicitly specifying a job group. LSF associates the job with the job group specified with `DEFAULT_JOBGROUP` in `lsb.params`. The `LSB_DEFAULT_JOBGROUP` environment variable overrides the setting of `DEFAULT_JOBGROUP`. The `bsub -g job_group_name` option overrides both `LSB_DEFAULT_JOBGROUP` and `DEFAULT_JOBGROUP`.

Default job group specification supports macro substitution for project name (%p) and user name (%u). When you specify `bsub -P project_name`, the value of %p is the specified project name. If you do not specify a project name at job submission, %p is the project name defined by setting the environment variable `LSB_DEFAULTPROJECT`, or the project name specified by `DEFAULT_PROJECT` in `lsb.params`. the default project name is `default`.

For example, a default job group name specified by `DEFAULT_JOBGROUP=/canada/%p/%u` is expanded to the value for the LSF project name and the user name of the job submission user (for example, `/canada/projects/user1`).

Job group names must follow this format:

- Job group names must start with a slash character (/). For example, `DEFAULT_JOBGROUP=/A/B/C` is correct, but `DEFAULT_JOBGROUP=A/B/C` is not correct.
- Job group names cannot end with a slash character (/). For example, `DEFAULT_JOBGROUP=/A/` is not correct.
- Job group names cannot contain more than one slash character (/) in a row. For example, job group names like `DEFAULT_JOBGROUP=/A//B` or `DEFAULT_JOBGROUP=A///B` are not correct.

- Job group names cannot contain spaces. For example, `DEFAULT_JOBGROUP=/A/B C/D` is not correct.
- Project names and user names used for macro substitution with `%p` and `%u` cannot start or end with slash character (`/`).
- Project names and user names used for macro substitution with `%p` and `%u` cannot contain spaces or more than one slash character (`/`) in a row.
- Project names or user names containing slash character (`/`) will create separate job groups. For example, if the project name is `canada/projects`, `DEFAULT_JOBGROUP=%p` results in a job group hierarchy `/canada/projects`.

Job group limits

Job group limits specified with `bgadd -L` apply to the job group hierarchy. The job group limit is a positive number greater than or equal to zero (0), specifying the maximum number of running and suspended jobs under the job group (including child groups). If limit is zero (0), no jobs under the job group can run. By default, a job group has no limit. Limits persist across `mbatchd` restart and reconfiguration.

You cannot specify a limit for the root job group. The root job group has no job limit. Job groups added with no limits specified inherit any limits of existing parent job groups. The `-L` option only limits the lowest level job group created. The maximum number of running and suspended jobs (including `USUSP` and `SSUSP`) in a job group cannot exceed the limit defined on the job group and its parent job group.

The job group limit is based on the number of running and suspended jobs in the job group. If you specify a job group limit as 2, at most 2 jobs can run under the group at any time, regardless of how many jobs or job slots are used. If the currently available job slots is zero (0), even if the job group job limit is not exceeded, LSF cannot dispatch a job to the job group.

If a parallel job requests 2 CPUs (`bsub -n 2`), the job group limit is per job, not per slots used by the job.

A job array may also be under a job group, so job arrays also support job group limits.

Job group limits are not supported at job submission for job groups created automatically with `bsub -g`. Use `bgadd -L` before job submission.

Jobs forwarded to the execution cluster in a MultiCluster environment are not counted towards the job group limit.

Examples

```
bgadd -L 6 /canada/projects/test
```

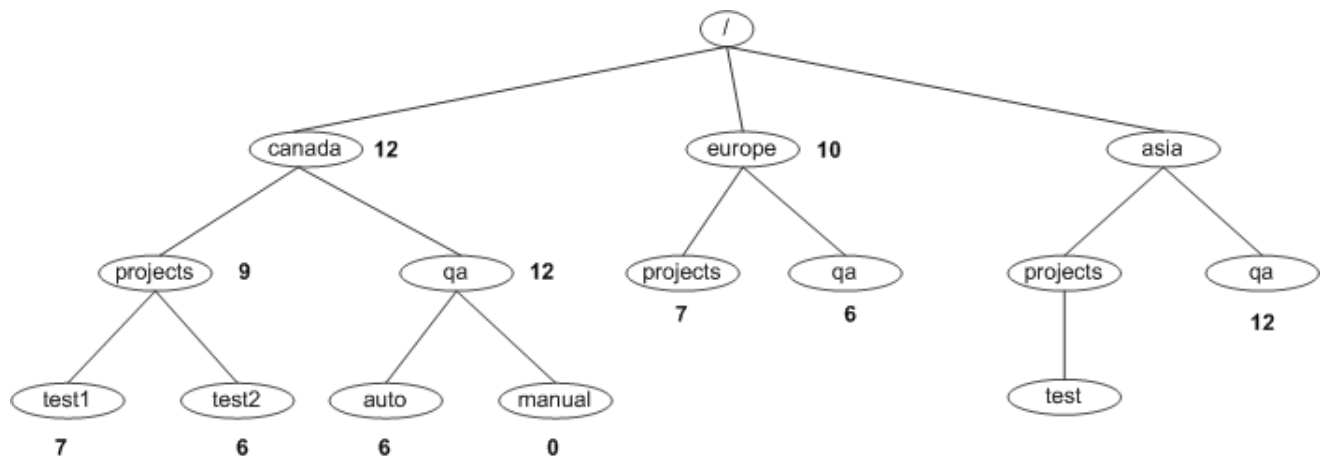
If `/canada` is existing job group, and `/canada/projects` and `/canada/projects/test` are new groups, only the job group `/canada/projects/test` is limited to 6 running and suspended jobs. Job group `/canada/projects` will have whatever limit is specified for its parent job group `/canada`. The limit of `/canada` does not change.

The limits on child job groups cannot exceed the parent job group limit. For example, if `/canada/projects` has a limit of 5:

```
bgadd -L 6 /canada/projects/test
```

is rejected because `/canada/projects/test` attempts to increase the limit of its parent `/canada/projects` from 5 to 6.

Example job group hierarchy with limits



In this configuration:

- Every node is a job group, including the root (/) job group
- The root (/) job group cannot have any limit definition
- By default, child groups have the same limit definition as their direct parent group, so /asia, /asia/projects, and /asia/projects/test all have no limit
- The number of running and suspended jobs in a job group (including all of its child groups) cannot exceed the defined limit
- If there are 7 running or suspended jobs in job group /canada/projects/test1, even though the job limit of group /canada/qa/auto is 6, /canada/qa/auto can only have a maximum of 5 running and suspended (12-7=5)
- When a job is submitted to a job group, LSF checks the limits for the entire job group. For example, for a job is submitted to job group /canada/qa/auto, LSF checks the limits on groups /canada/qa/auto, /canada/qa and /canada. If any one limit in the branch of the hierarchy is exceeded, the job remains pending
- The zero (0) job limit for job group /canada/qa/manual means no job in the job group can enter running status

Create a job group

1. Use the `bgadd` command to create a new job group.

You must provide full group path name for the new job group. The last component of the path is the name of the new group to be created:

`bgadd /risk_group`

The above example creates a job group named `risk_group` under the root group `/`.

`bgadd /risk_group/portfolio1`

The above example creates a job group named `portfolio1` under job group `/risk_group`.

`bgadd /risk_group/portfolio1/current`

The above example creates a job group named `current` under job group `/risk_group/portfolio1`.

If the group hierarchy `/risk_group/portfolio1/current` does not exist, LSF checks its parent recursively, and if no groups in the hierarchy exist, all three job groups are created with the specified hierarchy.

Add a job group limit (bgadd)

1. Run `bgadd -L limit/job_group_name` to specify a job limit for a job group.

Where *limit* is a positive number greater than or equal to zero (0), specifying the maximum the number of running and suspended jobs under the job group (including child groups). If limit is zero (0), no jobs under the job group can run.

For example:

```
bgadd -L 6 /canada/projects/test
```

If `/canada` is existing job group, and `/canada/projects` and `/canada/projects/test` are new groups, only the job group `/canada/projects/test` is limited to 6 running and suspended jobs. Job group `/canada/projects` will have whatever limit is specified for its parent job group `/canada`. The limit of `/canada` does not change.

Submit jobs under a job group

1. Use the `-g` option of `bsub` to submit a job into a job group.

The job group does not have to exist before submitting the job.

```
bsub -g /risk_group/portfolio1/current myjob  
Job <105> is submitted to default queue.
```

Submits my job to the job group `/risk_group/portfolio1/current`.

If group `/risk_group/portfolio1/current` exists, job 105 is attached to the job group.

If group `/risk_group/portfolio1/current` does not exist, LSF checks its parent recursively, and if no groups in the hierarchy exist, all three job groups are created with the specified hierarchy and the job is attached to group.

-g and -sla options

Tip:

Use `-sla` with `-g` to attach all jobs in a job group to a service class and have them scheduled as SLA jobs. Multiple job groups can be created under the same SLA. You can submit additional jobs to the job group without specifying the service class name again.

MultiCluster

In a MultiCluster job forwarding mode, job groups only apply on the submission cluster, not on the execution cluster. LSF treats the execution cluster as execution engine, and only enforces job group policies at the submission cluster.

Jobs forwarded to the execution cluster in a MultiCluster environment are not counted towards job group limits.

View information about job groups (bjgroup)

1. Use the `bj group` command to see information about jobs in job groups.

bjgroup									
GROUP_NAME	NJOBS	PEND	RUN	SSUSP	USUSP	FINISH	SLA	JLIMIT	OWNER
/A	0	0	0	0	0	0	()	0/10	user1
/X	0	0	0	0	0	0	()	0/-	user2
/A/B	0	0	0	0	0	0	()	0/5	user1
/X/Y	0	0	0	0	0	0	()	0/5	user2

2. Use `bj group -s` to sort job groups by group hierarchy.

For example, for job groups named /A, /A/B, /X and /X/Y, `bj group -s` displays:

bjgroup -s									
GROUP_NAME	NJOBS	PEND	RUN	SSUSP	USUSP	FINISH	SLA	JLIMIT	OWNER
/A	0	0	0	0	0	0	()	0/10	user1
/A/B	0	0	0	0	0	0	()	0/5	user1
/X	0	0	0	0	0	0	()	0/-	user2
/X/Y	0	0	0	0	0	0	()	0/5	user2

3. Specify a job group name to show the hierarchy of a single job group:

bjgroup -s /X									
GROUP_NAME	NJOBS	PEND	RUN	SSUSP	USUSP	FINISH	SLA	JLIMIT	OWNER
/X	25	0	25	0	0	0	pucini	25/100	user1
/X/Y	20	0	20	0	0	0	pucini	20/30	user1
/X/Z	5	0	5	0	0	0	pucini	5/10	user2

4. Specify a job group name with a trailing slash character (/) to show only the root job group:

bjgroup -s /X/									
GROUP_NAME	NJOBS	PEND	RUN	SSUSP	USUSP	FINISH	SLA	JLIMIT	OWNER
/X	25	0	25	0	0	0	pucini	25/100	user1

5. Use `bj group -N` to display job group information by job slots instead of number of jobs. NSLOTS, PEND, RUN, SSUSP, USUSP, RSV are all counted in slots rather than number of jobs:

bjgroup -N									
GROUP_NAME	NSLOTS	PEND	RUN	SSUSP	USUSP	RSV	SLA	OWNER	
/X	25	0	25	0	0	0	pucini	user1	
/A/B	20	0	20	0	0	0	wagner	batch	

`-N` by itself shows job slot info for all job groups, and can combine with `-s` to sort the job groups by hierarchy:

bjgroup -N -s									
GROUP_NAME	NSLOTS	PEND	RUN	SSUSP	USUSP	RSV	SLA	OWNER	
/A	0	0	0	0	0	0	wagner	batch	
/A/B	0	0	0	0	0	0	wagner	user1	
/X	25	0	25	0	0	0	pucini	user1	
/X/Y	20	0	20	0	0	0	pucini	batch	
/X/Z	5	0	5	0	0	0	pucini	batch	

View jobs for a specific job group (bjobs)

1. Run `bj obs -g` and specify a job group path to view jobs attached to the specified group.

bjobs -g /risk_group							
JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
113	user1	PEND	normal	hostA		myjob	Jun 17 16:15
111	user2	RUN	normal	hostA	hostA	myjob	Jun 14 15:13
110	user1	RUN	normal	hostB	hostA	myjob	Jun 12 05:03

```
104      user3      RUN      normal      hostA      hostC      myj ob      Jun 11 13:18
```

`bj obs -l` displays the full path to the group to which a job is attached:

```
bjobs -l -g /risk_group
Job <101>, User <user1>, Project <default>, Job Group </risk_group>, Status <RUN>,
Queue <normal>, Command <myjob>
Tue Jun 17 16:21:49 2009: Submitted from host <hostA>, CWD </home/user1>;
Tue Jun 17 16:22:01 2009: Started on <hostA>;
...
```

Control jobs in job groups

Suspend and resume jobs in job groups, move jobs to different job groups, terminate jobs in job groups, and delete job groups.

Suspend jobs (bstop)

1. Use the `-g` option of `bstop` and specify a job group path to suspend jobs in a job group

```
bstop -g /risk_group 106
Job <106> is being stopped
```

2. Use job ID 0 (zero) to suspend all jobs in a job group:

```
bstop -g /risk_group/consolidate 0
Job <107> is being stopped
Job <108> is being stopped
Job <109> is being stopped
```

Resume suspended jobs (bresume)

1. Use the `-g` option of `bresume` and specify a job group path to resume suspended jobs in a job group:

```
bresume -g /risk_group 106
Job <106> is being resumed
```

2. Use job ID 0 (zero) to resume all jobs in a job group:

```
bresume -g /risk_group 0
Job <109> is being resumed
Job <110> is being resumed
Job <112> is being resumed
```

Move jobs to a different job group (bmod)

1. Use the `-g` option of `bmod` and specify a job group path to move a job or a job array from one job group to another.

```
bmod -g /risk_group/portfolio2/monthly 105
```

moves job 105 to job group `/risk_group/portfolio2/monthly`.

Like `bsub -g`, if the job group does not exist, LSF creates it.

`bmod -g` cannot be combined with other `bmod` options. It can only operate on pending jobs. It cannot operate on running or finished jobs.

If you define `LSB_MOD_ALL_JOBS=Y` in `lsf.conf`, `bmod -g` can also operate on running jobs.

You can modify your own job groups and job groups that other users create under your job groups. The LSF administrator can modify job groups of all users.

You cannot move job array elements from one job group to another, only entire job arrays. If any job array elements in a job array are running, you cannot move the job array to another group. A job array can only belong to one job group at a time.

You cannot modify the job group of a job attached to a service class.

`bhist -l` shows job group modification information:

bhist -l 105

```
Job <105>, User <user1>, Project <default>, Job Group </risk_group>, Command <myjob>
Wed May 14 15:24:07 2009: Submitted from host <hostA>, to Queue <normal>, CWD <$HOME/lsf51/5.1/
sparc-sol7-64/bin>;
Wed May 14 15:24:10 2009: Parameters of Job are changed:
                          Job group changes to: /risk_group/portfolio2/monthly;
Wed May 14 15:24:17 2009: Dispatched to <hostA>;
Wed May 14 15:24:17 2009: Starting (Pid 8602);
...
```

Terminate jobs (bkill)

1. Use the `-g` option of `bkill` and specify a job group path to terminate jobs in a job group.

bkill -g /risk_group 106

```
Job <106> is being terminated
```

2. Use job ID 0 (zero) to terminate all jobs in a job group:

bkill -g /risk_group 0

```
Job <1413> is being terminated
Job <1414> is being terminated
Job <1415> is being terminated
Job <1416> is being terminated
```

`bkill` only kills jobs in the job group you specify. It does not kill jobs in lower level job groups in the path. For example, jobs are attached to job groups `/risk_group` and `/risk_group/consolidate`:

bsub -g /risk_group myjob

```
Job <115> is submitted to default queue <normal>.
```

bsub -g /risk_group/consolidate myjob2

```
Job <116> is submitted to default queue <normal>.
```

The following `bkill` command only kills jobs in `/risk_group`, not the subgroup `/risk_group/consolidate`:

bkill -g /risk_group 0

```
Job <115> is being terminated
```

To kill jobs in `/risk_group/consolidate`, specify the path to the `consolidate` job group explicitly:

bkill -g /risk_group/consolidate 0

```
Job <116> is being terminated
```

Delete a job groups manually (bgdel)

1. Use the `bgdel` command to manually remove a job group. The job group cannot contain any jobs.

bgdel /risk_group

```
Job group /risk_group is deleted.
```

deletes the job group `/risk_group` and all its subgroups.

Normal users can only delete the empty groups they own that are specified by the requested *job_group_name*. These groups can be explicit or implicit.

2. Run `bgdel 0` to delete all empty job groups you own. These groups can be explicit or implicit.
3. LSF administrators can use `bgdel -u user_name 0` to delete all empty job groups created by specific users. These groups can be explicit or implicit.

Run `bgdel -u all 0` to delete all the users' empty job groups and their sub groups. LSF administrators can delete empty job groups created by any user. These groups can be explicit or implicit.

4. Run `bgdel -c job_group_name` to delete all empty groups below the requested *job_group_name* including *job_group_name* itself.

Modify a job group limit (bgmod)

1. Run `bgmod` to change a job group limit.

```
bgmod [-L limit | -Ln] /job_group_name
```

`-L limit` changes the limit of *job_group_name* to the specified value. If the job group has parent job groups, the new limit cannot exceed the limits of any higher level job groups. Similarly, if the job group has child job groups, the new value must be greater than any limits on the lower level job groups.

`-Ln` removes the existing job limit for the job group. If the the job group has parent job groups, the job modified group automatically inherits any limits from its direct parent job group.

You must provide full group path name for the modified job group. The last component of the path is the name of the job group to be modified.

Only root, LSF administrators, or the job group creator, or the creator of the parent job groups can use `bgmod` to modify a job group limit.

The following command only modifies the limit of group `/canada/projects/test1`. It does not modify limits of `/canada` or `/canada/projects`.

```
bgmod -L 6 /canada/projects/test1
```

To modify limits of `/canada` or `/canada/projects`, you must specify the exact group name:

```
bgmod -L 6 /canada
```

or

```
bgmod -L 6 /canada/projects
```

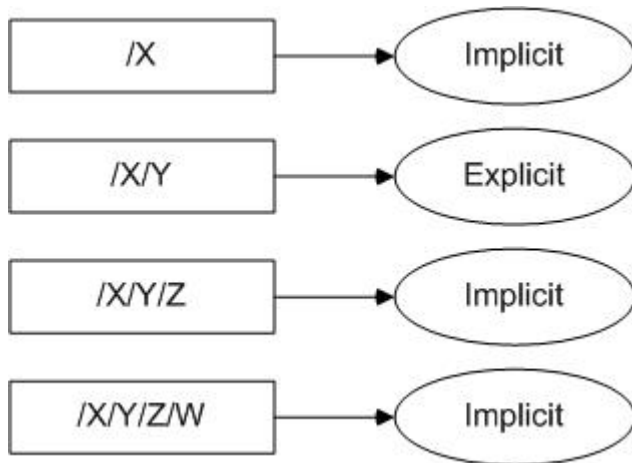
Automatic job group cleanup

When an implicitly created job group becomes empty, it can be automatically deleted by LSF. Job groups that can be automatically deleted cannot:

- Have limits specified including their child groups
- Have explicitly created child job groups
- Be attached to any SLA

Configure `JOB_GROUP_CLEAN=Y` in `lsb.params` to enable automatic job group deletion.

For example, for the following job groups:



When automatic job group deletion is enabled, LSF only deletes job groups `/X/Y/Z/W` and `/X/Y/Z`. Job group `/X/Y` is not deleted because it is an explicitly created job group, Job group `/X` is also not deleted because it has an explicitly created child job group `/X/Y`.

Automatic job group deletion does not delete job groups attached to SLA service classes. Use `bgdel` to manually delete job groups attached to SLAs.

Handle job exceptions

You can configure hosts and queues so that LSF detects exceptional conditions while jobs are running, and take appropriate action automatically. You can customize what exceptions are detected and their corresponding actions. By default, LSF does not detect any exceptions.

Run `bj obs -d -m host_name` to see exited jobs for a particular host.

Job exceptions LSF can detect

If you configure job exception handling in your queues, LSF detects the following job exceptions:

- Job underrun — jobs end too soon (run time is less than expected). Underrun jobs are detected when a job exits abnormally
- Job overrun — job runs too long (run time is longer than expected). By default, LSF checks for overrun jobs every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `l sb. params` to change how frequently LSF checks for job overrun.
- Job estimated run time exceeded — the job's actual run time has exceeded the estimated run time.
- Idle job — running job consumes less CPU time than expected (in terms of CPU time/runtime). By default, LSF checks for idle jobs every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `l sb. params` to change how frequently LSF checks for idle jobs.

Host exceptions LSF can detect

If you configure host exception handling, LSF can detect jobs that exit repeatedly on a host. The host can still be available to accept jobs, but some other problem prevents the jobs from running. Typically jobs dispatched to such “black hole”, or “job-eating” hosts exit abnormally. By default, LSF monitors the job exit rate for hosts, and closes the host if the rate exceeds a threshold you configure (`EXIT_RATE` in `l sb. hosts`).

If `EXIT_RATE` is not specified for the host, LSF invokes `eadmi n` if the job exit rate for a host remains above the configured threshold for longer than 5 minutes. Use `JOB_EXIT_RATE_DURATION` in `l sb. params` to change how frequently LSF checks the job exit rate.

Use `GLOBAL_EXIT_RATE` in `l sb. params` to set a cluster-wide threshold in minutes for exited jobs. If `EXIT_RATE` is not specified for the host in `l sb. hosts`, `GLOBAL_EXIT_RATE` defines a default exit rate for all hosts in the cluster. Host-level `EXIT_RATE` overrides the `GLOBAL_EXIT_RATE` value.

Customize job exception actions with the `eadmi n` script

When an exception is detected, LSF takes appropriate action by running the script `LSF_SERVERDIR/eadmi n` on the master host.

You can customize `eadmi n` to suit the requirements of your site. For example, `eadmi n` could find out the owner of the problem jobs and use `bst op -u` to stop all jobs that belong to the user.

In some environments, a job running 1 hour would be an overrun job, while this may be a normal job in other environments. If your configuration considers jobs running longer than 1 hour to be overrun jobs, you may want to close the queue when LSF detects a job that has run longer than 1 hour and invokes `eadmi n`.

Email job exception details

Set LSF to send you an email about job exceptions that includes details including `JOB_ID`, `RUN_TIME`, `IDLE_FACTOR` (if job has been idle), `USER`, `QUEUE`, `EXEC_HOST`, and `JOB_NAME`.

1. In `lsb.params`, set `EXTEND_JOB_EXCEPTION_NOTIFY=Y`.
2. Set the format option in the `eadmin` script (`LSF_SERVERDIR/eadmin` on the master host).
 - a) Uncomment the `JOB_EXCEPTION_EMAIL_FORMAT` line and add a value for the format:
 - `JOB_EXCEPTION_EMAIL_FORMAT=fixed`: The `eadmin` shell generates an exception email with a fixed length for the job exception information. For any given field, the characters truncate when the maximum is reached (between 10-19).
 - `JOB_EXCEPTION_EMAIL_FORMAT=full`: The `eadmin` shell generates an exception email without a fixed length for the job exception information.

Default eadmin actions

For host-level exceptions, LSF closes the host and sends email to the LSF administrator. The email contains the host name, job exit rate for the host, and other host information. The message `eadmin: JOB EXIT THRESHOLD EXCEEDED` is attached to the closed host event in `lsb.events`, and displayed by `badmin hist` and `badmin hhist`.

For job exceptions, LSF sends email to the LSF administrator. The email contains the job ID, exception type (overrun, underrun, idle job), and other job information.

An email is sent for all detected job exceptions according to the frequency configured by `EADMIN_TRIGGER_DURATION` in `lsb.params`. For example, if `EADMIN_TRIGGER_DURATION` is set to 5 minutes, and 1 overrun job and 2 idle jobs are detected, after 5 minutes, `eadmin` is invoked and only one email is sent. If another overrun job is detected in the next 5 minutes, another email is sent.

Handle job initialization failures

By default, LSF handles job exceptions for jobs that exit after they have started running. You can also configure LSF to handle jobs that exit during initialization because of an execution environment problem, or because of a user action or LSF policy.

LSF detects that the jobs are exiting before they actually start running, and takes appropriate action when the job exit rate exceeds the threshold for specific hosts (`EXIT_RATE` in `lsb.hosts`) or for all hosts (`GLOBAL_EXIT_RATE` in `lsb.params`).

Use `EXIT_RATE_TYPE` in `lsb.params` to include job initialization failures in the exit rate calculation. The following table summarizes the exit rate types you can configure:

Table 1: Exit rate types you can configure

Exit rate type ...	Includes ...
JOBEXIT	Local exited jobs Remote job initialization failures Parallel job initialization failures on hosts other than the first execution host Jobs exited by user action (e.g., <code>bkill</code> , <code>bstop</code> , etc.) or LSF policy (e.g., load threshold exceeded, job control action, advance reservation expired, etc.)

Exit rate type ...	Includes ...
JOBEXIT_NONLSF	Local exited jobs
This is the default when EXIT_RATE_TYPE is not set	Remote job initialization failures Parallel job initialization failures on hosts other than the first execution host
JOBINIT	Local job initialization failures Parallel job initialization failures on the first execution host
HPCINIT	Job initialization failures for HPC jobs

Job exits excluded from exit rate calculation

By default, jobs that are exited for non-host related reasons (user actions and LSF policies) are not counted in the exit rate calculation. Only jobs that are exited for what LSF considers host-related problems and are used to calculate a host exit rate.

The following cases are *not included* in the exit rate calculations:

- `bkill, bkill -r`
- `brequeue`
- RERUNNABLE jobs killed when a host is unavailable
- Resource usage limit exceeded (for example, PROCESSLIMIT, CPULIMIT, etc.)
- Queue-level job control action TERMINATE and TERMINATE_WHEN
- Checkpointing a job with the kill option (`bchkpnt -k`)
- Rerunnable job migration
- Job killed when an advance reservation has expired
- Remote lease job start fails
- Any jobs with an exit code found in SUCCESS_EXIT_VALUES, where a particular exit value is deemed as successful.

Exclude Platform LSF and user-related job exits

To explicitly *exclude* jobs exited because of user actions or LSF-related policies from the job exit calculation, set `EXIT_RATE_TYPE=JOBEXIT_NONLSF` in `lsb.params`. `JOBEXIT_NONLSF` tells LSF to include all job exits *except* those that are related to user action or LSF policy. This is the default value for `EXIT_RATE_TYPE`.

To *include* all job exit cases in the exit rate count, you must set `EXIT_RATE_TYPE = JOBEXIT` in `lsb.params`. `JOBEXIT` considers all job exits.

Jobs killed by signal external to LSF will still be counted towards exit rate

Jobs killed because of job control `SUSPEND` action and `RESUME` action are still counted towards the exit rate. This because LSF cannot distinguish between jobs killed from `SUSPEND` action and jobs killed by external signals.

If both `JOBEXIT` and `JOBEXIT_NONLSF` are defined, `JOBEXIT_NONLSF` is used.

Local jobs

When `EXIT_RATE_TYPE=JOBINIT`, various job initialization failures are included in the exit rate calculation, including:

- Host-related failures; for example, incorrect user account, user permissions, incorrect directories for checkpointable jobs, host name resolution failed, or other execution environment problems
- Job-related failures; for example, pre-execution or setup problem, job file not created, etc.

Parallel jobs

By default, or when `EXIT_RATE_TYPE=JOBEXIT_NONLSF`, job initialization failure on the first execution host does not count in the job exit rate calculation. Job initialization failure for hosts other than the first execution host are counted in the exit rate calculation.

When `EXIT_RATE_TYPE=JOBINIT`, job initialization failure happens on the first execution host are counted in the job exit rate calculation. Job initialization failures for hosts other than the first execution host are *not* counted in the exit rate calculation.

Tip:

For parallel job exit exceptions to be counted for *all* hosts, specify
`EXIT_RATE_TYPE=HPCINIT` or
`EXIT_RATE_TYPE=JOBEXIT_NONLSF JOBINIT`.

Remote jobs

By default, or when `EXIT_RATE_TYPE=JOBEXIT_NONLSF`, job initialization failures are counted as exited jobs on the remote execution host and are included in the exit rate calculation for that host. To include only *local* job initialization failures on the execution cluster from the exit rate calculation, set `EXIT_RATE_TYPE` to include only `JOBINIT` or `HPCINIT`.

Scale and tune job exit rate by number of slots

On large, multiprocessor hosts, use to `ENABLE_EXIT_RATE_PER_SLOT=Y` in `l sb. params` to scale the job exit rate so that the host is only closed when the job exit rate is high enough in proportion to the number of processors on the host. This avoids having a relatively low exit rate close a host inappropriately.

Use a float value for `GLOBAL_EXIT_RATE` in `l sb. params` to tune the exit rate on multislot hosts. The actual calculated exit rate value is never less than 1.

Example: exit rate of 5 on single processor and multiprocessor hosts

On a single-processor host, a job exit rate of 5 is much more severe than on a 20-processor host. If a stream of jobs to a single-processor host is consistently failing, it is reasonable to close the host or take some other action after 5 failures.

On the other hand, for the same stream of jobs on a 20-processor host, it is possible that 19 of the processors are busy doing other work that is running fine. To close this host after only 5 failures would be wrong because effectively less than 5% of the jobs on that host are actually failing.

Example: float value for GLOBAL_EXIT_RATE on multislot hosts

Using a float value for `GLOBAL_EXIT_RATE` allows the exit rate to be less than the number of slots on the host. For example, on a host with 4 slots, `GLOBAL_EXIT_RATE=0.25` gives an exit rate of 1. The same value on an 8 slot machine would be 2 and so on. On a single-slot host, the value is never less than 1.

5

Working with Queues

Queue states

Queue states, displayed by `bqueues`, describe the ability of a queue to accept and start batch jobs using a combination of the following states:

- Open: queues accept new jobs
- Closed: queues do not accept new jobs
- Active: queues start jobs on available hosts
- Inactive: queues hold all jobs

State	Description
Open: Active	Accepts and starts new jobs—normal processing
Open: Inact	Accepts and holds new jobs—collecting
Closed: Active	Does not accept new jobs, but continues to start jobs — draining
Closed: Inact	Does not accept new jobs and does not start jobs—all activity is stopped

Queue state can be changed by an LSF administrator or root.

Queues can also be activated and inactivated by run windows and dispatch windows (configured in `lsb.queues`, displayed by `bqueues -l`).

`bqueues -l` displays `Inact_Adm` when explicitly inactivated by an Administrator (`badmi n qi nact`), and `Inact_Win` when inactivated by a run or dispatch window.

View queue information

The `bqueues` command displays information about queues. The `bqueues -l` option also gives current statistics about the jobs in a particular queue, such as the total number of jobs in the queue, the number of jobs running, suspended, and so on.

To view the...	Run...
Available queues	<code>bqueues</code>
Queue status	<code>bqueues</code>
Detailed queue information	<code>bqueues -l</code>
State change history of a queue	<code>badmi n qhi st</code>
Queue administrators	<code>bqueues -l for queue</code>

In addition to the procedures listed here, see the `bqueues(1)` man page for more details.

View available queues and queue status

1. Run `bqueues`. You can view the current status of a particular queue or all queues. The `bqueues` command also displays available queues in the cluster.

```

bqueues
QUEUE_NAME  PRI O  STATUS  MAX  JL/U  JL/P  JL/H  NJOBS  PEND  RUN  SUSP
interactive  400    Open: Active  -  -  -  -  2  0  2  0
priority    43    Open: Active  -  -  -  -  16  4  11  1
night       40    Open: Inactive  -  -  -  -  4  4  0  0
short       35    Open: Active  -  -  -  -  6  1  5  0
license     33    Open: Active  -  -  -  -  0  0  0  0
normal      30    Open: Active  -  -  -  -  0  0  0  0
idle        20    Open: Active  -  -  -  -  6  3  1  2

```

A dash (-) in any entry means that the column does not apply to the row. In this example no queues have per-queue, per-user, per-processor, or per host job limits configured, so the MAX, JL/U, JL/P, and JL/H entries are shown as a dash.

Job slots required by parallel jobs

Important:

A parallel job with *N* components requires *N* job slots.

View detailed queue information

1. To see the complete status and configuration for each queue, run `bqueues -l`.

Specify queue names to select specific queues. The following example displays details for the queue `normal`.

bqueues -l normal

```

QUEUE: normal
--For normal low priority jobs, running only if hosts are lightly loaded. This is
the default queue.

```

```

PARAMETERS/STATISTICS
PRIO NICE STATUS MAX JL/U JL/P NJOBS PEND RUN SSUSP USUSP
40 20 Open: Active 100 50 11 1 1 0 0 0
Migration threshold is 30 min.
CPULIMIT RUNLIMIT
20 min of IBM350 342800 min of IBM350
FILELIMIT DATALIMIT STACKLIMIT CORELIMIT MEMLIMIT PROCLIMIT
20000 K 20000 K 2048 K 20000 K 5000 K 3

SCHEDULING PARAMETERS r15s r1m r15m ut pg io ls it tmp swp mem
loadSched - 0.7 1.0 0.2 4.0 50 - - - - -
loadStop - 1.5 2.5 - 8.0 240 - - - - -

loadSched cpuspeed bandwidth
loadStop - -

SCHEDULING POLICIES: FAIRSHARE PREEMPTIVE PREEMPTABLE EXCLUSIVE
USER_SHARES: [groupA, 70] [groupB, 15] [default, 1]

DEFAULT HOST SPECIFICATION : IBM350

RUN_WINDOWS: 2: 40-23:00 23:30-1:30
DISPATCH_WINDOWS: 1: 00-23:50

USERS: groupA/ groupB/ user5
HOSTS: hostA, hostD, hostB
ADMINISTRATORS: user7
PRE_EXEC: /tmp/apex_pre.x > /tmp/preexec.log 2>&1
POST_EXEC: /tmp/apex_post.x > /tmp/postexec.log 2>&1
REQUEUE_EXIT_VALUES: 45

```

View the state change history of a queue

1. Run `badmin qhist` to display the times when queues are opened, closed, activated, and inactivated.

```

badmin qhist
Wed Mar 31 09:03:14: Queue <normal> closed by user or administrator <root>.
Wed Mar 31 09:03:29: Queue <normal> opened by user or administrator <root>.

```

View queue administrators

1. Run `bqueues -l` for the queue.

View exception status for queues (bqueues)

1. Use `bqueues` to display the configured threshold for job exceptions and the current number of jobs in the queue in each exception state.

For example, queue `normal` configures `JOB_IDLE` threshold of 0.10, `JOB_OVERRUN` threshold of 5 minutes, and `JOB_UNDERRUN` threshold of 2 minutes. The following `bqueues` command shows no overrun jobs, one job that finished in less than 2 minutes (underrun) and one job that triggered an idle exception (less than idle factor of 0.10):

```

bqueues -l normal
QUEUE: normal
-- For normal low priority jobs, running only if hosts are lightly loaded. This is the default queue.

PARAMETERS/STATISTICS
PRIO NICE STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SSUSP USUSP RSV
30 20 Open: Active - - - - 0 0 0 0 0 0

STACKLIMIT MEMLIMIT
2048 K 5000 K

```

SCHEDULING PARAMETERS

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

	cpuspeed	bandwidth
loadSched	-	-
loadStop	-	-

JOB EXCEPTION PARAMETERS

	OVERRUN(min)	UNDERRUN(min)	IDLE(cputime/runtime)
Threshold	5	2	0.10
Jobs	0	1	1

USERS: all users
 HOSTS: all allremote
 CHUNK_JOB_SIZE: 3

Control queues

Queues are controlled by:

- an LSF Administrator or root issuing a command
- configured dispatch and run windows

Close a queue

1. Run `badmi n qclose`:

```
badmi n qclose normal  
Queue <normal> is closed
```

When a user tries to submit a job to a closed queue the following message is displayed:

```
bsub -q normal ...  
normal: Queue has been closed
```

Open a queue

1. Run `badmi n qopen`:

```
badmi n qopen normal  
Queue <normal> is opened
```

Deactivate a queue

1. Run `badmi n qi nact`:

```
badmi n qi nact normal  
Queue <normal> is inactivated
```

Activate a queue

1. Run `badmi n qact`:

```
badmi n qact normal  
Queue <normal> is activated
```

Log a comment when controlling a queue

1. Use the `-C` option of `badmi n` queue commands `qclose`, `qopen`, `qact`, and `qi nact` to log an administrator comment in `lsb.events`.

```
badmi n qclose -C "change configuration" normal
```

The comment text `change configuration` is recorded in `lsb.events`.

A new event record is recorded for each queue event. For example:

```
badmi n qclose -C "add user" normal
```

followed by

```
badmi n qclose -C "add user user1" normal
```

will generate records in lsb. events:

```
"QUEUE_CTRL" "7.0 1050082373 1 "normal" 32185 "lsfadmin" "add user"
"QUEUE_CTRL" "7.0 1050082380 1 "normal" 32185 "lsfadmin" "add user user1"
```

2. Use `badmi n hi st` or `badmi n qhi st` to display administrator comments for closing and opening hosts.

```
badmi n qhist
```

```
Fri Apr 4 10:50:36: Queue <normal> closed by administrator <lsfadmin> change
configuration.
```

`bqueues -l` also displays the comment text:

```
bqueues -l normal
```

```
QUEUE: normal
```

```
-- For normal low priority jobs, running only if hosts are lightly loaded. This is the default
queue.
```

```
PARAMETERS/STATISTICS
```

```
PRI O NICE STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SSUSP  USUSP  RSV
 30  20 Closed:Active      -  -  -  -  0    0    0    0    0  0 Interval for a host
to accept two jobs is 0 seconds
```

```
THREADLIMIT
7
```

```
SCHEDULING PARAMETERS
```

```
      r15s  r1m  r15m  ut      pg      io      ls      it      tmp      swp      mem
loadSched -    -    -    -      -      -      -      -      -      -      -
loadStop  -    -    -    -      -      -      -      -      -      -      -
```

```
      cpuspeed      bandwidth
loadSched      -      -
loadStop      -      -
```

```
JOB EXCEPTION PARAMETERS
```

```
      OVERRUN(mi n)  UNDERRUN(mi n)  IDLE(cputime/runtime)
Threshold      -      2      -
Jobs           -      0      -
```

```
USERS: all users
```

```
HOSTS: all
```

```
RES_REQ: select[type==any]
```

```
ADMIN ACTION COMMENT: "change configuration"
```

Configure dispatch windows

A dispatch window specifies one or more time periods during which batch jobs are dispatched to run on hosts. Jobs are not dispatched outside of configured windows. Dispatch windows do not affect job submission and running jobs (they are allowed to run until completion). By default, queues are always Active; you must explicitly configure dispatch windows in the queue to specify a time when the queue is Inactive.

To configure a dispatch window:

1. Edit lsb. queues
2. Create a DISPATCH_WINDOW keyword for the queue and specify one or more time windows.

```
Begin Queue
QUEUE_NAME = queue1
PRIORITY   = 45
DISPATCH_WINDOW = 4:30-12:00
```

```
End Queue
```

3. Reconfigure the cluster:
 - a) Run `lsadmin reconfig`.
 - b) Run `badmin reconfig`.
4. Run `bqueues -l` to display the dispatch windows.

Configure run windows

A run window specifies one or more time periods during which jobs dispatched from a queue are allowed to run. When a run window closes, running jobs are suspended, and pending jobs remain pending. The suspended jobs are resumed when the window opens again. By default, queues are always Active and jobs can run until completion. You must explicitly configure run windows in the queue to specify a time when the queue is Inactive.

To configure a run window:

1. Edit `lsb.queues`.
2. Create a `RUN_WINDOW` keyword for the queue and specify one or more time windows.

```
Begin Queue QUEUE_NAME = queue1 PRIORITY = 45 RUN_WINDOW = 4:30-12:00 End Queue
```

3. Reconfigure the cluster:
 - a) Run `lsadmin reconfig`.
 - b) Run `badmin reconfig`.
4. Run `bqueues -l` to display the run windows.

Add a queue

1. Log in as the LSF administrator on any host in the cluster.
2. Edit `lsb.queues` to add the new queue definition.

You can copy another queue definition from this file as a starting point; remember to change the `QUEUE_NAME` of the copied queue.

3. Save the changes to `lsb.queues`.
4. Run `badmin reconfig` to reconfigure `mbatchd`.

Adding a queue does not affect pending or running jobs.

Example queue:

```
Begin Queue
QUEUE_NAME = normal
PRIORITY = 30
STACKLIMIT= 2048
DESCRIPTION = For normal low priority jobs, running only if hosts are lightly
loaded.
QJOB_LIMIT = 60      # job limit of the queue
PJOB_LIMIT = 2       # job limit per processor
ut = 0.2
io = 50/240
USERS = all
HOSTS = all
NICE = 20
End Queue
```


Remove a queue

Important:

Before removing a queue, make sure there are no jobs in that queue.

If there are jobs in the queue, move pending and running jobs to another queue, then remove the queue. If you remove a queue that has jobs in it, the jobs are temporarily moved to a queue named `lost_and_found`. Jobs in the `lost_and_found` queue remain pending until the user or the LSF administrator uses the `bswitch` command to switch the jobs into an existing queue. Jobs in other queues are not affected.

1. Log in as the LSF administrator on any host in the cluster.
2. Close the queue to prevent any new jobs from being submitted.

```
badadmin qclose nightQueue <night> is closed
```

3. Move all pending and running jobs into another queue.

Below, the `bswitch -q night` argument chooses jobs from the `night` queue, and the job ID number 0 specifies that all jobs should be switched:

bjobs -u all -q night

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
5308	user5	RUN	night	hostA	hostD	job5	Nov 21 18:16
5310	user5	PEND	night	hostA	hostC	job10	Nov 21 18:17

bswitch -q night idle 0

```
Job <5308> is switched to queue <idle>
Job <5310> is switched to queue <idle>
```

4. Edit `lsb.queues` and remove or comment out the definition for the queue being removed.
5. Save the changes to `lsb.queues`.
6. Run `badadmin reconfig` to reconfigure `mbatchd`.

Restrict host use by queues

You may want a host to be used only to run jobs submitted to specific queues. For example, if you just added a host for a specific department such as engineering, you may only want jobs submitted to the queues `engi neeri ng1` and `engi neeri ng2` to be able to run on the host.

1. Log on as root or the LSF administrator on any host in the cluster.
2. Edit `lsb.queues`, and add the host to the `HOSTS` parameter of specific queues.

```
Begin Queue
QUEUE_NAME = queue1
...
HOSTS=mynewhost hostA hostB
...
End Queue
```

3. Save the changes to `lsb.queues`.
4. Use `badadmin ckconfig` to check the new queue definition. If any errors are reported, fix the problem and check the configuration again.
5. Run `badadmin reconfig` to reconfigure `mbatchd`.
6. If you add a host to a queue, the new host will not be recognized by jobs that were submitted before you reconfigured. If you want the new host to be recognized, you must use the command `badadmin mbdrestart`.

Add queue administrators

Queue administrators are optionally configured after installation. They have limited privileges; they can perform administrative operations (open, close, activate, inactivate) on the specified queue, or on jobs running in the specified queue. Queue administrators cannot modify configuration files, or operate on LSF daemons or on queues they are not configured to administer.

To switch a job from one queue to another, you must have administrator privileges for both queues.

1. In the `lsb.queues` file, between `Begin Queue` and `End Queue` for the appropriate queue, specify the `ADMINISTRATORS` parameter, followed by the list of administrators for that queue. Separate the administrator names with a space. You can specify user names and group names.

```
Begin Queue
ADMINISTRATORS = User1 GroupA
End Queue
```

Handle job exceptions in queues

You can configure queues so that LSF detects exceptional conditions while jobs are running, and take appropriate action automatically. You can customize what exceptions are detected, and the corresponding actions. By default, LSF does not detect any exceptions.

Job exceptions LSF can detect

If you configure job exception handling in your queues, LSF detects the following job exceptions:

- Job underrun — jobs end too soon (run time is less than expected). Underrun jobs are detected when a job exits abnormally
- Job overrun — job runs too long (run time is longer than expected). By default, LSF checks for overrun jobs every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for job overrun.
- Idle job — running job consumes less CPU time than expected (in terms of CPU time/runtime). By default, LSF checks for idle jobs every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for idle jobs.

Configure job exception handling (lsb.queues)

You can configure your queues to detect job exceptions. Use the following parameters:

JOB_IDLE

Specify a threshold for idle jobs. The value should be a number between 0.0 and 1.0 representing CPU time/runtime. If the job idle factor is less than the specified threshold, LSF invokes `eadmi n` to trigger the action for a job idle exception.

JOB_OVERRUN

Specify a threshold for job overrun. If a job runs longer than the specified run time, LSF invokes `eadmi n` to trigger the action for a job overrun exception.

JOB_UNDERRUN

Specify a threshold for job underrun. If a job exits before the specified number of minutes, LSF invokes `eadmi n` to trigger the action for a job underrun exception.

Example

The following queue defines thresholds for all types job exceptions:

```
Begin Queue
...
JOB_UNDERRUN = 2
JOB_OVERRUN  = 5
JOB_IDLE     = 0.10
...
End Queue
```

For this queue:

- A job underrun exception is triggered for jobs running less than 2 minutes
- A job overrun exception is triggered for jobs running longer than 5 minutes
- A job idle exception is triggered for jobs with an idle factor (CPU time/runtime) less than 0.10

Configure thresholds for job exception handling

By default, LSF checks for job exceptions every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for overrun, underrun, and idle jobs.

Tuning

Tip:

Tune `EADMIN_TRIGGER_DURATION` carefully. Shorter values may raise false alarms, longer values may not trigger exceptions frequently enough.

Platform LSF Resources

About Platform LSF resources

The LSF system uses built-in and configured resources to track job resource requirements and schedule jobs according to the resources available on individual hosts.

View cluster resources (lsinfo)

1. Use `lsinfo` to list the resources available in your cluster.

The `lsinfo` command lists all resource names and descriptions.

lsinfo			
RESOURCE_NAME	TYPE	ORDER	DESCRIPTION
r15s	Numeric	Inc	15-second CPU run queue length
r1m	Numeric	Inc	1-minute CPU run queue length (alias: cpu)
r15m	Numeric	Inc	15-minute CPU run queue length
ut	Numeric	Inc	1-minute CPU utilization (0.0 to 1.0)
pg	Numeric	Inc	Paging rate (pages/second)
io	Numeric	Inc	Disk I/O rate (Kbytes/second)
ls	Numeric	Inc	Number of login sessions (alias: login)
it	Numeric	Dec	Idle time (minutes) (alias: idle)
tmp	Numeric	Dec	Disk space in /tmp (Mbytes)
swp	Numeric	Dec	Available swap space (Mbytes) (alias: swap)
mem	Numeric	Dec	Available memory (Mbytes)
ncpus	Numeric	Dec	Number of CPUs
nprocs	Numeric	Dec	Number of physical processors
ncores	Numeric	Dec	Number of cores per physical processor
nthreads	Numeric	Dec	Number of threads per processor
corendisks	Numeric	Dec	Number of local disks
maxmem	Numeric	Dec	Maximum memory (Mbytes)
maxswp	Numeric	Dec	Maximum swap space (Mbytes)
maxtmp	Numeric	Dec	Maximum /tmp space (Mbytes)
cpuf	Numeric	Dec	CPU factor
...			

View host resources (lshosts)

1. Run `lshosts` for a list of the resources defined on a specific host:

lshosts hostA								
HOST_NAME	type	model	cpuf	ncpus	maxmem	maxswp	server	RESOURCES
hostA	SOL732	Ultra2	20.2	2	256M	679M	Yes	()

View host load by resource

1. Run `lshosts -s` to view host load by shared resource:

lshosts -s		
RESOURCE	VALUE	LOCATION
tot_lic	5	host1 host2
tot_scratch	500	host1 host2

The above output indicates that 5 licenses are available, and that the shared scratch directory currently contains 500 MB of space.

The VALUE field indicates the amount of that resource. The LOCATION column shows the hosts which share this resource. The `lshosts -s` command displays static shared resources. The `lsl load -s` command displays dynamic shared resources.

Resource categories

By values

Boolean resources	Resources that denote the availability of specific features
Numerical resources	Resources that take numerical values, such as all the load indices, number of processors on a host, or host CPU factor
String resources	Resources that take string values, such as host type, host model, host status

By the way values change

Dynamic Resources	Resources that change their values dynamically: host status and all the load indices.
Static Resources	Resources that do not change their values: all resources except for load indices or host status.

By definitions

External Resources	Custom resources defined by user sites: external load indices and resources defined in the <code>lsf.shared</code> file (shared resources).
Built-In Resources	Resources that are always defined in LSF, such as load indices, number of CPUs, or total swap space.

By scope

Host-Based Resources	Resources that are not shared among hosts, but are tied to individual hosts, such as swap space, CPU, or memory. An application must run on a particular host to access the resources. Using up memory on one host does not affect the available memory on another host.
Shared Resources	Resources that are not associated with individual hosts in the same way, but are owned by the entire cluster, or a subset of hosts within the cluster, such as floating licenses or shared file systems. An application can access such a resource from any host which is configured to share it, but doing so affects its value as seen by other hosts.

Boolean resources

Boolean resources (for example, `server` to denote LSF server hosts) have a value of one (1) if they are defined for a host, and zero (0) if they are not defined for the host. Use Boolean resources to configure host attributes to be used in selecting hosts to run jobs. For example:

- Machines may have different types and versions of operating systems.
- Machines may play different roles in the system, such as file server or compute server.
- Some machines may have special-purpose devices needed by some applications.
- Certain software packages or licenses may be available only on some of the machines.

Specify a Boolean resource in a resource requirement selection string of a job to select only hosts that can run the job.

Some examples of Boolean resources:

Resource Name	Describes	Meaning of Example Name
cs	Role in cluster	Compute server
fs	Role in cluster	File server
solari s	Operating system	Solaris operating system
frame	Available software	FrameMaker license

Shared resources

Shared resources are configured resources that are not tied to a specific host, but are associated with the entire cluster, or a specific subset of hosts within the cluster. For example:

- Floating licenses for software packages
- Disk space on a file server which is mounted by several machines
- The physical network connecting the hosts

LSF does not contain any built-in shared resources. All shared resources must be configured by the LSF administrator. A shared resource may be configured to be dynamic or static. In the above example, the total space on the shared disk may be static while the amount of space currently free is dynamic. A site may also configure the shared resource to report numeric, string or Boolean values.

An application may use a shared resource by running on any host from which that resource is accessible. For example, in a cluster in which each host has a local disk but can also access a disk on a file server, the disk on the file server is a shared resource, and the local disk is a host-based resource. In contrast to host-based resources such as memory or swap space, using a shared resource from one machine affects the availability of that resource as seen by other machines. There is one value for the entire cluster which measures the utilization of the shared resource, but each host-based resource is measured separately.

The following restrictions apply to the use of shared resources in LSF products.

- A shared resource cannot be used as a load threshold in the `Hosts` section of the `lsf.cluster.cluster_name` file.
- A shared resource cannot be used in the `loadSched/loadStop` thresholds, or in the `STOP_COND` or `RESUME_COND` parameters in the queue definition in the `lsb.queues` file.

View shared resources for hosts

1. Run `bhosts -s` to view shared resources for hosts. For example:

```
bhosts -s
RESOURCE      TOTAL    RESERVED    LOCATION
tot_lic        5         0.0        hostA hostB
tot_scratch    00         0.0        hostA hostB
avail_lic       2         3.0        hostA hostB
avail_scratch  100       400.0      hostA hostB
```

The `TOTAL` column displays the value of the resource. For dynamic resources, the `RESERVED` column displays the amount that has been reserved by running jobs.

How LSF uses resources

Jobs submitted through LSF have resource usage monitored while they are running. This information is used to enforce resource usage limits and load thresholds as well as for fairshare scheduling.

LSF collects information such as:

- Total CPU time consumed by all processes in the job
- Total resident memory usage in KB of all currently running processes in a job
- Total virtual memory usage in KB of all currently running processes in a job
- Currently active process group ID in a job
- Currently active processes in a job

On UNIX, job-level resource usage is collected through a special process called PIM (Process Information Manager). PIM is managed internally by LSF.

View job resource usage

- Run `bj obs -l` to display the current resource usage of the job.

Usage information is sampled by PIM every 30 seconds and collected by `sbat chd` at a maximum frequency of every `SBD_SLEEP_TIME` (configured in the `l sb. params` file) and sent to `mbat chd`.

An update occurs only if the value for the CPU time, resident memory usage, or virtual memory usage has changed by more than 10 percent from the previous update, or if a new process or process group has been created.

View load on a host

1. Run `bhosts -l` to check the load levels on the host.

A dash (-) in the output indicates that the particular threshold is not defined.

```
bhosts -l hostB
HOST: hostB
STATUS          CPUF  JL/U  MAX NJOBS  RUN  SSUSP  USUSP  RSV
ok              20.00  2    2    0    0    0    0    0

CURRENT LOAD USED FOR SCHEDULING:
          r15s  r1m  r15m  ut   pg   io   ls   t   tmp   swp
mem
Total    0.3   0.8  0.9   61%  3.8  72   26   0   6M   253
M 297M
Reserved 0.0   0.0  0.0   0%   0.0  0    0    0   0M   0M
OM

LOAD THRESHOLD USED FOR SCHEDULING:
          r15s  r1m  r15m  ut   pg   io   ls   it  tmp  swp  mem
loadSched -    -    -    -    -    -    -    -    -    -    -
loadStop  -    -    -    -    -    -    -    -    -    -    -

          cpuspeed  bandwidth
loadSched          -    -
loadStop           -    -
```

Load indices

Load indices are built-in resources that measure the availability of static or dynamic, non-shared resources on hosts in the LSF cluster.

Load indices built into the LIM are updated at fixed time intervals.

External load indices are defined and configured by the LSF administrator, who writes an external load information manager (`el i m`) executable. The `el i m` collects the values of the external load indices and sends these values to the LIM.

Load indices collected by LIM

Index	Measures	Units	Direction	Averaged over	Update Interval
<code>st at us</code>	host status	string			15 seconds
<code>r 15s</code>	run queue length	processes	increasing	15 seconds	15 seconds
<code>r 1m</code>	run queue length	processes	increasing	1 minute	15 seconds
<code>r 15m</code>	run queue length	processes	increasing	15 minutes	15 seconds
<code>ut</code>	CPU utilization	percent	increasing	1 minute	15 seconds
<code>pg</code>	paging activity	pages in + pages out per second	increasing	1 minute	15 seconds
<code>l s</code>	logins	users	increasing	N/A	30 seconds
<code>i t</code>	idle time	minutes	decreasing	N/A	30 seconds
<code>swp</code>	available swap space	MB	decreasing	N/A	15 seconds
<code>mem</code>	available memory	MB	decreasing	N/A	15 seconds
<code>t mp</code>	available space in temporary file system	MB	decreasing	N/A	120 seconds
<code>i o</code>	disk I/O (shown by <code>lsload -l</code>)	KB per second	increasing	1 minute	15 seconds
<i>name</i>	external load index configured by LSF administrator				site-defined

Status

The `st at us` index is a string indicating the current status of the host. This status applies to the LIM and RES.

The possible values for `st at us` are:

Status	Description
<code>ok</code>	The host is available to accept remote jobs. The LIM can select the host for remote execution.
<code>- ok</code>	When the status of a host is preceded by a dash (-), it means LIM is available but RES is not running on that host or is not responding.

Status	Description
busy	The host is overloaded (busy) because a load index exceeded a configured threshold. An asterisk (*) marks the offending index. LIM will not select the host for interactive jobs.
lockW	The host is locked by its run window. Use lshosts to display run windows.
lockU	The host is locked by an LSF administrator or root.
unavail	The host is down or the LIM on the host is not running or is not responding.
unlicensed	The host does not have a valid license.

Note:

The term `availabl` is frequently used in command output titles and headings. `Availabl` means a host is in any state except `unavail`. This means an `availabl` host could be `unlicensed`, `locked`, `busy`, or `ok`.

CPU run queue lengths (r15s, r1m, r15m)

The `r15s`, `r1m` and `r15m` load indices are the 15-second, 1-minute and 15-minute average CPU run queue lengths. This is the average number of processes ready to use the CPU during the given interval.

On UNIX, run queue length indices are not necessarily the same as the load averages printed by the `uptime(1)` command; `uptime` load averages on some platforms also include processes that are in short-term wait states (such as paging or disk I/O).

Effective run queue length

On multiprocessor systems, more than one process can execute at a time. LSF scales the run queue value on multiprocessor systems to make the CPU load of uniprocessors and multiprocessors comparable. The scaled value is called the effective run queue length.

Use `lslload -E` to view the effective run queue length.

Normalized run queue length

LSF also adjusts the CPU run queue based on the relative speeds of the processors (the CPU factor). The normalized run queue length is adjusted for both number of processors and CPU speed. The host with the lowest normalized run queue length runs a CPU-intensive job the fastest.

Use `lslload -N` to view the normalized CPU run queue lengths.

CPU utilization (ut)

The `ut` index measures CPU utilization, which is the percentage of time spent running system and user code. A host with no process running has a `ut` value of 0 percent; a host on which the CPU is completely loaded has a `ut` of 100 percent.

Paging rate (pg)

The `pg` index gives the virtual memory paging rate in pages per second. This index is closely tied to the amount of available RAM memory and the total size of the processes running on a host; if there is not

enough RAM to satisfy all processes, the paging rate is high. Paging rate is a good measure of how a machine responds to interactive use; a machine that is paging heavily feels very slow.

Login sessions (ls)

The `ls` index gives the number of users logged in. Each user is counted once, no matter how many times they have logged into the host.

Interactive idle time (it)

On UNIX, the `it` index is the interactive idle time of the host, in minutes. Idle time is measured from the last input or output on a directly attached terminal or a network pseudo-terminal supporting a login session. This does not include activity directly through the X server such as CAD applications or emacs windows, except on Solaris and HP-UX systems.

On Windows, the `it` index is based on the time a screen saver has been active on a particular host.

Temporary directories (tmp)

The `tmp` index is the space available in MB on the file system that contains the temporary directory:

- `/tmp` on UNIX
- `C:\temp` on Windows

Swap space (swp)

The `swp` index gives the currently available virtual memory (swap space) in MB. This represents the largest process that can be started on the host.

Memory (mem)

The `mem` index is an estimate of the real memory currently available to user processes. This represents the approximate size of the largest process that could be started on a host without causing the host to start paging.

LIM reports the amount of free memory available. LSF calculates free memory as a sum of physical free memory, cached memory, buffered memory and an adjustment value. The command `vmstat` also reports free memory but displays these values separately. There may be a difference between the free memory reported by LIM and the free memory reported by `vmstat` because of virtual memory behavior variations among operating systems. You can write an ELIM that overrides the free memory values returned by LIM.

I/O rate (io)

The `io` index measures I/O throughput to disks attached directly to this host, in KB per second. It does not include I/O to disks that are mounted from other hosts.

View information about load indices

lsinfo -l

The `lsinfo -l` command displays all information available about load indices in the system. You can also specify load indices on the command line to display information about selected indices:

```
lsinfo -l swp
RESOURCE_NAME:  swp
DESCRIPTION:    Available swap space (Mbytes) (alias: swap)
TYPE            ORDER  INTERVAL  BUILTIN  DYNAMIC  RELEASE
Numeric         Dec    60       Yes      Yes      NO
```

lsload -l

The `lsload -l` command displays the values of all load indices. External load indices are configured by your LSF administrator:

```
lsload
HOST_NAME  status  r15s  r1m  r15m  ut    pg    ls    it    tmp    swp    mem
hostN      ok      0.0   0.0   0.1   1%    0.0   1     224   43M    67M    3M
hostK      -ok     0.0   0.0   0.0   3%    0.0   3     0     38M    40M    7M
hostF      busy    0.1   0.1   0.3   7%    *17   6     0     9M     23M    28M
hostG      busy    *6.2   6.9   9.5   85%   1.1   30    0     5M     400M   385M
hostV      unavail
```

Static resources

Static resources are built-in resources that represent host information that does not change over time, such as the maximum RAM available to user processes or the number of processors in a machine. Most static resources are determined by the LIM at start-up time, or when LSF detects hardware configuration changes.

Static resources can be used to select appropriate hosts for particular jobs based on binary architecture, relative CPU speed, and system configuration.

The resources `ncpus`, `nprocs`, `ncores`, `ntthreads`, `maxmem`, `maxswp`, and `maxtmp` are not static on UNIX hosts that support dynamic hardware reconfiguration.

Static resources reported by LIM

Index	Measures	Units	Determined by
<code>type</code>	host type	string	configuration
<code>model</code>	host model	string	configuration
<code>hname</code>	host name	string	configuration
<code>cpuf</code>	CPU factor	relative	configuration
<code>server</code>	host can run remote jobs	Boolean	configuration
<code>rexpri</code>	execution priority	ni ce(2) argument	configuration
<code>ncpus</code>	number of processors	processors	LIM
<code>ndisks</code>	number of local disks	disks	LIM
<code>nprocs</code>	number of physical processors	processors	LIM
<code>ncores</code>	number of cores per physical processor	cores	LIM
<code>ntthreads</code>	number of threads per processor core	threads	LIM
<code>maxmem</code>	maximum RAM	MB	LIM
<code>maxswp</code>	maximum swap space	MB	LIM
<code>maxtmp</code>	maximum space in <code>/tmp</code>	MB	LIM

Host type (`type`)

Host type is a combination of operating system and CPU architecture. All computers that run the same operating system on the same computer architecture are of the same type. You can add custom host types in the `HostType` section of `lsf.conf`. This alphanumeric value can be up to 39 characters long.

An example of host type is `LINUX86`.

Host model (`model`)

Host model is the combination of host type and CPU speed (CPU factor) of your machine. All hosts of the same relative type and speed are assigned the same host model. You can add custom host models in the `HostModel` section of `lsf.conf`. This alphanumeric value can be up to 39 characters long.

An example of host model is `Intel_IA64`.

Host name (hname)

Host name specifies the name with which the host identifies itself.

CPU factor (cpuf)

The CPU factor (frequently shortened to `cpuf`) represents the speed of the host CPU relative to other hosts in the cluster. For example, if one processor is twice the speed of another, its CPU factor should be twice as large. For multiprocessor hosts, the CPU factor is the speed of a single processor; LSF automatically scales the host CPU load to account for additional processors. The CPU factors are detected automatically or defined by the administrator.

Server

The server static resource is Boolean. It has the following values:

- 1 if the host is configured to run jobs from other hosts
- 0 if the host is an LSF client for submitting jobs to other hosts

Number of CPUs (ncpus)

By default, the number of CPUs represents the number of cores a machine has. As most CPUs consist of multiple cores, threads, and processors, `ncpus` can be defined by the cluster administrator (either globally or per-host) to consider one of the following:

- Processors
- Processors and cores
- Processors, cores, and threads

Globally, this definition is controlled by the parameter `EGO_DEFINE_NCPUS` in `lsf.conf` or `ego.conf`. The default behavior for `ncpus` is to consider the number of cores (`EGO_DEFINE_NCPUS=cores`).

Note:

1. On a machine running AIX, `ncpus` detection is different. Under AIX, the number of detected physical processors is always 1, whereas the number of detected cores is the number of cores across all physical processors. Thread detection is the same as other operating systems (the number of threads per core).
 2. When `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the resource requirement string keyword `ncpus` refers to the number of slots instead of the number of CPUs, however `lshosts` output continues to show `ncpus` as defined by `EGO_DEFINE_NCPUS` in `lsf.conf`.
-

Number of disks (ndisks)

The number of disks specifies the number of local disks a machine has, determined by the LIM.

Maximum memory (maxmem)

Maximum memory is the total available memory of a machine, measured in megabytes (MB).

Maximum swap (maxswp)

Maximum swap is the total available swap space a machine has, measured in megabytes (MB).

Maximum temporary space (maxtmp)

Maximum temporary space is the total temporary space a machine has, measured in megabytes (MB).

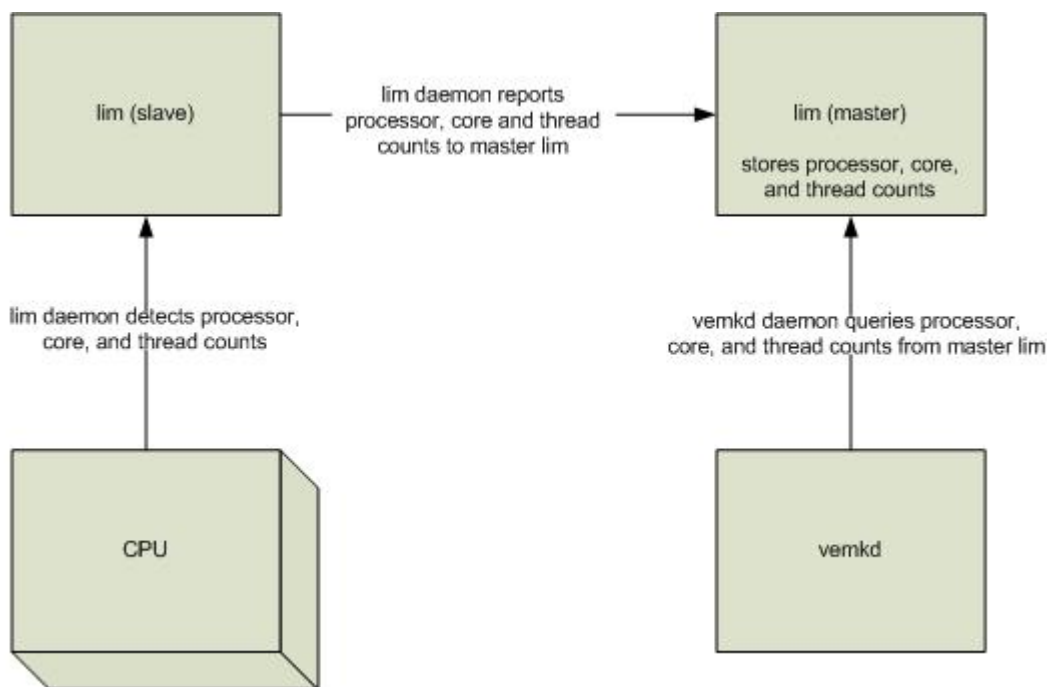
How LIM detects cores, threads, and processors

Traditionally, the value of `ncpus` has been equal to the number of physical CPUs. However, many CPUs consist of multiple cores and threads, so the traditional 1:1 mapping is no longer useful. A more useful approach is to set `ncpus` to equal one of the following:

- The number of processors
- Cores—the number of cores (per processor) * the number of processors (this is the `ncpus` default setting)
- Threads—the number of threads (per core) * the number of cores (per processor) * the number of processors

A cluster administrator globally defines how `ncpus` is computed using the `EGO_DEFINE_NCPUS` parameter in `lsf.conf` or `ego.conf` (instead of `LSF_ENABLE_DUALCORE` in `lsf.conf`, or `EGO_ENABLE_DUALCORE` in `ego.conf`).

LIM detects and stores the number of processors, cores, and threads for all supported architectures. The following diagram illustrates the flow of information between daemons, CPUs, and other components.



Although the `ncpus` computation is applied globally, it can be overridden on a per-host basis.

To correctly detect processors, cores, and threads, LIM assumes that all physical processors on a single machine are of the same type.

In cases where CPU architectures and operating system combinations may not support accurate processor, core, thread detection, LIM uses the defaults of 1 processor, 1 core per physical processor, and

1 thread per core. If LIM detects that it is running in a virtual environment (for example, VMware®), each detected processor is similarly reported (as a single-core, single-threaded, physical processor).

LIM only detects hardware that is recognized by the operating system. LIM detection uses processor- or OS-specific techniques (for example, the Intel CPUID instruction, or Solaris `kstat() /core_id`). If the operating system does not recognize a CPU or core (for example, if an older OS does not recognize a quad-core processor and instead detects it as dual-core), then LIM does not recognize it either.

Note:

RQL normalization never considers threads. Consider a hyper-thread enabled Pentium: Threads are not full-fledged CPUs, so considering them as CPUs would artificially lower the system load.

ncpus detection on AIX

On a machine running AIX, detection of ncpus is different. Under AIX, the number of detected physical processors is always 1, whereas the number of detected cores is always the number of cores across all physical processors. Thread detection is the same as other operating systems (the number of threads per core).

Define ncpus—processors, cores, or threads

A cluster administrator must define how ncpus is computed. Usually, the number of available job slots is equal to the value of ncpus; however, slots can be redefined at the EGO resource group level. The ncpus definition is globally applied across the cluster.

1. Open `lsf.conf` or `ego.conf`.

- UNIX and Linux:

```
LSF_CONFDIR/lsf.conf
```

```
LSF_CONFDIR/ego/cluster_name/kernel/ego.conf
```

- Windows:

```
LSF_CONFDIR\lsf.conf
```

```
LSF_CONFDIR\ego\cluster_name\kernel\ego.conf
```

Important:

You can set `EGO_DEFINE_NCPUS` in `ego.conf` only if EGO is enabled in the LSF cluster. If EGO is not enabled, you must set `EGO_DEFINE_NCPUS` in `lsf.conf`.

2. Define the parameter `EGO_DEFINE_NCPUS=[procs | cores | threads]`.

Set it to one of the following:

- `procs` (where `ncpus=procs`)
- `cores` (where `ncpus=procs * cores`)
- `threads` (where `ncpus=procs * cores * threads`)

By default, ncpus is set to `cores` (number of cores).

Note:

In clusters with older LIMs that do not recognize cores and threads, `EGO_DEFINE_NCPUS` is ignored. In clusters where only the master

LIM recognizes cores and threads, the master LIM assigns default values (for example, in Platform LSF 6.2: 1 core, 1 thread).

3. Save and close `lsf.conf` or `ego.conf`.

Tip:

As a best practice, set `EGO_DEFINE_NCPUS` instead of `EGO_ENABLE_DUALCORE`. The functionality of `EGO_ENABLE_DUALCORE=y` is preserved by setting `EGO_DEFINE_NCPUS=cores`.

Interaction with LSF_LOCAL_RESOURCES in lsf.conf

If EGO is enabled, and `EGO_LOCAL_RESOURCES` is set in `ego.conf` and `LSF_LOCAL_RESOURCES` is set in `lsf.conf`, `EGO_LOCAL_RESOURCES` takes precedence.

Define computation of ncpus on dynamic hosts

The `ncpus` global definition can be overridden on specified dynamic and static hosts in the cluster.

1. Open `lsf.conf` or `ego.conf`.

- UNIX and Linux:

`LSF_CONFDIR/lsf.conf`

`LSF_CONFDIR/ego/cluster_name/kernel/ego.conf`

- Windows:

`LSF_CONFDIR\lsf.conf`

`LSF_CONFDIR\ego\cluster_name\kernel\ego.conf`

Important:

You can set `EGO_LOCAL_RESOURCES` in `ego.conf` only if EGO is enabled in the LSF cluster. If EGO is not enabled, you must set `EGO_LOCAL_RESOURCES` in `lsf.conf`.

2. Define the parameter `EGO_LOCAL_RESOURCES="[resource resource_name]"`.

Set *resource_name* to one of the following:

- `define_ncpus_procs`
- `define_ncpus_cores`
- `define_ncpus_threads`

Note:

Resource definitions are mutually exclusive. Choose only one resource definition per host.

For example:

- Windows: `EGO_LOCAL_RESOURCES="[type NTX86] [resource define_ncpus_procs]"`
- Linux: `EGO_LOCAL_RESOURCES="[resource define_ncpus_cores]"`

3. Save and close `ego.conf`.

Note:

In multi-cluster environments, if `ncpus` is defined on a per-host basis (thereby overriding the global setting) the definition is applied to *all* clusters that the host is a part of. In contrast, globally defined `ncpus` settings *only* take effect within the cluster for which `EGO_DEFINE_NCPUS` is defined.

Define computation of `ncpus` on static hosts

The `ncpus` global definition can be overridden on specified dynamic and static hosts in the cluster.

1. Open `lsf.cluster.cluster_name`.
 - Linux: `LSF_CONFDIR/lsf.cluster.cluster_name`
 - Windows: `LSF_CONFDIR\lsf.cluster.cluster_name`
2. Find the host you for which you want to define `ncpus` computation. In the `RESOURCES` column, add one of the following definitions:
 - `define_ncpus_procs`
 - `define_ncpus_cores`
 - `define_ncpus_threads`

Note:

Resource definitions are mutually exclusive. Choose only one resource definition per host.

For example:

```

Begin Host
HOSTNAME  model    type      r1m  mem  swp  RESOURCES  #Keywords
#lmon     PC200      LINUX86   3.5  1    2    (linux)
#plum     !          NTX86     3.5  1    2    (nt)
Host_name !          NTX86     -    -    -    (define_ncpus_procs)
End       Host

```

3. Save and close `lsf.cluster.cluster_name`.
4. Restart the master host.

Note:

In multi-cluster environments, if `ncpus` is defined on a per-host basis (thereby overriding the global setting) the definition is applied to *all* clusters that the host is a part of. In contrast, globally defined `ncpus` settings *only* take effect within the cluster for which `EGO_DEFINE_NCPUS` is defined.

Automatic detection of hardware reconfiguration

Some UNIX operating systems support dynamic hardware reconfiguration—that is, the attaching or detaching of system boards in a live system without having to reboot the host.

Supported platforms

LSF is able to recognize changes in `ncpus`, `maxmem`, `maxswp`, `maxtmp` in the following platforms:

- Sun Solaris 2.5+
- HP-UX 10.10+
- IBM AIX 4.0+
- SGI IRIX 6.2+

Dynamic changes in `ncpus`

LSF is able to automatically detect a change in the number of processors in systems that support dynamic hardware reconfiguration.

The local LIM checks if there is a change in the number of processors at an internal interval of 2 minutes. If it detects a change in the number of processors, the local LIM also checks `maxmem`, `maxswp`, `maxtmp`. The local LIM then sends this new information to the master LIM.

Dynamic changes in `maxmem`, `maxswp`, `maxtmp`

If you dynamically change `maxmem`, `maxswp`, or `maxtmp` without changing the number of processors, you need to restart the local LIM with the command `lsadmin limrestart` so that it can recognize the changes.

If you dynamically change the number of processors and any of `maxmem`, `maxswp`, or `maxtmp`, the change is automatically recognized by LSF. When it detects a change in the number of processors, the local LIM also checks `maxmem`, `maxswp`, `maxtmp`.

View dynamic hardware changes

lsxxx Commands

There may be a 2 minute delay before the changes are recognized by `lsxxx` commands (for example, before `lshosts` displays the changes).

bxxx Commands

There may be at most a 2 + 10 minute delay before the changes are recognized by `bxxx` commands (for example, before `bhosts -l` displays the changes).

This is because `mbatchd` contacts the master LIM at an internal interval of 10 minutes.

Platform MultiCluster

Configuration changes from a local cluster are communicated from the master LIM to the remote cluster at an interval of $2 * \text{CACHE_INTERVAL}$. The parameter `CACHE_INTERVAL` is configured in `lsf.cluster.cluster_name` and is by default 60 seconds.

This means that for changes to be recognized in a remote cluster there is a maximum delay of 2 minutes + $2 * \text{CACHE_INTERVAL}$.

How dynamic hardware changes affect Platform LSF

LSF uses `ncpus`, `maxmem`, `maxswp`, `maxtmp` to make scheduling and load decisions.

When processors are added or removed, LSF licensing is affected because LSF licenses are based on the number of processors.

If you put a processor offline:

- Per host or per-queue load thresholds may be exceeded sooner. This is because LSF uses the number of CPUs and relative CPU speeds to calculate effective run queue length.
- The value of CPU run queue lengths (`r15s`, `r1m`, and `r15m`) increases.
- Jobs may also be suspended or not dispatched because of load thresholds.
- Per-processor job slot limit (`PJOB_LIMIT` in `lsb.queues`) may be exceeded sooner.

If you put a new processor online:

- Load thresholds may be reached later.
- The value of CPU run queue lengths (`r15s`, `r1m`, and `r15m`) is decreased.
- Jobs suspended due to load thresholds may be resumed.

Per-processor job slot limit (`PJOB_LIMIT` in `lsb.queues`) may be reached later.

Set the external static LIM

Use the external static LIM to automatically detect the operating system type and version of hosts.

1. In `lsf.shared`, uncomment the indices you want detected.
2. In `$LSF_SERVERDIR`, rename `tmp.eslim.<extension>` to `eslim.extension`.
3. Set `EGO_ESLIM_TIMEOUT` in `lsf.conf` or `ego.conf`.
4. Restart the `lim` on all hosts.

About configured resources

LSF schedules jobs based on available resources. There are many resources built into LSF, but you can also add your own resources, and then use them same way as built-in resources.

For maximum flexibility, you should characterize your resources clearly enough so that users have satisfactory choices. For example, if some of your machines are connected to both Ethernet and FDDI, while others are only connected to Ethernet, then you probably want to define a resource called `fddi` and associate the `fddi` resource with machines connected to FDDI. This way, users can specify resource `fddi` if they want their jobs to run on machines connected to FDDI.

Add new resources to your cluster

1. Log in to any host in the cluster as the LSF administrator.
2. Define new resources in the `Resource` section of `lsf.shared`. Specify at least a name and a brief description, which is displayed to a user by `lsinfo`.
3. For static Boolean resources and static or dynamic string resources, for all hosts that have the new resources, add the resource name to the `RESOURCES` column in the `Host` section of `lsf.cluster.cluster_name`.
4. For shared resources, for all hosts that have the new resources, associate the resources with the hosts (you might also have a reason to configure non-shared resources in this section).
5. Run `lsadmin reconfig` to reconfigure LIM.
6. Run `badmin mbdrestart` to restart `mbatchd`.

Configure the `lsf.shared` resource section

Define configured resources in the `Resource` section of `lsf.shared`. There is no distinction between shared and non-shared resources. When optional attributes are not specified, the resource is treated as static and Boolean.

1. Specify a name and description for the resource, using the keywords `RESOURCENAME` and `DESCRIPTION`.

Resource names are case sensitive and can be up to 39 characters in length, with the following restrictions:

- cannot begin with a number, or contain the following special characters

```
: . ( ) [ + - * / ! & | < > @ =
```

- cannot be any of the following reserved keywords:

```
cpu cpuf io logins ls idle maxmem maxswp maxtmp type model
status it mem ncpus nprocs ncores nthreads
define_ncpus_cores define_ncpus_procs define_ncpus_threads
ndisks pg r15m r15s r1m swap swp tmp ut local
```

- cannot begin with `inf` or `nan` (upper case or lower case). Use `-R "defined(infxx)"` or `-R "defined(nanxx)"` instead if required.

2. Optional. Specify optional attributes for the resource.
 - a) Set the resource type (`TYPE = Boolean | String | Numeric`). Default is Boolean.
 - b) For dynamic resources, set the update interval (`INTERVAL`, in seconds).
 - c) For numeric resources, set where a higher value indicates greater load (`INCREASING = Y`)
 - d) For numeric shared resources, set where LSF releases the resource when a job using the resource is suspended (`RELEASE = Y`)

- e) Set resources as consumable in the CONSUMABLE column.

Static and dynamic numeric resources can be specified as consumable. A non-consumable resource should not be releasable and should be usable in order, select and same sections of a resource requirement string.

Defaults for built-in indices:

- The following are consumable: r15s, r1m, r15m, ut, pg, io, ls, it, tmp, swp, mem.
- All other built-in static resources are not consumable. (For example, ncpus, ndisks, maxmem, maxswp, maxtmp, cpuf, type, model, status, rexpri, server, hname).

Defaults for external shared resources:

- All numeric resources are consumable.
- String and boolean resources are not consumable.

Note:

Non-consumable resources are ignored in rusage sections. When LSF_STRICT_RESREQ=Y in lsf.conf, LSF rejects resource requirement strings where an rusage section contains a non-consumable resource.

Begin Resource							
RESOURCENAME	TYPE	INTERVAL	INCREASING	CONSUMABLE	DESCRIPTION	# Keywords	
patchrev	Numeric	()	Y	()	(Patch revision)		
specman	Numeric	()	N	()	(Specman)		
switch	Numeric	()	Y	N	(Network Switch)		
rack	String	()	()	()	(Server room rack)		
owner	String	()	()	()	(Owner of the host)		
elimres	Numeric	10	Y	()	(elim generated index)		
End Resource							

3. Run `lsfinfo -l` to view consumable resources.

lsinfo -l switch

```
RESOURCE_NAME:  switch
DESCRIPTION:    Network Switch
TYPE            ORDER  INTERVAL  BUILTIN  DYNAMIC  RELEASE  CONSUMABLE
Numeric         Inc    0         No       No       No       No
```

lsinfo -l specman

```
RESOURCE_NAME:  specman
DESCRIPTION:    Specman
TYPE            ORDER  INTERVAL  BUILTIN  DYNAMIC  RELEASE  CONSUMABLE
Numeric         Dec    0         No       No       Yes      Yes
```

Resources required for JSDL

The following resources are pre-defined to support the submission of jobs using JSDL files.

Begin Resource				
RESOURCENAME	TYPE	INTERVAL	INCREASING	DESCRIPTION
osname	String	600	()	(OperatingSystemName)
osver	String	600	()	(OperatingSystemVersion)
cpuarch	String	600	()	(CPUArchitectureName)
cpuspeed	Numeric	60	Y	(Individual CPUSpeed)
bandwidth	Numeric	60	Y	(Individual NetworkBandwidth)
End Resource				

Configure `lsf.cluster.cluster_name` Host section

The Host section is the only required section in `lsf.cluster.cluster_name`. It lists all the hosts in the cluster and gives configuration information for each host. The Host section must precede the ResourceMap section.

1. Define the resource names as strings in the Resource section of `lsf.shared`.

List any number of resources, enclosed in parentheses and separated by blanks or tabs.

Use the RESOURCES column to associate static Boolean resources with particular hosts.

2. Optional. To define shared resources across hosts, use the ResourceMap section.

String resources cannot contain spaces. Static numeric and string resources both use following syntax:

```
resource_name=resource_value
```

- *Resource_value* must be alphanumeric.
- For dynamic numeric and string resources, use *resource_name* directly.

Note:

If resources are defined in both the resource column of the Host section and the ResourceMap section, the definition in the resource column takes effect.

Example

```
Begin Host
HOSTNAME model type server rlm mem swp RESOURCES #Keywords
hostA ! ! 1 3.5 () () (mg elimres patchrev=3 owner=user1)
hostB ! ! 1 3.5 () () (specman=5 switch=1 owner=test)
hostC ! ! 1 3.5 () () (switch=2 rack=rack2_2_3 owner=test)
hostD ! ! 1 3.5 () () (switch=1 rack=rack2_2_3 owner=test)
End Host
```

Configure `lsf.cluster.cluster_name` ResourceMap section

Resources are associated with the hosts for which they are defined in the ResourceMap section of `lsf.cluster.cluster_name`.

1. For each resource, specify the name (RESOURCENAME) and the hosts that have it (LOCATION).

Note:

If the ResourceMap section is not defined, then any dynamic resources specified in `lsf.shared` are not tied to specific hosts, but are shared across all hosts in the cluster.

- RESOURCENAME: The name of the resource, as defined in `lsf.shared`.
- LOCATION: The hosts that share the resource. For a static resource, you must define an initial value here as well. Do not define a value for a dynamic resource.

Syntax:

```
([resource_value@][host_name... | all [~host_name]... | others | default] ...)
```

- For resource_value, square brackets are not valid.
- For static resources, you must include the resource value, which indicates the quantity of the resource.

- Type square brackets around the list of hosts, as shown. You can omit the parenthesis if you only specify one set of hosts.
- The same host cannot be in more than one instance of a resource, as indicated by square brackets. All hosts within the instance share the quantity of the resource indicated by its value.
- The keyword `all` refers to all the server hosts in the cluster, collectively. Use the not operator (`~`) to exclude hosts or host groups.
- The keyword `others` refers to all hosts not otherwise listed in the instance.
- The keyword `default` refers to each host in the cluster, individually.

Most resources specified in the `ResourceMap` section are interpreted by LSF commands as shared resources, which are displayed using `lsload -s` or `lshosts -s`.

The exceptions are:

- Non-shared static resources
- Dynamic numeric resources specified using the `default` keyword. These are host-based resources and behave like the built-in load indices such as `mem` and `swp`. They are viewed using `lsload -l` or `lsload -I`.

Example

A cluster consists of hosts `host1`, `host2`, and `host3`.

```
Begin ResourceMap
RESOURCENAME  LOCATION
verilog       (5@[all ~host1 ~host2])
synopsys      (2@[host1 host2] 2@[others])
console       (1@[host1] 1@[host2] 1@[host3])
xyz           (1@[default])
End ResourceMap
```

In this example:

- 5 units of the `verilog` resource are defined on `host3` only (all hosts except `host1` and `host2`).
- 2 units of the `synopsys` resource are shared between `host1` and `host2`. 2 more units of the `synopsys` resource are defined on `host3` (shared among all the remaining hosts in the cluster).
- 1 unit of the `console` resource is defined on each host in the cluster (assigned explicitly). 1 unit of the `xyz` resource is defined on each host in the cluster (assigned with the keyword `default`).

Restriction:

For Solaris machines, the keyword `int` is reserved.

Resources required for JSDL

1. To submit jobs using JSDL files, you must uncomment the following lines:

```
RESOURCENAME  LOCATION
osname        [default]
osver         [default]
cpuarch       [default]
cpuspeed      [default]
bandwidth     [default]
```

Reserve a static shared resource

Use resource reservation to prevent over-committing static shared resources when scheduling.

1. To indicate that a shared resource is to be reserved while a job is running, specify the resource name in the `rusage` section of the resource requirement string.

Example

You configured licenses for the Verilog application as a resource called `verilog_lic`. To submit a job to run on a host when there is a license available:

```
bsub -R "select[defined(verilog_lic)] rusage[verilog_lic=1]" myjob
```

If the job can be placed, the license it uses are reserved until the job completes.

External load indices

If you have specific workload or resource requirements at your site, the LSF administrator can define *external resources*. You can use both built-in and external resources for LSF job scheduling and host selection.

External load indices report the values of dynamic external resources. A dynamic external resource is a site-specific resource with a numeric value that changes over time, such as the space available in a directory. Use the external load indices feature to make the values of dynamic external resources available to LSF, or to override the values reported for an LSF built-in load index. For detailed information about the external load indices feature, see [External Load Indices](#) on page 155.

Modify a built-in load index

An `elim` executable can be used to override the value of a built-in load index. For example, if your site stores temporary files in the `/usr/tmp` directory, you might want to monitor the amount of space available in that directory. An `elim` can report the space available in the `/usr/tmp` directory as the value for the `tmp` built-in load index. For detailed information about how to use an `elim` to override a built-in load index, see [External Load Indices](#) on page 155.

Use licensed software with LSF

Many applications have restricted access based on the number of software licenses purchased. LSF can help manage licensed software by automatically forwarding jobs to licensed hosts, or by holding jobs in batch queues until licenses are available.

Host-locked licenses

Host-locked software licenses allow users to run an unlimited number of copies of the product on each of the hosts that has a license.

Configure host-locked licenses

- Configure a Boolean resource to represent the software license, and configure your application to require the license resource.

When users run the application, LSF chooses the best host from the set of licensed hosts.

See the *LSF Configuration Reference* for information about the `lsf.task` file and instructions on configuring resource requirements for an application.

Counted host-locked licenses

Counted host-locked licenses are only available on specific licensed hosts, but also place a limit on the maximum number of copies available on the host.

You configure counted host-locked licenses by having LSF determine the number of licenses currently available. Use either of the following to count the host-locked licenses:

- External LIM (ELIM)
- A `check_license` shell script

Count using an external LIM (ELIM)

- Configure an external load index `license` giving the number of free licenses on each host.

To restrict the application to run only on hosts with available licenses, specify `license>=1` in the resource requirements for the application.

See the *LSF Configuration Reference* for information about the `lsf.task` file.

Count Using a `check_license` script

There are two ways to use a `check_license` shell script to check license availability and acquire a license if one is available:

- Configure the `check_license` script as a job-level pre-execution command when submitting the licensed job:

```
bsub -m licensed_hosts -E check_license licensed_job
```

- Configure the `check_license` script as a queue-level pre-execution command (`preexec`).

It is possible that the license becomes unavailable between the time the `check_license` script is run, and when the job is actually run. To handle this case, configure a queue so that jobs in this queue are requeued if they exit with values indicating that the license was not successfully obtained.

Network floating licenses

A network floating license allows a fixed number of machines or users to run the product at the same time, without restricting which host the software can run on. Floating licenses are cluster-wide resources rather than belonging to a specific host.

LSF can be used to manage floating licenses using the following LSF features:

- Shared resources
- Resource reservation
- Job requeuing

Using LSF to run licensed software the licenses can be kept in use 24 hours a day, 7 days a week. For expensive licenses, this increases their value to the users and increases productivity because users do not have to wait for a license to become available.

LSF jobs can make use of floating licenses when:

- All license jobs are run through LSF
- Licenses are managed outside of LSF control

All licenses used through Platform LSF

If all jobs requiring licenses are submitted through LSF, then LSF could regulate the allocation of licenses to jobs and ensure that a job is not started if the required license is not available. A static resource is used to hold the total number of licenses that are available. The static resource is used by LSF as a counter which is decremented by the resource reservation mechanism each time a job requiring that resource is started.

Example

For example, suppose that there are 10 licenses for the Veri l og package shared by all hosts in the cluster. The LSF configuration files should be specified as shown below. The resource is a static value, so an ELIM is not necessary.

lsf.shared

```
Begin Resource
RESOURCENAME TYPE INTERVAL INCREASING DESCRIPTION
verilog Numeric () N (Floating licenses for Verilog)
End Resource
```

lsf.cluster.cluster_name

```
Begin ResourceMap
RESOURCENAME LOCATION
verilog (10@[all])
End ResourceMap
```

Submit jobs

The users submit jobs requiring verilog licenses as follows:

```
bsub -R "rusage[verilog=1]" myprog
```

Licenses used outside of LSF control

To handle the situation where application licenses are used by jobs outside of LSF, use an ELIM to dynamically collect the actual number of licenses available instead of relying on a statically configured value. The ELIM periodically informs LSF of the number of available licenses, and LSF takes this into consideration when scheduling jobs.

Example

Assuming there are a number of licenses for the Veri l og package that can be used by all the hosts in the cluster, the LSF configuration files could be set up to monitor this resource as follows:

lsf.shared

```
Begin Resource
RESOURCENAME TYPE INTERVAL INCREASING DESCRIPTION
verilog Numeric 60 N (Floating licenses for Verilog)
End Resource
```

lsf.cluster.cluster_name

```
Begin ResourceMap
RESOURCENAME LOCATION v
erilog ([all])
End ResourceMap
```

The INTERVAL in the lsf. shared file indicates how often the ELIM is expected to update the value of the Veri l og resource — in this case every 60 seconds. Since this resource is shared by all hosts in the cluster, the ELIM only needs to be started on the master host. If the Veri l og licenses can only be accessed

by some hosts in the cluster, specify the LOCATION field of the ResourceMap section as ([hostA hostB hostC . . .]). In this case an ELIM is only started on hostA.

Submit jobs

The users submit jobs requiring veril og licenses as follows:

```
bsub -R "rusage[verilog=1:duration=1]" myprog
```

Configure a dedicated queue for floating licenses

With a dedicated queue to run jobs requiring a floating software license, LSF reserves a software license before dispatching each job, and releases the license when the job finishes.

1. Configure a dedicated queue to run jobs requiring a floating software license.

The following example defines a queue named q_veri log in l sb. queues dedicated to jobs that require Veri l og licenses:

```
Begin Queue
QUEUE_NAME = q_verilog
RES_REQ=rusage[verilog=1: duration=1]
End Queue
```

The queue named q_veri log contains jobs that reserve one Veri l og license when started.

If the Veri l og licenses are not cluster-wide, but can only be used by some hosts in the cluster, the resource requirement string should include the defi ned() tag in the sel ect section:

```
select[defined(verilog)] rusage[verilog=1]
```

2. Run bhosts -s command to display the number of licenses being reserved by the dedicated queue.

Prevent underutilization of licenses

If a job submitted to the dedicated queue requiring a floating license does not actually use the license, then licenses can be under-utilized.

LSF assumes that each job indicating that it requires a license actually uses it, and subtracts the total number of jobs requesting specific licenses from the total number available.

- Use the durat i on keyword in the queue resource requirement specification to release the shared resource after the specified number of minutes expires.

By limiting the duration of the reservation and using the actual license usage as reported by the ELIM, underutilization is also avoided and licenses used outside of LSF can be accounted for.

When interactive jobs compete for licenses

In situations where an interactive job outside the control of LSF competes with batch jobs for a software license, it is possible that a batch job, having reserved the software license, may fail to start as its license is intercepted by an interactive job. To handle this situation,

1. Configure job requeue for each queue using REQUEUE_EXIT_VALUES in l sb. queues.

If a job exits with one of the values in REQUEUE_EXIT_VALUES, LSF requeues the job.

For example, jobs submitted to the following queue use Veri l og licenses:

```
Begin Queue
QUEUE_NAME = q_verilog
RES_REQ=rusage[verilog=1: duration=1]
# application exits with value 99 if it fails to get license
REQUEUE_EXIT_VALUES = 99
JOB_STARTER = lic_starter
End Queue
```

All jobs in the queue are started by the job starter `lic_starter`, which checks if the application failed to get a license and exits with an exit code of 99. This exit code causes the job to be requeued.

`lic_starter` job starter script

The `lic_starter` job starter can be coded as follows:

```
#!/bin/sh
# lic_starter: If application fails with no license, exit 99, # otherwise,
exit 0. The application displays
# "no license" when it fails without license available.
$* 2>&1 | grep "no license" if [ $? != "0" ] then    exit 0
# string not found, application got the license else    exit 99 fi
```

External Load Indices

External load indices report the values of dynamic external resources. A dynamic external resource is a customer-defined resource with a numeric value that changes over time, such as the space available in a directory. Use the external load indices feature to make the values of dynamic external resources available to LSF, or to override the values reported for an LSF built-in load index.

About external load indices

About external load indices

LSF bases job scheduling and host selection decisions on the resources available within your cluster. A *resource* is a characteristic of a host (such as available memory) or a cluster (such as the number of shared software licenses) that LSF uses to make job scheduling and host selection decisions.

A *static resource* has a value that does not change, such as a host's maximum swap space. A *dynamic resource* has a numeric value that changes over time, such as a host's currently available swap space. *Load indices* supply the values of dynamic resources to a host's load information manager (LIM), which periodically collects those values.

LSF has a number of built-in load indices that measure the values of dynamic, *host-based resources* (resources that exist on a single host)—for example, CPU, memory, disk space, and I/O. You can also define *shared resources* (resources that hosts in your cluster share, such as floating software licenses) and make these values available to LSF to use for job scheduling decisions.

If you have specific workload or resource requirements at your site, the LSF administrator can define *external resources*. You can use both built-in and external resources for LSF job scheduling and host selection.

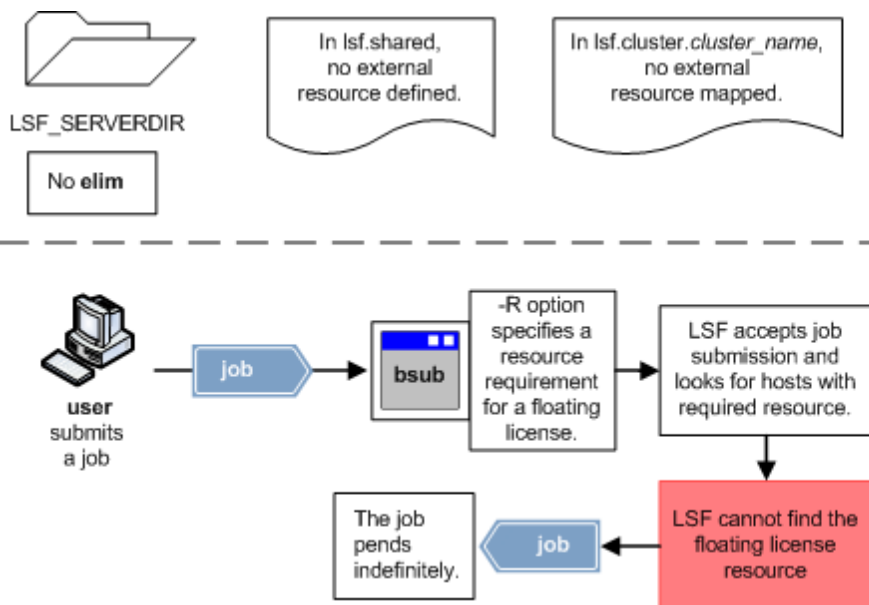
To supply the LIM with the values of dynamic external resources, either host-based or shared, the LSF administrator writes a site-specific executable called an *external load information manager* (`el i m`) executable. The LSF administrator programs the `el i m` to define external load indices, populate those indices with the values of dynamic external resources, and return the indices and their values to `std out`. An `el i m` can be as simple as a small script, or as complicated as a sophisticated C program.

Note:

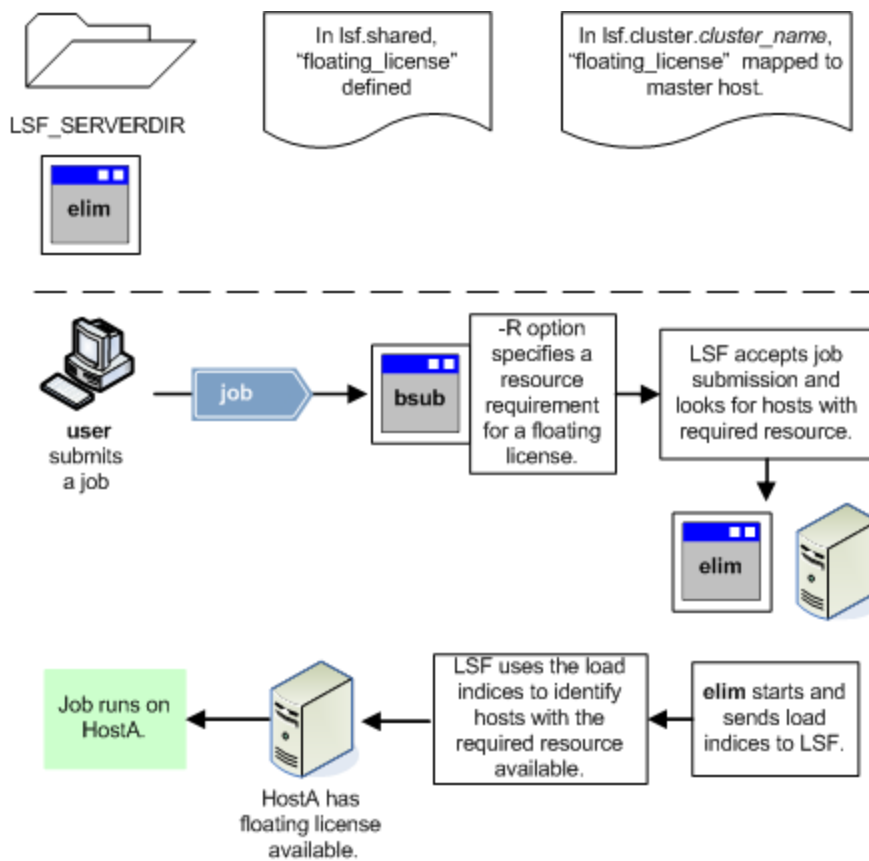
LSF does not include a default `el i m`; you should write your own executable to meet the requirements of your site.

The following illustrations show the benefits of using the external load indices feature. In these examples, jobs require the use of floating software licenses.

Default behavior (feature not enabled)



With external load indices enabled



Scope

Applicability	Details
Operating system	<ul style="list-style-type: none">• UNIX• Windows• A mix of UNIX and Windows hosts
Dependencies	<ul style="list-style-type: none">• UNIX and Windows user accounts must be valid on all hosts in the cluster and must have the correct permissions to successfully run jobs.• All <code>elim</code> executables run under the same user account as the load information manager (LIM)—by default, the LSF administrator (<code>lsfadmin</code>) account.• External dynamic resources (host-based or shared) must be defined in <code>lsf.shared</code>.

Configuration to enable external load indices

To enable the use of external load indices, you must

- Define the dynamic external resources in `lsf.shared`. By default, these resources are host-based (local to each host) until the LSF administrator configures a resource-to-host-mapping in the ResourceMap section of `lsf.cluster.cluster_name`. The presence of the dynamic external resource in `lsf.shared` and `lsf.cluster.cluster_name` triggers LSF to start the `elim` executables.
- Map the external resources to hosts in your cluster in `lsf.cluster.cluster_name`.

Important:

You must run the command **lsadmin reconfig** followed by **badmin mbdrestart** to apply changes.

- Create one or more `elim` executables in the directory specified by the parameter `LSF_SERVERDIR`. LSF does not include a default `elim`; you should write your own executable to meet the requirements of your site. The section [Create an elim executable](#) provides guidelines for writing an `elim`.

Define a dynamic external resource

To define a dynamic external resource for which `elim` collects an external load index value, define the following parameters in the Resource section of `lsf.shared`:

Configuration file	Parameter and syntax	Description
lsf.shared	RESOURCENAME <i>resource_name</i>	<ul style="list-style-type: none"> Specifies the name of the external resource.
	TYPE Numeric	<ul style="list-style-type: none"> Specifies the type of external resource: Numeric resources have numeric values. Specify Numeric for all dynamic resources.
	INTERVAL <i>seconds</i>	<ul style="list-style-type: none"> Specifies the interval for data collection by an elim. For numeric resources, defining an interval identifies the resource as a dynamic resource with a corresponding external load index. <p>Important:</p> <p>You must specify an interval: LSF treats a numeric resource with no interval as a static resource and, therefore, does not collect load index values for that resource.</p>
	INCREASING Y N	<ul style="list-style-type: none"> Specifies whether a larger value indicates a greater load. <ul style="list-style-type: none"> Y—a larger value indicates a greater load. For example, if you define an external load index for the number of shared software licenses <i>in use</i>, the larger the value, the heavier the load. N—a larger value indicates a lighter load. For example, if you define an external load index for the number of shared software licenses <i>currently available</i>, the larger the value, the lighter the load, and the more licenses are available.
	RELEASE Y N	<ul style="list-style-type: none"> For shared resources only, specifies whether LSF releases the resource when a job that uses the resource is suspended. <ul style="list-style-type: none"> Y—Releases the resource when a job is suspended. N—Holds the resource when a job is suspended.
	DESCRIPTION <i>description</i>	<ul style="list-style-type: none"> Brief description of the resource. Enter a description that enables you to easily identify the type and purpose of the resource. The lsiinfo command and the ls_info() API call return the contents of the DESCRIPTION parameter.

Map an external resource

Once external resources are defined in `lsf.shared`, they must be mapped to hosts in the `ResourceMap` section of `lsf.cluster`. *cluster_name*.

Configuration file	Parameter and syntax	Default behavior
<code>lsf.cluster</code> <i>cluster_name</i>	<code>RESOURCENAME</code> <i>resource_name</i>	<ul style="list-style-type: none"> Specifies the name of the external resource as defined in the <code>Resource</code> section of <code>lsf.shared</code>.
	LOCATION <ul style="list-style-type: none"> <code>([all]) ([all ~host_name ...])</code> 	<ul style="list-style-type: none"> Maps the resource to the master host only; all hosts share a single instance of the dynamic external resource. To prevent specific hosts from accessing the resource, use the not operator (<code>~</code>) and specify one or more host names. All other hosts can access the resource.
	<ul style="list-style-type: none"> <code>[default]</code> 	<ul style="list-style-type: none"> Maps the resource to all hosts in the cluster; every host has an instance of the dynamic external resource. If you use the <code>default</code> keyword for any external resource, all <code>elim</code> executables in <code>LSF_SERVERDIR</code> run on all hosts in the cluster. For information about how to control which <code>elim</code> executables run on each host, see the section <code>How LSF determines which hosts should run an elim executable</code>.
	<ul style="list-style-type: none"> <code>([host_name ...]) ([host_name ...] [host_name ...])</code> 	<ul style="list-style-type: none"> Maps the resource to one or more specific hosts. To specify sets of hosts that share a dynamic external resource, enclose each set in square brackets (<code>[]</code>) and use a space to separate each host name.

Create an elim executable

You can write one or more `elim` executables. The load index names defined in your `elim` executables must be the same as the external resource names defined in the `lsf.shared` configuration file.

All `elim` executables must

- Be located in `LSF_SERVERDIR` and follow these naming conventions:

Operating system	Naming convention
UNIX	<code>LSF_SERVERDIR\elim.application</code>

Operating system	Naming convention
Windows	LSF_SERVERDIR\elim. <i>application.exe</i> or LSF_SERVERDIR\elim. <i>application.bat</i>

Restriction:

The name `elim.user` is reserved for backward compatibility. Do not use the name `elim.user` for your application-specific `elim`.

Note:

LSF invokes any `elim` that follows this naming convention,—move backup copies out of LSF_SERVERDIR or choose a name that does not follow the convention. For example, use `elim_backup` instead of `elim.backup`.

- Exit upon receipt of a SIGTERM signal from the load information manager (LIM).
- Periodically output a *load update string* to `stdout` in the format *number_indices index_name index_value [index_name index_value ...]* where

Value	Defines
<i>number_indices</i>	• The number of external load indices collected by the <code>elim</code> .
<i>index_name</i>	• The name of the external load index.
<i>index_value</i>	• The external load index value returned by your <code>elim</code> .

For example, the string

```
3 tmp2 47.5 nio 344.0 licenses 5
```

reports three indices: `tmp2`, `nio`, and `licenses`, with values 47.5, 344.0, and 5, respectively.

- The load update string must report values between `-INF` and `INF` as defined in the `lsf.h` header file.
- The `elim` should ensure that the entire load update string is written successfully to `stdout`. Program the `elim` to exit if it fails to write the load update string to `stdout`.
 - If the `elim` executable is a C program, check the return value of `printf(3s)`.
 - If the `elim` executable is a shell script, check the return code of `/bin/echo(1)`.
- If the `elim` executable is implemented as a C program, use `setbuf(3)` during initialization to send unbuffered output to `stdout`.
- Each LIM sends updated load information to the master LIM every 15 seconds; the `elim` executable should write the load update string at most once every 15 seconds. If the external load index values rarely change, program the `elim` to report the new values only when a change is detected.

If you map any external resource as default in `lsf.cluster.cluster_name`, all `elim` executables in LSF_SERVERDIR run on all hosts in the cluster. If LSF_SERVERDIR contains more than one `elim` executable, you should include a header that checks whether the `elim` is programmed to report values for the resources expected on the host. For detailed information about using a checking header, see the section *How environment variables determine elim hosts*.

Overriding built-in load indices

An `elim` executable can be used to override the value of a built-in load index. For example, if your site stores temporary files in the `/usr/tmp` directory, you might want to monitor the amount of space available in that directory. An `elim` can report the space available in the `/usr/tmp` directory as the value for the `tmp` built-in load index. However, the value reported by an `elim` must be less than the maximum size of `/usr/tmp`.

To override a built-in load index value, you must:

- Write an `elim` executable that periodically measures the value of the dynamic external resource and writes the numeric value to standard output. The external load index must correspond to a numeric, dynamic external resource as defined by `TYPE` and `INTERVAL` in `lsf.shared`.
- Configure an external resource in `lsf.shared` and map the resource in `lsf.cluster.cluster_name`, even though you are overriding a built-in load index. Use a name other than the built-in load index, for example, `mytmp` rather than `tmp`.
- Program your `elim` to output the formal name of the built-in index (for example, `rlmtls`, or `swp`), *not* the resource name alias (`cpu`, `idle`, `login`, or `swap`). For example, an `elim` that collects the value of the external resource `mytmp` reports the value as `tmp` (the built-in load index) in the load update string: `1 tmp 20`.

Setting up an ELIM to support JSDL

To support the use of Job Submission Description Language (JSDL) files at job submission, LSF collects the following load indices:

Attribute name	Attribute type	Resource name
OperatingSystemName	string	osname
OperatingSystemVersion	string	osver
CPUArchitectureName	string	cpuarch
IndividualCPUSpeed	int64	cpuspeed
IndividualNetworkBandwidth	int64	bandwidth (This is the maximum bandwidth).

The file `elim.jsdl` is automatically configured to collect these resources. To enable the use of `elim.jsdl`, uncomment the lines for these resources in the `ResourceMap` section of the file `lsf.cluster.cluster_name`.

Example of an elim executable

See the section [How environment variables determine elim hosts](#) for an example of a simple `elim` script.

You can find additional `elim` examples in the `LSF_MISC/examples` directory. The `elim.c` file is an `elim` written in C. You can modify this example to collect the external load indices required at your site.

External load indices behavior

How Platform LSF manages multiple elim executables

The LSF administrator can write one `elim` executable to collect multiple external load indices, or the LSF administrator can divide external load index collection among multiple `elim` executables. On each host, the load information manager (LIM) starts a master `elim` (MELIM), which manages all `elim` executables on the host and reports the external load index values to the LIM. Specifically, the MELIM

- Starts `elim` executables on the host. The LIM checks the ResourceMap section LOCATION settings (`default`, `all`, or `host list`) and directs the MELIM to start `elim` executables on the corresponding hosts.

Note:

If the ResourceMap section contains even one resource mapped as `default`, and if there are multiple `elim` executables in `LSF_SERVERDIR`, the MELIM starts all of the `elim` executables in `LSF_SERVERDIR` on all hosts in the cluster. Not all of the `elim` executables continue to run, however. Those that use a checking header could exit with `ELIM_ABORT_VALUE` if they are not programmed to report values for the resources listed in `LSF_RESOURCES`.

- Restarts an `elim` if the `elim` exits. To prevent system-wide problems in case of a fatal error in the `elim`, the maximum restart frequency is once every 90 seconds. The MELIM does *not* restart any `elim` that exits with `ELIM_ABORT_VALUE`.
- Collects the load information reported by the `elim` executables.
- Checks the syntax of load update strings before sending the information to the LIM.
- Merges the load reports from each `elim` and sends the merged load information to the LIM. If there is more than one value reported for a single resource, the MELIM reports the latest value.
- Logs its activities and data into the log file `LSF_LOGDIR/melim.log.host_name`
- Increases system reliability by buffering output from multiple `elim` executables; failure of one `elim` does not affect other `elim` executables running on the same host.

How Platform LSF determines which hosts should run an elim executable

LSF provides configuration options to ensure that your `elim` executables run only when they can report the resources values expected on a host. This maximizes system performance and simplifies the implementation of external load indices. To control which hosts run `elim` executables, you

- Must map external resource names to locations in `lsf.cluster.cluster_name`
- Optionally, use the environment variables `LSF_RESOURCES`, `LSF_MASTER`, and `ELIM_ABORT_VALUE` in your `elim` executables

How resource mapping determines elim hosts

The following table shows how the resource mapping defined in `lsf.cluster.cluster_name` determines the hosts on which your `elim` executables start.

If the specified LOCATION is ...	Then the elim executables start on ...
<ul style="list-style-type: none"> • <code>([all]) ([all ~host_name ...])</code> 	<ul style="list-style-type: none"> • The master host, because all hosts in the cluster (except those identified by the not operator <code>[~]</code>) share a single instance of the external resource.
<ul style="list-style-type: none"> • <code>[default]</code> 	<ul style="list-style-type: none"> • Every host in the cluster, because the <code>default</code> setting identifies the external resource as host-based. • If you use the <code>default</code> keyword for any external resource, all <code>elim</code> executables in <code>LSF_SERVERDIR</code> run on all hosts in the cluster. For information about how to program an <code>elim</code> to exit when it cannot collect information about resources on a host, see How environment variables determine elim hosts.
<ul style="list-style-type: none"> • <code>([host_name ...]) ([host_name ...] [host_name ...])</code> 	<ul style="list-style-type: none"> • On the specified hosts. • If you specify a set of hosts, the <code>elim</code> executables start on the first host in the list. For example, if the <code>LOCATION</code> in the <code>ResourceMap</code> section of <code>lsf.cluster.cluster_name</code> is <code>([hostA hostB hostC] [hostD hostE hostF])</code>: <ul style="list-style-type: none"> • LSF starts the <code>elim</code> executables on <code>hostA</code> and <code>hostD</code> to report values for the resources shared by that set of hosts. • If the host reporting the external load index values becomes unavailable, LSF starts the <code>elim</code> executables on the next available host in the list. In this example, if <code>hostA</code> becomes unavailable, LSF starts the <code>elim</code> executables on <code>hostB</code>. • If <code>hostA</code> becomes available again, LSF starts the <code>elim</code> executables on <code>hostA</code> and shuts down the <code>elim</code> executables on <code>hostB</code>.

How environment variables determine elim hosts

If you use the `default` keyword for any external resource in `lsf.cluster.cluster_name`, all `elim` executables in `LSF_SERVERDIR` run on all hosts in the cluster. You can control the hosts on which your `elim` executables run by using the environment variables `LSF_MASTER`, `LSF_RESOURCES`, and `ELIM_ABORT_VALUE`. These environment variables provide a way to ensure that `elim` executables run only when they are programmed to report the values for resources expected on a host.

- **LSF_MASTER**—You can program your `elim` to check the value of the `LSF_MASTER` environment variable. The value is `Y` on the master host and `N` on all other hosts. An `elim` executable can use this parameter to check the host on which the `elim` is currently running.
- **LSF_RESOURCES**—When the LIM starts an MELIM on a host, the LIM checks the resource mapping defined in the `ResourceMap` section of `lsf.cluster.cluster_name`. Based on the mapping location (`default`, `all`, or a host list), the LIM sets `LSF_RESOURCES` to the list of resources expected on the host.

When the location of the resource is defined as `default`, the resource is listed in `LSF_RESOURCES` on the server hosts. When the location of the resource is defined as `all`, the resource is only listed in `LSF_RESOURCES` on the master host.

Use `LSF_RESOURCES` in a checking header to verify that an `elim` is programmed to collect values for at least one of the resources listed in `LSF_RESOURCES`.

- **ELIM_ABORT_VALUE**—An `elim` should exit with `ELIM_ABORT_VALUE` if the `elim` is not programmed to collect values for at least one of the resources listed in `LSF_RESOURCES`. The MELIM does not restart an `elim` that exits with `ELIM_ABORT_VALUE`. The default value is 97.

The following sample code shows how to use a header to verify that an `elim` is programmed to collect load indices for the resources expected on the host. If the `elim` is not programmed to report on the requested resources, the `elim` does not need to run on the host.

```
#!/bin/sh
# list the resources that the elim can report to lim
my_resource="myrsc"
# do the check when SLSF_RESOURCES is defined by lim
if [ -n "SLSF_RESOURCES" ]; then
# check if the resources elim can report are listed in SLSF_RESOURCES
res_ok=`echo " SLSF_RESOURCES " | /bin/grep " $my_resource "`
# exit with $ELIM_ABORT_VALUE if the elim cannot report on at least
# one resource listed in SLSF_RESOURCES
    if [ "$res_ok" = "" ]; then
        exit $ELIM_ABORT_VALUE
    fi
fi
while [ 1 ];do
# set the value for resource "myrsc"
val="1"
# create an output string in the format:
# number_indices index1_name index1_value...
reportStr="1 $my_resource $val"
echo "$reportStr"
# wait for 30 seconds before reporting again
sleep 30
done
```

Configuration to modify external load indices

Configuration file	Parameter and syntax	Behavior
<code>lsf.cluster. cluster_name</code>	<code>ELIMARGS=cmd_line_args</code>	<ul style="list-style-type: none"> Specifies the command-line arguments required by an <code>elim</code> on startup.
Parameters section	<code>ELIM_POLL_INTERVAL=seconds</code>	<ul style="list-style-type: none"> Specifies the frequency with which the LIM samples external load index information from the MELIM.
	<code>LSF_ELIM_BLOCKTIME=seconds</code>	<ul style="list-style-type: none"> UNIX only. Specifies how long the MELIM waits before restarting an <code>elim</code> that fails to send a complete load update string. The MELIM does not restart an <code>elim</code> that exits with <code>ELIM_ABORT_VALUE</code>.
	<code>LSF_ELIM_DEBUG=y</code>	<ul style="list-style-type: none"> UNIX only. Used for debugging; logs all load information received from <code>elim</code> executables to the MELIM log file (<code>melim.log</code>, <i>host_name</i>).
	<code>LSF_ELIM_RESTARTS=integer</code>	<ul style="list-style-type: none"> UNIX only. Limits the number of times an <code>elim</code> can be restarted. You must also define either <code>LSF_ELIM_DEBUG</code> or <code>LSF_ELIM_BLOCKTIME</code>. Defining this parameter prevents an ongoing restart loop in the case of a faulty <code>elim</code>.

External load indices commands

Commands to submit workload

Command	Description
bsub -R "res_req" [-R "res_req"] ...	<ul style="list-style-type: none"> Runs the job on a host that meets the specified resource requirements. If you specify a value for a dynamic external resource in the resource requirements string, LSF uses the most recent values provided by your <code>el i m</code> executables for host selection. For example: <ul style="list-style-type: none"> Define a dynamic external resource called <code>usr_tmp</code> that represents the space available in the <code>/usr/tmp</code> directory. Write an <code>el i m</code> executable to report the value of <code>usr_tmp</code> to LSF. To run the job on hosts that have more than 15 MB available in the <code>/usr/tmp</code> directory, run the command bsub -R "usr_tmp > 15" myjob LSF uses the external load index value for <code>usr_tmp</code> to locate a host with more than 15 MB available in the <code>/usr/tmp</code> directory.

Commands to monitor

Command	Description
lsl oad	<ul style="list-style-type: none"> Displays load information for all hosts in the cluster on a per host basis.
lsl oad -R "res_req"	<ul style="list-style-type: none"> Displays load information for specific resources.

Commands to control

Command	Description
lsadmin reconfig followed by badmin mbdrestart	<ul style="list-style-type: none"> Applies changes when you modify <code>lsf . shared</code> or <code>lsf . cluster. cluster_name</code>.

Commands to display configuration

Command	Description
lsinfo	<ul style="list-style-type: none"> Displays configuration information for all resources, including the external resources defined in <code>lsf . shared</code>.
lsinfo -l	<ul style="list-style-type: none"> Displays detailed configuration information for external resources.
lsinfo resource_name ...	<ul style="list-style-type: none"> Displays configuration information for the specified resources.
bhost s -s	<ul style="list-style-type: none"> Displays information about numeric shared resources, including which hosts that share each resource.

Command	Description
<code>bhosts -s</code> <i>shared_resource_name ...</i>	<ul style="list-style-type: none">• Displays configuration information for the specified resources.

Managing Users and User Groups

View user and user group information

You can display information about LSF users and user groups using the `busers` and `bugroup` commands.

The `busers` command displays information about users and user groups. The default is to display information about the user who invokes the command. The `busers` command displays:

- Maximum number of jobs a user or group may execute on a single processor
- Maximum number of job slots a user or group may use in the cluster
- Maximum number of pending jobs a user or group may have in the system.
- Total number of job slots required by all submitted jobs of the user
- Number of job slots in the PEND, RUN, SSUSP, and USUSP states

The `bugroup` command displays information about user groups and which users belong to each group.

The `busers` and `bugroup` commands have additional options. See the `busers(1)` and `bugroup(1)` man pages for more details.

Restriction:

The keyword `all` is reserved by LSF. Ensure that no actual users are assigned the user name "all."

View user information

1. Run `busers all`.

```
busers all
USER/GROUP  JL/P  MAX  NJOBS  PEND  RUN  SSUSP  USUSP  RSV
default t   12    -    -    -    -    -    -    -
user9       1   12    34    22    10    2    0    0
groupA      -  100    20    7    11    1    1    0
```

View user pending job threshold information

1. Run `busers -w`, which displays the pending job threshold column at the end of the `busers all` output.

```
busers -w
USER/GROUP  JL/P  MAX  NJOBS  PEND  RUN  SSUSP  USUSP  RSV  MPEND
default t   12    -    -    -    -    -    -    -    10
user9       1   12    34    22    10    2    0    0    500
groupA      -  100    20    7    11    1    1    0 200000
```

View user group information

1. Run `bugroup`.

```
bugroup
GROUP_NAME  USERS
testers     user1 user2
engineers   user3 user4 user10 user9
devel op    user4 user10 user11 user34 engineers/
system      all users
```

View user share information

1. Run `bugroup -l`, which displays user share group membership information in long format.

```
bugroup -l
```



```
GROUP_NAME: testers
USERS:      user1 user2
SHARES:     [user1, 4] [others, 10]

GROUP_NAME: engineers
USERS:      user3 user4 user10 user9
SHARES:     [others, 10] [user9, 4]

GROUP_NAME: system
USERS:      all users
SHARES:     [user9, 10] [others, 15]

GROUP_NAME: develop
USERS:      user4 user10 user11 engineers/
SHARES:     [engineers, 40] [user4, 15] [user10, 34] [user11, 16]
```

View user group admin information

If user group administrators are configured in the UserGroup sections of lsb.users they appear in bugroup output.

- 1. Run `bugroup -w`, which displays the user group configuration without truncating columns.

```
bugroup -w
GROUP_NAME  USERS          GROUP_ADMIN
engineering user2 groupX groupZ  adminA[usershares]
drafting    user1 user10 user12  adminA adminB[full]
```

About user groups

User groups act as aliases for lists of users. The administrator can also limit the total number of running jobs belonging to a user or a group of users.

You can define user groups in LSF in several ways:

- Use existing user groups in the configuration files
- Create LSF-specific user groups
- Use an external executable to retrieve user group members

If desired, you can use all three methods, provided the user and group names are different.

Existing user groups as Platform LSF user groups

User groups already defined in your operating system often reflect existing organizational relationships among users. It is natural to control computer resource access using these existing groups.

You can specify existing UNIX user groups anywhere an LSF user group can be specified.

How Platform LSF recognizes UNIX user groups

Only group members listed in the `/etc/group` file or the file group. byname NIS map are accepted. The user's primary group as defined in the `/etc/passwd` file is ignored.

The first time you specify a UNIX user group, LSF automatically creates an LSF user group with that name, and the group membership is retrieved by `getgrnam(3)` on the master host at the time `mbatchd` starts. The membership of the group might be different from the one on another host. Once the LSF user group is created, the corresponding UNIX user group might change, but the membership of the LSF user group is not updated until you reconfigure LSF (`badmi n`). To specify a UNIX user group that has the same name as a user, use a slash (/) immediately after the group name: *group_name/*.

Requirements

UNIX group definitions referenced by LSF configuration files must be uniform across all hosts in the cluster. Unexpected results can occur if the UNIX group definitions are not homogeneous across machines.

How Platform LSF resolves users and user groups with the same name

If an individual user and a user group have the same name, LSF assumes that the name refers to the individual user. To specify the group name, append a slash (/) to the group name.

For example, if you have both a user and a group named `admin` on your system, LSF interprets `admin` as the name of the user, and `admin/` as the name of the group.

Where to use existing user groups

Existing user groups can be used in defining the following parameters in LSF configuration files:

- `USERS` in `l sb. queues` for authorized queue users
- `USER_NAME` in `l sb. users` for user job slot limits
- `USER_SHARES` (optional) in `l sb. hosts` for host partitions or in `l sb. queues` or `l sb. users` for queue fairshare policies

Platform LSF user groups

You can define an LSF user group within LSF or use an external executable to retrieve user group members.

User groups configured within LSF can have user group administrators configured, delegating responsibility for job control away from cluster administrators.

Use `bugroup` to view user groups and members, use `busers` to view all users in the cluster.

Where to use Platform LSF user groups

LSF user groups can be used in defining the following parameters in LSF configuration files:

- `USERS` and `ADMINISTRATORS` (optional) in `l sb. queues`
- `USER_NAME` in `l sb. users` for user job slot limits
- `USER_SHARES` (optional) in `l sb. hosts` for host partitions or in `l sb. queues` for queue fairshare policies
- `USERS` and `PER_USER` in `l sb. resources` for resource limits or resource reservation.
- `USER_GROUP` and `ACCESS_CONTROL` in `l sb. serviceclasses` for SLA access.

If you are using existing OS-level user groups instead of LSF-specific user groups, you can also specify the names of these groups in the files mentioned above.

Configure user groups

1. Log in as the LSF administrator to any host in the cluster.
2. Open `l sb. users`.
3. If the `UserGroup` section does not exist, add it:

```
Begin UserGroup
GROUP_NAME  GROUP_MEMBER      USER_SHARES
financial    (user1 user2 user3)    ([user1, 4] [others, 10])
system      (all)                ([user2, 10] [others, 15])
regular_users (user1 user2 user3 user4) -
part_time_users (!)              -
End UserGroup
```

4. Specify the group name under the `GROUP_NAME` column.

External user groups must also be defined in the `egroup` executable.

5. Specify users in the `GROUP_MEMBER` column.

For external user groups, put an exclamation mark (!) in the `GROUP_MEMBER` column to tell LSF that the group members should be retrieved using `egroup`.

Note:

If **ENFORCE_UG_TREE=Y** is defined in `l sb. params`, all user groups must conform to a tree-like structure, and a user group can appear in `GROUP_MEMBER` once at most. The second and subsequent occurrence of a user group in `GROUP_MEMBER` is ignored.

6. Optional: To enable hierarchical fairshare, specify share assignments in the `USER_SHARES` column.
7. Save your changes.
8. Run `badmi n ckconfi g` to check the new user group definition. If any errors are reported, fix the problem and check the configuration again.
9. Run `badmi n reconfi g` to reconfigure the cluster.

Configure user group administrators

By default, user group administrators can control all jobs submitted by users who are members of the user group.

Note:

Define **STRICT_UG_CONTROL=Y** in `l sb. params` to:

- Configure user group administrators for user groups with `all` as a member
- Limit user group administrators to controlling jobs in the user group when jobs are submitted with `bsub -G`.

1. Log in as the LSF administrator to any host in the cluster.
2. Open `l sb. users`.
3. Edit the UserGroup section:

```
Begin UserGroup
GROUP_NAME  GROUP_MEMBER      GROUP_ADMIN
ugAdmins    (Toby Steve)
marketing    (user1 user2)    (shelley ugAdmins)
financial    (user3 user1 ugA) (john)
engineering (all)
End UserGroup
```

4. To enable user group administrators, specify users or user groups in the `GROUP_ADMIN` column. Separate users and user groups with spaces, and enclose each `GROUP_ADMIN` entry in brackets.
5. Save your changes.
6. Run `badmin ckconfi g` to check the new user group definition. If any errors are reported, fix the problem and check the configuration again.
7. Run `badmin reconfi g` to reconfigure the cluster.

For example, for the configuration shown and the default setting **STRICT_UG_CONTROL=N** in `l sb. params`, `user1` submits a job:

```
bsub -G marketing job1.
```

`job1` can be controlled by user group administrators for both the `marketing` and `financial` user groups since `user1` is a member of both groups.

With **STRICT_UG_CONTROL=Y** defined, only the user group administrators for `marketing` can control `job1`. In addition, a user group administrator can be set for the group `engineering` which has `all` as a member.

Configure user group administrator rights

User group administrators with rights assigned can adjust user shares, adjust group membership, and create new user groups.

1. Log in as the LSF administrator to any host in the cluster.
2. Open `l sb. users`.
3. Edit the UserGroup section:

```
Begin UserGroup
GROUP_NAME  GROUP_MEMBER      GROUP_ADMIN
ugAdmins    (Toby Steve)
```

```
marketing      (user1 user2)      (shelley[full] ugAdmins)
financial      (user3 ugA)       (john ugAdmins[usershares])
End UserGroup
```

4. To enable user group administrator rights, specify users or user groups in the GROUP_ADMIN column with the rights in square brackets.
 - no rights specified: user group admins can control all jobs submitted to the user group.
 - usershares: user group admins can adjust usershares using bconf and control all jobs submitted to the user group.
 - full: user group admins can create new user groups, adjust group membership, and adjust usershares using bconf, as well as control all jobs submitted to the user group.

User group admins with full rights can only add a user group member to the user group if they also have full rights for the member user group.
5. Save your changes.
6. Run `badmin ckconfig` to check the new user group definition. If any errors are reported, fix the problem and check the configuration again.
7. Run `badmin reconfig` to reconfigure the cluster.

Import external user groups (egroup)

When the membership of a user group changes frequently, or when the group contains a large number of members, you can use an external executable called `egroup` to retrieve a list of members rather than having to configure the group membership manually. You can write a site-specific `egroup` executable that retrieves user group names and the users that belong to each group. For information about how to use the external host and user groups feature, see [External Host and User Groups](#) on page 179.

External Host and User Groups

Use the external host and user groups feature to maintain group definitions for your site in a location external to LSF, and to import the group definitions on demand.

About external host and user groups

LSF provides you with the option to configure host groups, user groups, or both. When the membership of a host or user group changes frequently, or when the group contains a large number of members, you can use an external executable called `egroup` to retrieve a list of members rather than having to configure the group membership manually. You can write a site-specific `egroup` executable that retrieves host or user group names and the hosts or users that belong to each group.

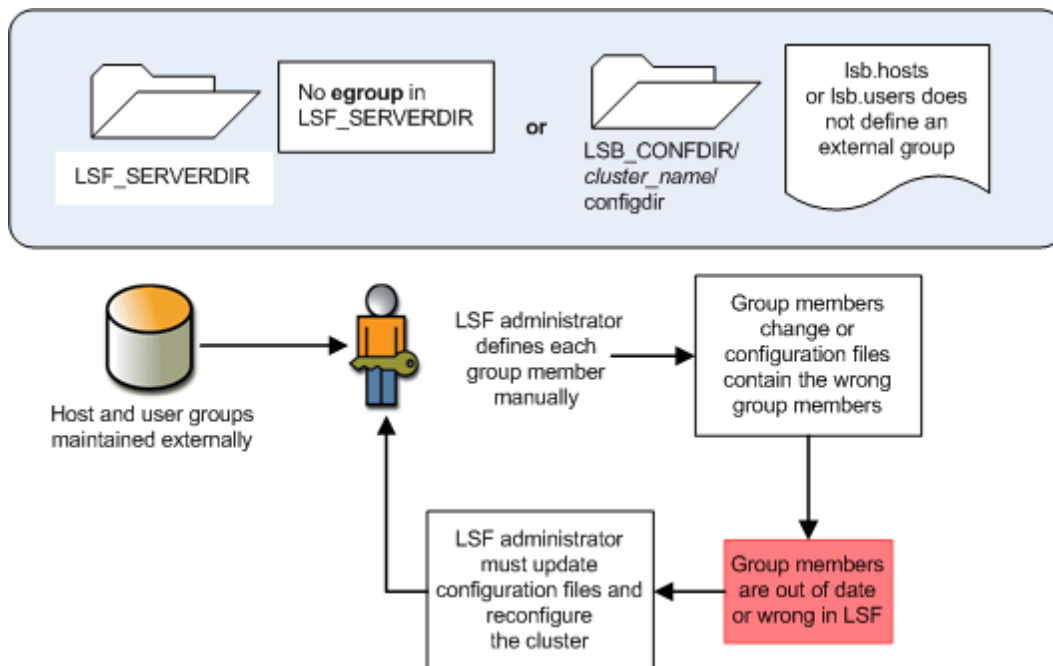
You can write your `egroup` executable to retrieve group members for:

- One or more host groups
- One or more user groups
- Any combination of host and user groups

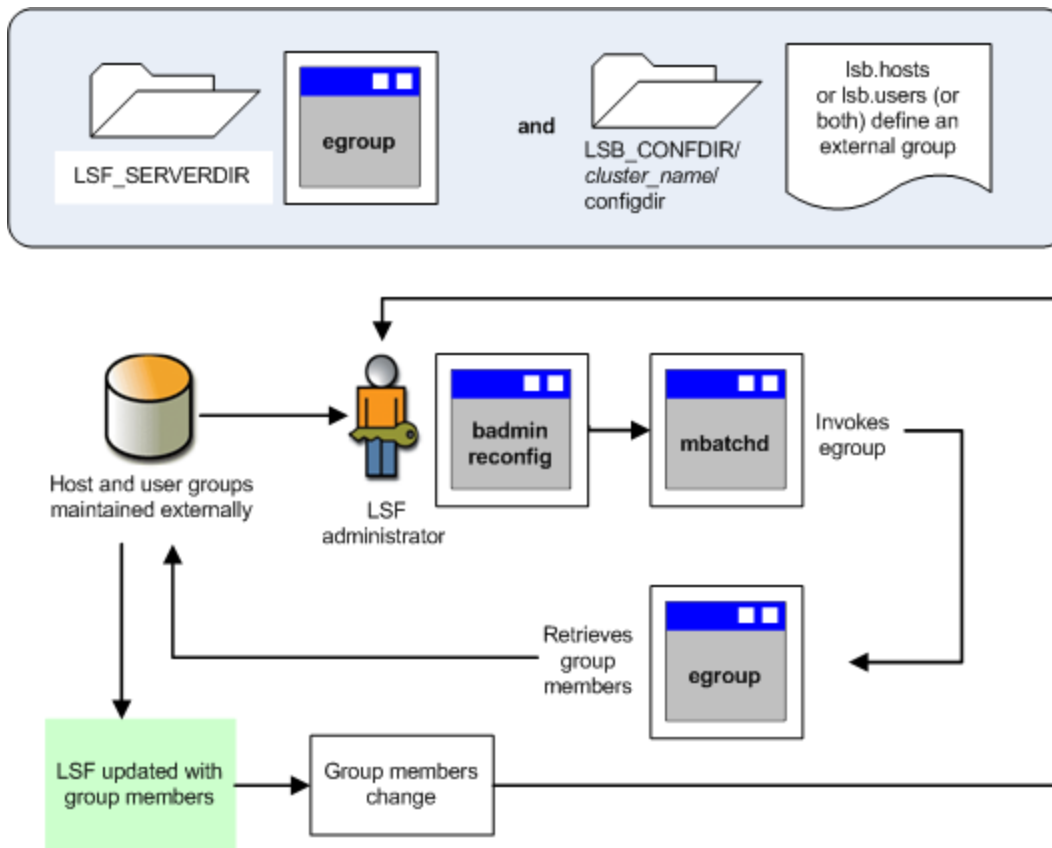
LSF does not include a default `egroup`; you should write your own executable to meet the requirements of your site.

Default behavior (feature not enabled)

The following illustrations show the benefits of using the external host and user groups feature.



With external host and user groups enabled



Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> • UNIX • Windows • A mix of UNIX and Windows hosts
Dependencies	<ul style="list-style-type: none"> • UNIX and Windows user accounts must be valid on all hosts in the cluster and must have the correct permissions to successfully run jobs. • You must reconfigure the cluster using <code>badmin reconfig</code> each time you want to run the <code>egroup</code> executable to retrieve host or user group members.
Limitations	<ul style="list-style-type: none"> • The <code>egroup</code> executable works with static hosts only; you cannot use an <code>egroup</code> executable to add a dynamically added host to a host group.
Not used with	<ul style="list-style-type: none"> • Host groups when you have configured EGO-enabled service-level agreement (SLA) scheduling, because EGO resource groups replace LSF host groups.

Configuration to enable external host and user groups

To enable the use of external host and user groups, you must

- Define the host group in `l sb. hosts`, or the user group in `l sb. users`, and put an exclamation mark (!) in the `GROUP_MEMBER` column.
- Create an egroup executable in the directory specified by the parameter `LSF_SERVERDIR` in `l sf. conf`. LSF does not include a default egroup; you should write your own executable to meet the requirements of your site.
- Run the command `badmi n reconfi g` to reconfigure the cluster and import your external host and user groups.

Define an external host or user group

External host groups are defined in `l sb. hosts`, and external user groups are defined in `l sb. users`. Your egroup executable must define the same group names that you use in the `l sb. hosts` and `l sb. users` configuration files.

Configuration file	Parameter and syntax	Default behavior
<code>l sb. hosts</code>	<code>GROUP_NAME GROUP_MEMBER</code> <i>hostgroup_name</i> (!)	<ul style="list-style-type: none">• Enables the use of an egroup executable to retrieve external host group members.• The <i>hostgroup_name</i> specified in <code>l sb. hosts</code> must correspond to the group name defined by the egroup executable.• You can configure one or more host groups to use the egroup executable.• LSF does not support the use of external host groups that contain dynamically added hosts.
<code>l sb. users</code>	<code>GROUP_NAME GROUP_MEMBER</code> <i>usergroup_name</i> (!)	<ul style="list-style-type: none">• Enables the use of an egroup executable to retrieve external user group members.• The <i>usergroup_name</i> specified in <code>l sb. users</code> must correspond to the group name defined by the egroup executable.• You can configure one or more user groups to use the egroup executable.

Create an egroup executable

The egroup executable must

- Be located in `LSF_SERVERDIR` and follow these naming conventions:

Operating system	Naming convention
UNIX	LSF_SERVERDIR\egroup
Windows	LSF_SERVERDIR\egroup. exe or LSF_SERVERDIR\egroup. bat

- Run when invoked by the commands `egroup -m hostgroup_name` and `egroup -u usergroup_name`. When `mbatchd` finds an exclamation mark (!) in the GROUP_MEMBER column of `lsb. hosts` or `lsb. users`, `mbatchd` runs the `egroup` command to invoke your `egroup` executable.
- Output a space-delimited list of group members (hosts, users, or both) to `stdout`.
- Retrieve a list of static hosts only. You cannot use the `egroup` executable to retrieve hosts that have been dynamically added to the cluster.

The following example shows a simple `egroup` script that retrieves both host and user group members:

```
#!/bin/sh
if [ "$1"="-m" ]; then
if [ "$2"="linux_grp" ]; then
    echo "linux01 linux 02 linux03 linux04"
elif [ "$2"="sol_grp" ]; then
    echo "Sol02 Sol02 Sol03 Sol04"
fi
else
if [ "$2"="srv_grp" ]; then
    echo "userA userB userC userD"
elif [ "$2"="dev_grp" ]; then
    echo "user1 user2 user3 user4"
fi
fi
```

External host and user groups behavior

On restart and reconfiguration, `mbatchd` invokes the `egroup` executable to retrieve external host and user groups and then creates the groups in memory; `mbatchd` does *not* write the groups to `lsb.hosts` or `lsb.users`. The `egroup` executable runs under the same user account as `mbatchd`. By default, this is the primary cluster administrator account.

Once LSF creates the groups in memory, the external host and user groups work the same way as any other LSF host and user groups, including configuration and batch command usage.

Between-Host User Account Mapping

The between-host user account mapping feature enables job submission and execution within a cluster that has different user accounts assigned to different hosts. Using this feature, you can map a local user account to a different user account on a remote host.

About between-host user account mapping

For clusters with different user accounts assigned to different hosts., between-host user account mapping allows you to submit a job from a local host and run the job as a different user on a remote host. There are two types of between-host user account mapping:

- Local user account mapping—for UNIX or Windows hosts, a user can map the local user account to a different user on a remote host
- Windows workgroup account mapping—allows LSF administrators to map all Windows workgroup users to a single Windows system account, eliminating the need to create multiple users and passwords in LSF. Users can submit and run jobs using their local user names and passwords, and LSF runs the jobs using the mapped system account name and password. With Windows workgroup account mapping, all users have the same permissions because all users map to the same Windows system account.

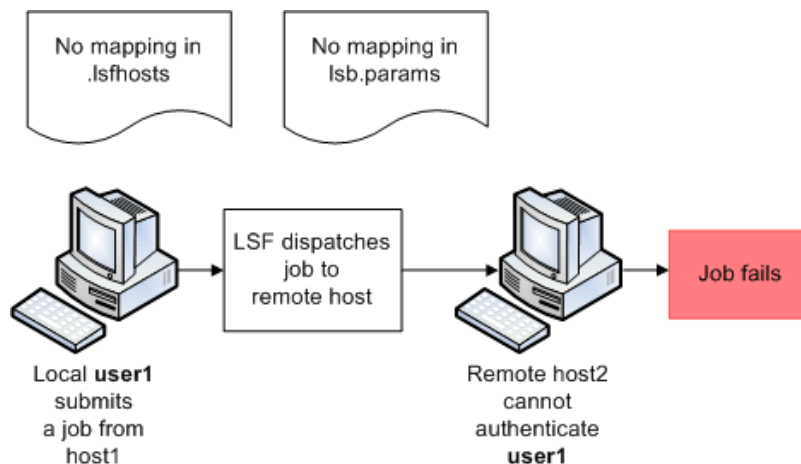


Figure 6: Default behavior (feature not enabled)

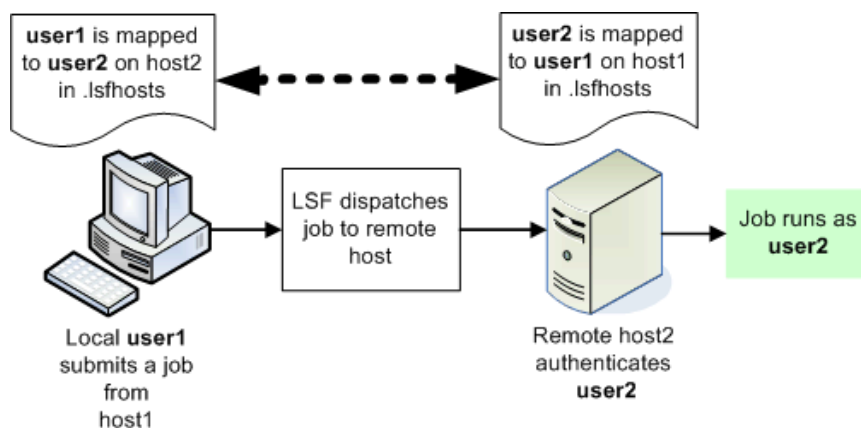


Figure 7: With local user account mapping enabled

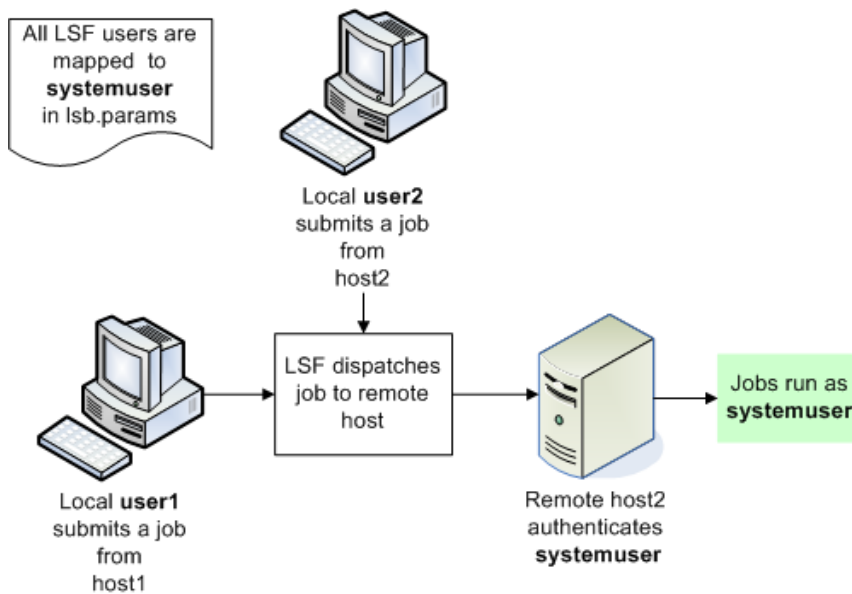


Figure 8: With Windows workgroup account mapping enabled

Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> • UNIX hosts • Windows hosts • A mix of UNIX and Windows hosts within a single clusters
Not required for	<ul style="list-style-type: none"> • A cluster with a uniform user name space • A mixed UNIX/Windows cluster in which user accounts have the same user name on both operating systems
Dependencies	<ul style="list-style-type: none"> • UNIX and Windows user accounts must be valid on all hosts in the cluster and must have the correct permissions to successfully run jobs. • For clusters that include both UNIX and Windows hosts, you must also enable the UNIX/Windows user account mapping feature.
Limitations	<ul style="list-style-type: none"> • For a MultiCluster environment that has different user accounts assigned to different hosts, you must also enable the cross-cluster user account mapping feature. Do not configure between-host user account mapping if you want to use system-level mapping in a MultiCluster environment; LSF ignores system-level mapping if mapping local user mapping is also defined in <code>.lsfhost.s</code>. • For Windows workgroup account mapping in a Windows workgroup environment, all jobs run using the permissions associated with the specified system account.

Configuration to enable between-host user account mapping

Between-host user account mapping can be configured in one of the following ways:

- Users can map their local accounts at the user level in the file `.lsfhosts`. This file must reside in the user's home directory with owner read-write permissions for UNIX and owner read-write-execute permissions for Windows. It must not be readable and writable by any other user other than the owner. Save the `.lsfhosts` file without a file extension. Both the remote and local hosts must have corresponding mappings in their respective `.lsfhosts` files.
- LSF administrators can set up Windows workgroup account mapping at the system level in `lsb.params`.

Local user account mapping configuration

Local user account mapping is enabled by adding lines to the file `.lsfhosts`. Both the remote and local hosts must have corresponding mappings in their respective `.lsfhosts` files.

Configuration file	Syntax	Behavior
<code>.lsfhosts</code>	<code>host_name user_name send</code>	• Jobs sent from the local account run as <code>user_name</code> on <code>host_name</code>
	<code>host_name user_name recv</code>	• The local account can run jobs received from <code>user_name</code> submitted on <code>host_name</code>
	<code>host_name user_name</code>	• The local account can send jobs to and receive jobs from <code>user_name</code> on <code>host_name</code>
	<code>++</code>	• The local account can send jobs to and receive jobs from any user on any LSF host

Windows workgroup account mapping

Windows workgroup account mapping is enabled by defining the parameter `SYSTEM_MAPPING_ACCOUNT` in the file `lsb.params`.

Configuration file	Parameter and syntax	Default behavior
<code>lsb.params</code>	<code>SYSTEM_MAPPING_ACCOUNT=account</code>	<ul style="list-style-type: none"> • Enables Windows workgroup account mapping • Windows local user accounts run LSF jobs using the system account name and permissions

Between-host user account mapping behavior

Local user account mapping example

The following example describes how local user account mapping works when configured in the file `.lsfhosts` in the user's home directory. Only mappings configured in `.lsfhosts` on both the local and remote hosts work.

In the following example, the cluster contains host A, host B, and host C. The account `user1` is valid on all hosts except host C, which requires a user account name of `user99`.

To allow ...	On ...	In the home directory oflsfhosts must contain the line ...
The account <code>user1</code> to run jobs on all hosts within the cluster:			
• <code>user1</code> to send jobs to <code>user99</code> on host C	host A	<code>user1</code>	<code>hostC user99 send</code>
	host B	<code>user1</code>	<code>hostC user99 send</code>
• <code>user99</code> to receive jobs from <code>user1</code> on either host A or host B	host C	<code>user99</code>	<code>hostA user1 recv</code> <code>hostB user1 recv</code>

Windows workgroup account mapping example

The following example describes how Windows workgroup account mapping works when configured in the file `lsb.params`. In this example, the cluster has a Windows workgroup environment, and only the user account `j obuser` is valid on all hosts.

To allow ...	In <code>lsb.params</code> , configure ...	Behavior
All hosts within the cluster to run jobs on any other host within the cluster:		
• Map all local users to user account <code>j obuser</code>	<code>SYSTEM_MAPPING_ACCOUNT=j obuser</code>	When any local user submits an LSF job, the job runs under the account <code>j obuser</code> , using the permissions associated with the <code>j obuser</code> account.

Between-host user account mapping commands

Commands for submission

Command	Description
<code>bsub</code>	<ul style="list-style-type: none"> Submits the job with the user name and password of the user who entered the command. The job runs on the execution host with the submission user name and password, unless you have configured between-host user account mapping. With between-host user account mapping enabled, jobs that execute on a remote host run using the account name configured at the system level for Windows workgroups, or at the user level for local user account mapping.

Commands to monitor

Command	Description
<code>bj obs -l</code>	<ul style="list-style-type: none"> Displays detailed information about jobs, including the user name of the user who submitted the job and the user name with which the job executed.
<code>bhi st -l</code>	<ul style="list-style-type: none"> Displays detailed historical information about jobs, including the user name of the user who submitted the job and the user name with which the job executed.

Commands to control

Not applicable.

Commands to display configuration

Command	Description
<code>bparams</code>	<ul style="list-style-type: none"> Displays the value of <code>SYSTEM_MAPPING_ACCOUNT</code> defined in <code>lsb.params</code>.
<code>badmi n showconf</code>	<ul style="list-style-type: none"> Displays all configured parameters and their values set in <code>lsf.conf</code> or <code>ego.conf</code> that affect <code>mbatchd</code> and <code>sbatchd</code>. Use a text editor to view other parameters in the <code>lsf.conf</code> or <code>ego.conf</code> configuration files. In a MultiCluster environment, displays the parameters of daemons on the local cluster.

Use a text editor to view the file `.lsfhosts`.

Cross-Cluster User Account Mapping

The cross-cluster user account mapping feature enables cross-cluster job submission and execution for a MultiCluster environment that has different user accounts assigned to different hosts. Using this feature, you can map user accounts in a local cluster to user accounts in one or more remote clusters.

About cross-cluster user account mapping

For MultiCluster environments that have different user accounts assigned to different hosts, cross-cluster user account mapping allows you to submit a job from a local host and run the job as a different user on a remote host.

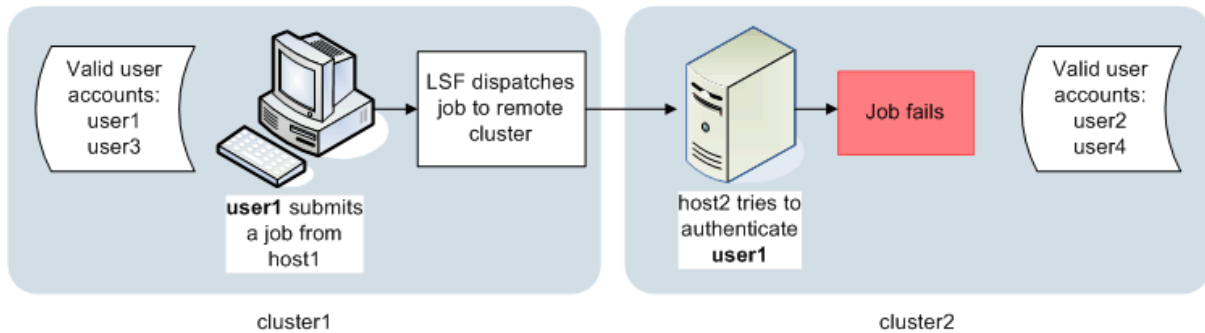


Figure 9: Default behavior (feature not enabled)

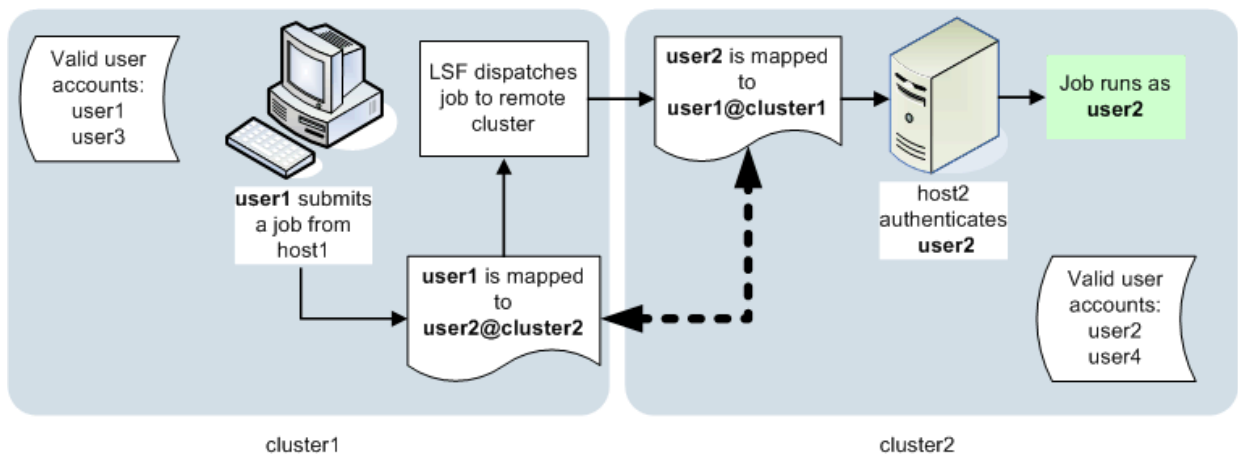


Figure 10: With cross-cluster user account mapping enabled

Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> • UNIX hosts • Windows hosts • A mix of UNIX and Windows hosts within one or more clusters
Not required for	<ul style="list-style-type: none"> • Multiple clusters with a uniform user name space

Applicability	Details
Dependencies	<ul style="list-style-type: none">• UNIX and Windows user accounts must be valid on all hosts in the cluster and must have the correct permissions to successfully run jobs.• If users at your site have different user names on UNIX and Windows hosts within a single cluster, you must configure between-host user account mapping at the user level in <code>.lsfhosts</code>.
Limitations	<ul style="list-style-type: none">• You cannot configure this feature at both the system-level and the user-level; LSF ignores system-level mapping if user-level mapping is also defined in <code>.lsfhosts</code>.• If one or more clusters include both UNIX and Windows hosts, you must also configure UNIX/Windows user account mapping.• If one or more clusters have different user accounts assigned to different hosts, you must also configure between-host user account mapping for those clusters, and then configure cross-cluster user account mapping at the system level only.

Configuration to enable cross-cluster user account mapping

- LSF administrators can map user accounts at the system level in the UserMap section of `lsb.users`. Both the remote and local clusters must have corresponding mappings in their respective `lsb.users` files.
- Users can map their local accounts at the user level in `.lsfhosts`. This file must reside in the user's home directory with owner read-write permissions for UNIX and owner read-write-execute permissions for Windows. Save the `.lsfhosts` file without a file extension. Both the remote and local hosts must have corresponding mappings in their respective `.lsfhosts` files.

Restriction:

Define *either* system-level or user-level mapping, but not both. LSF ignores system-level mapping if user-level mapping is also defined in `.lsfhosts`.

Configuration file	Level	Syntax	Behavior
<code>lsb.users</code>	System	Required fields: LOCAL REMOTE DI RECTI ON	<ul style="list-style-type: none">• Maps a user name on a local host to a different user name on a remote host• Jobs that execute on a remote host run using a mapped user name rather than the job submission user name
<code>.lsfhosts</code>	User	<code>host_name user_name send</code>	<ul style="list-style-type: none">• Jobs sent from the local account run as <i>user_name</i> on <i>host_name</i>
		<code>host_name user_name recv</code>	<ul style="list-style-type: none">• The local account can run jobs received from <i>user_name</i> submitted on <i>host_name</i>
		<code>host_name user_name</code>	<ul style="list-style-type: none">• The local account can send jobs to and receive jobs from <i>user_name</i> on <i>host_name</i>
		<code>cluster_name user_name</code>	<ul style="list-style-type: none">• The local account can send jobs to and receive jobs from <i>user_name</i> on any host in the cluster <i>cluster_name</i>
		<code>++</code>	<ul style="list-style-type: none">• The local account can send jobs to and receive jobs from any user on any LSF host

Cross-cluster user account mapping behavior

System-level configuration example

The following example illustrates LSF behavior when the LSF administrator sets up cross-cluster user account mapping at the system level. This example shows the UserMap section of the file `lsb.users` on both the local and remote clusters.

On cluster1:

```
Begin UserMap
LOCAL      REMOTE      DI RECTI ON
user1      user2@cl uster2  export
user3      user6@cl uster2  export
End UserMap
```

On cluster2:

```
Begin UserMap
LOCAL      REMOTE      DI RECTI ON
user2      user1@cl uster1  i mport
user6      user3@cl uster1  i mport
End UserMap
```

The mappings between users on different clusters are as follows:

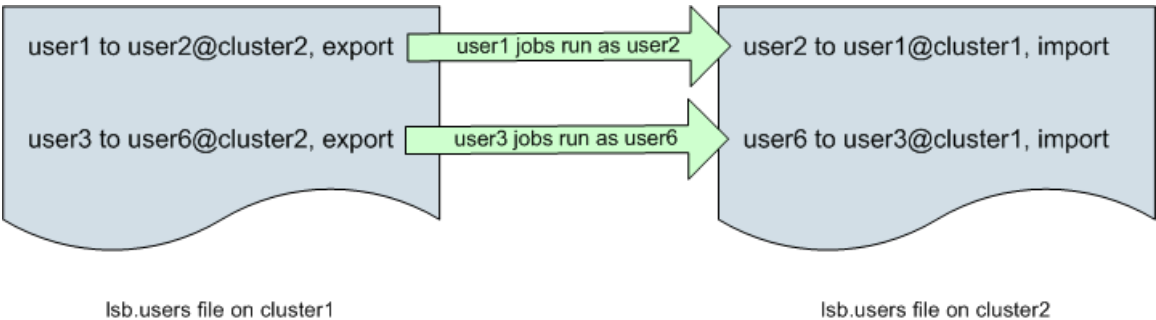


Figure 11: System-level mappings for both clusters

Only mappings configured in `lsb.users` on both clusters work. In this example, the common user account mappings are:

- `user1@cluster1` to `user2@cluster2`
- `user3@cluster1` to `user6@cluster2`

User-level configuration examples

The following examples describe how user account mapping works when configured at the user level in the file `.lsfhosts` in the user's home directory. Only mappings configured in `.lsfhosts` on hosts in both clusters work.

To allow ...	On ...	In the home directory oflsfhosts must contain the line ...
The accounts user 1 and user2 to run jobs on all hosts in both clusters:			
<ul style="list-style-type: none">• user 1 to send jobs to and receive jobs from user2 on cluster2	All hosts in cluster1	user1	cluster2 user2

To allow ...	On ...	In the home directory oflsfhosts must contain the line ...
<ul style="list-style-type: none"> user2 to send jobs to and receive jobs from user1 on cluster1 	All hosts in cluster2	user2	cluster1 user1
The account user1 to run jobs on cluster2 using the lsfguest account:			
<ul style="list-style-type: none"> user1 to send jobs as lsfguest to all hosts in cluster2 	All hosts in cluster1	user1	cluster2 lsfguest send
<ul style="list-style-type: none"> lsfguest to receive jobs from user1 on cluster1 	All hosts in cluster2	lsfguest	cluster1 user1 recv

Cross-cluster user account mapping commands

Commands for submission

Command	Description
<code>bsub</code>	<ul style="list-style-type: none"> Submits the job with the user name and password of the user who entered the command. The job runs on the execution host with the submission user name and password, unless you have configured cross-cluster user account mapping. With cross-cluster user account mapping enabled, jobs that execute on a remote host run using the account name configured at the system or user level.

Commands to monitor

Command	Description
<code>bj obs -l</code>	<ul style="list-style-type: none"> Displays detailed information about jobs, including the user name of the user who submitted the job and the user name with which the job executed.
<code>bhi st -l</code>	<ul style="list-style-type: none"> Displays detailed historical information about jobs, including the user name of the user who submitted the job and the user name with which the job executed.

UNIX/Windows User Account Mapping

The UNIX/Windows user account mapping feature enables cross-platform job submission and execution in a mixed UNIX/Windows environment. Using this feature, you can map Windows user accounts, which include a domain name, to UNIX user accounts, which do not include a domain name, for user accounts with the same user name on both operating systems.

About UNIX/Windows user account mapping

In a mixed UNIX/Windows cluster, LSF treats Windows user names (with domain) and UNIX user names (no domain) as different users. The UNIX/Windows user account mapping feature makes job submission and execution transparent across operating systems by mapping Windows accounts to UNIX accounts. With this feature enabled, LSF sends the user account name in the format required by the operating system on the execution host.

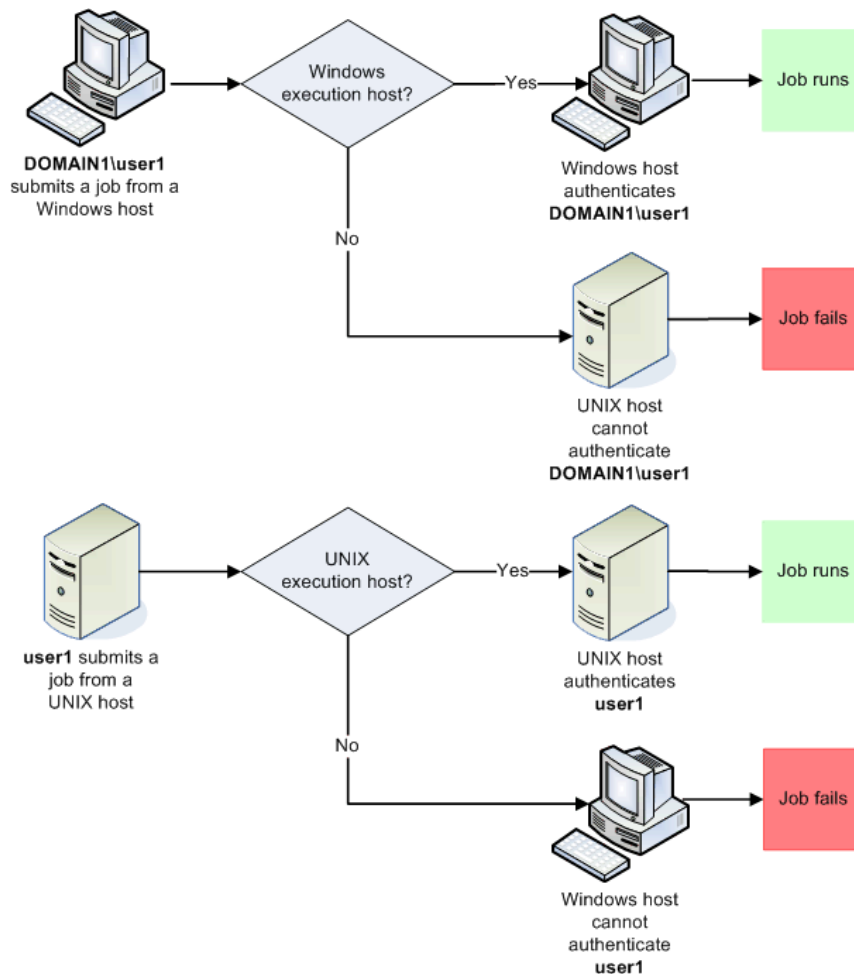


Figure 12: Default behavior (feature not enabled)

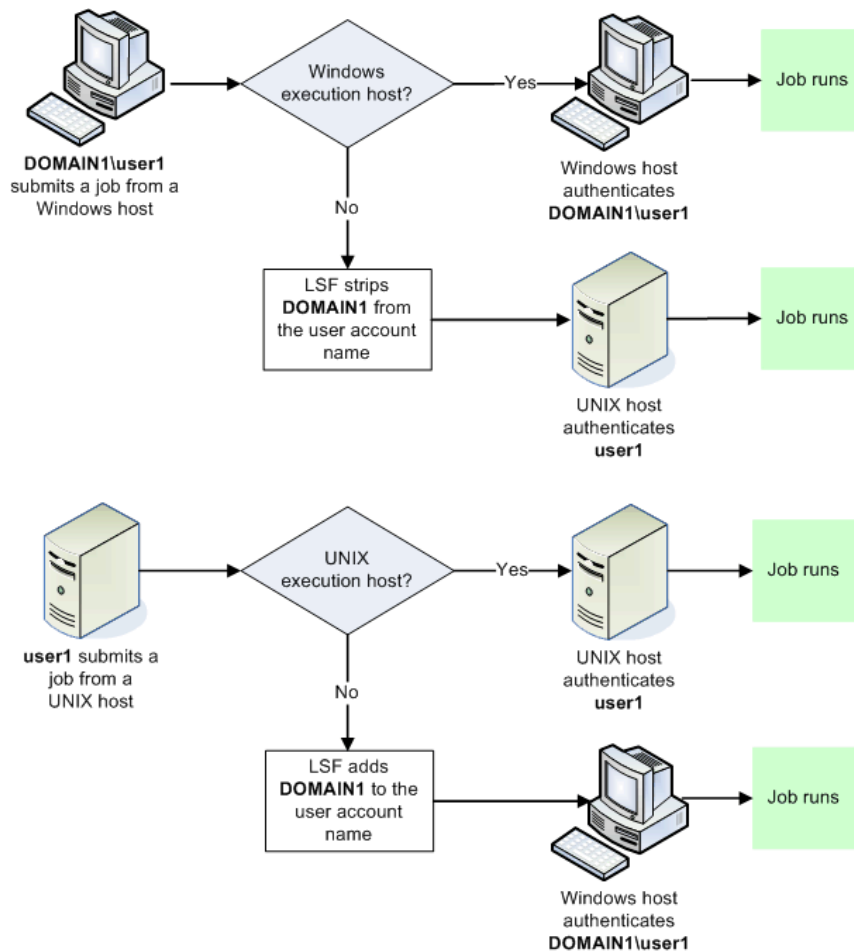


Figure 13: With UNIX/Windows user account mapping enabled

For mixed UNIX/Windows clusters, UNIX/Windows user account mapping allows you to do the following:

- Submit a job from a Windows host and run the job on a UNIX host
- Submit a job from a UNIX host and run the job on a Windows host
- Specify the domain\user combination used to run a job on a Windows host
- Schedule and track jobs submitted with either a Windows or UNIX account as though the jobs belong to a single user

LSF supports the use of both single and multiple Windows domains. In a multiple domain environment, you can choose one domain as the preferred execution domain for a particular job.

Existing Windows domain trust relationships apply in LSF. If the execution domain trusts the submission domain, the submission account is valid on the execution host.

Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> • UNIX and Windows hosts within a single cluster
Not required for	<ul style="list-style-type: none"> • Windows-only clusters • UNIX-only clusters
Dependencies	<ul style="list-style-type: none"> • UNIX and Windows user accounts must be valid on all hosts in the cluster and must have the correct permissions to successfully run jobs.
Limitations	<ul style="list-style-type: none"> • This feature works with a uniform user name space. If users at your site have different user names on UNIX and Windows hosts, you must enable between-host user account mapping. • This feature does not affect Windows workgroup installations. If you want to map all Windows workgroup users to a single Windows system account, you must configure between-host user account mapping. • This feature applies only to job execution. If you issue an LSF command or define an LSF parameter and specify a Windows user, you must use the long form of the user name, including the domain name typed in uppercase letters.

Configuration to enable UNIX/Windows user account mapping

Enable the UNIX/Windows user account mapping feature by defining one or more LSF user domains using the `LSF_USER_DOMAIN` parameter in `lsf.conf`.

Important:

Configure `LSF_USER_DOMAIN` immediately after you install LSF—changing this parameter in an existing cluster requires that you verify and possibly reconfigure service accounts, user group memberships, and user passwords.

Configuration file	Parameter and syntax	Behavior
<code>lsf.conf</code>	<code>LSF_USER_DOMAIN=domain_name</code>	<ul style="list-style-type: none"> Enables Windows domain account mapping in a single-domain environment To run jobs on a UNIX host, LSF strips the specified domain name from the user name To run jobs on a Windows host, LSF appends the domain name to the user name
	<code>LSF_USER_DOMAIN=domain_name:domain_name...</code>	<ul style="list-style-type: none"> Enables Windows domain account mapping in a multi-domain environment To run jobs on a UNIX host, LSF strips the specified domain names from the user name To run jobs on a Windows host, LSF appends the first domain name to the user name. If the first domain\user combination does not have permissions to run the job, LSF tries the next domain in the <code>LSF_USER_DOMAIN</code> list.
	<code>LSF_USER_DOMAIN=.</code>	<ul style="list-style-type: none"> Enables Windows domain account mapping To run jobs on a UNIX host, LSF strips the local machine name from the user name To run jobs on a Windows host, LSF appends the local machine name to the user name

UNIX/Windows user account mapping behavior

The following examples describe how UNIX/Windows user account mapping enables job submission and execution across a mixed UNIX/Windows cluster.

When...	In the file ...	And the job is submitted by ...	The job ...
UNIX/Windows user account mapping is not enabled	—	<ul style="list-style-type: none"> BUSINESS\user1 on a Windows host 	<ul style="list-style-type: none"> Runs on a Windows host as BUSINESS\user1 Fails on a UNIX host: BUSINESS\user1 is not a valid UNIX user name
UNIX/Windows user account mapping is not enabled	—	<ul style="list-style-type: none"> user1 on a UNIX host 	<ul style="list-style-type: none"> Fails on a Windows host: Windows requires a domain\user combination Runs on a UNIX host as user1
LSF_USER_DOMAIN=BUSINESS	lsf.conf	<ul style="list-style-type: none"> BUSINESS\user1 on a Windows host 	<ul style="list-style-type: none"> Runs on a Windows host as BUSINESS\user1 Runs on a UNIX host as user1
LSF_USER_DOMAIN=BUSINESS	lsf.conf	<ul style="list-style-type: none"> user1 on a UNIX host 	<ul style="list-style-type: none"> Runs on a Windows host as BUSINESS\user1 Runs on a UNIX host as user1
LSF_USER_DOMAIN=SUPPORT:ENGINEERING	lsf.conf	<ul style="list-style-type: none"> SUPPORT\user1 on a Windows host 	<ul style="list-style-type: none"> Runs on a Windows host as SUPPORT\user1 Runs on a UNIX host as user1
LSF_USER_DOMAIN=SUPPORT:ENGINEERING	lsf.conf	<ul style="list-style-type: none"> BUSINESS\user1 on a Windows host 	<ul style="list-style-type: none"> Runs on a Windows host as BUSINESS\user1 Fails on a UNIX host: LSF cannot strip the domain name, and BUSINESS\user1 is not a valid UNIX user name
LSF_USER_DOMAIN=SUPPORT:ENGINEERING	lsf.conf	<ul style="list-style-type: none"> user1 on a UNIX host 	<ul style="list-style-type: none"> Runs on a Windows host as SUPPORT\user1; if the job cannot run with those credentials, the job runs as ENGINEERING\user1 Runs on a UNIX host as user1

Configuration to modify UNIX/Windows user account mapping behavior

You can select a preferred execution domain for a particular job. The execution domain must be included in the LSF_USER_DOMAIN list. When you specify an execution domain, LSF ignores the order of the domains listed in LSF_USER_DOMAIN and runs the job using the specified domain. The environment variable LSF_EXECUTE_DOMAIN, defined in the user environment or from the command line, defines the preferred execution domain. Once you submit a job with an execution domain defined, you cannot change the execution domain for that particular job.

Configuration file	Parameter and syntax	Behavior
.cshrc .profile	LSF_EXECUTE_DOMAIN= <i>domain_name</i>	<ul style="list-style-type: none"> Specifies the domain that LSF uses to run jobs on a Windows host If LSF_USER_DOMAIN contains a list of multiple domains, LSF tries the LSF_EXECUTE_DOMAIN first

The following example shows the changed behavior when you define the LSF_EXECUTE_DOMAIN.

When...	In the file ...	And the job is submitted by ...	The job ...
LSF_USER_DOMAIN=SUPPORT: ENGINEERING and LSF_EXECUTE_DOMAIN=ENGINEERING	lsf.conf .profile cshrc	<ul style="list-style-type: none"> user1 on a UNIX host 	<ul style="list-style-type: none"> Runs on a Windows host as ENGINEERING\user1; if the job cannot run with those credentials, runs as SUPPORT\user1 Runs on a UNIX host as user1

These additional examples are based on the following conditions:

- In lsf.conf, LSF_USER_DOMAIN=SALES: ENGINEERING: BUSINESS
- The user has sufficient permissions to run the job in any of the LSF user domains

UNIX user1 enters ...	And LSF_EXECUTE_DOMAIN is ...	Then LSF runs the job as ...
bsub -m "hostb" myjob	Not defined in the user environment file	SALES\user1
bsub -m "hostb" myjob	Defined as BUSINESS in the user environment file	BUSINESS\user1
setenv LSF_EXECUTE_DOMAIN BUSINESS bsub -m "hostb" myjob	Either defined or not defined in the user environment file	BUSINESS\user1 The command line overrides the user environment file.

UNIX/Windows user account mapping commands

Commands for submission

Command	Description
<code>bsub</code>	<ul style="list-style-type: none"> Submits the job with the user name and password of the user who entered the command. The job runs on the execution host with the same user name and password, unless you have configured UNIX/Windows user account mapping. With UNIX/Windows user account mapping enabled, jobs that execute on a remote host run with the user account name in the format required by the operating system on the execution host.

Commands to monitor

Command	Description
<code>bj obs -w</code>	<ul style="list-style-type: none"> Displays detailed information about jobs. Displays the long form of the Windows user name including the domain name.

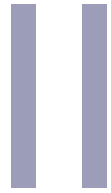
Commands to control

Command	Description
<code>l spasswd</code>	<ul style="list-style-type: none"> Registers a password for a Windows user account. Windows users must register a password for each domain\user account using this command.

Commands to display configuration

Command	Description
<code>bugroup -w</code>	<ul style="list-style-type: none"> Displays information about user groups. If UNIX/Windows user account mapping is enabled, the command <code>bugroup</code> displays user names without domains. If UNIX/Windows user account mapping is not enabled, the command <code>bugroup</code> displays user names with domains.
<code>busers</code>	<ul style="list-style-type: none"> Displays information about specific users and user groups. If UNIX/Windows user account mapping is enabled, the command <code>busers</code> displays user names without domains. If UNIX/Windows user account mapping is not enabled, the command <code>busers</code> displays user names with domains.

Command	Description
<code>badmi n showconf</code>	<ul style="list-style-type: none">Displays all configured parameters and their values set in <code>lsf.conf</code> or <code>ego.conf</code> that affect <code>mbatchd</code> and <code>sbatchd</code>. Use a text editor to view other parameters in the <code>lsf.conf</code> or <code>ego.conf</code> configuration files.In a MultiCluster environment, displays the parameters of daemons on the local cluster.



Licensing and Upgrading Platform LSF Versions

13

Platform LSF Licensing

The Platform LSF license file

You must have a valid license to run LSF.

Evaluation (demo) license

You can use a demo license to install Platform LSF and get it running temporarily, then switch to the permanent license before the evaluation period expires with no interruption in service.

Although there may be exceptions, a typical demo license:

- Is used during your free evaluation of LSF
- Expires on a preset calendar date (30 days after the license was generated)
- Is file-based (does not require Flexera® FlexNet™ software)
- Licenses all LSF products
- Allows an unlimited number of hosts to be LSF servers

Permanent license

Although there may be exceptions, a typical permanent license:

- Is granted when you purchase LSF
- Licenses specific LSF products that you have purchased
- Limits the number of hosts allowed to be LSF servers
- Requires FlexNet™ 10.8.5 or later
- Is keyed to one or more specific FlexNet license server hosts
- Does not expire

Determine how many licenses a host needs

1. Run `lim -t` to see the license requirements for a host.

```
Host Type      : NTX64
Host Architecture : EM64T_1596
Physical Processors : 1
Cores per Processor : 2
Threads per Core : 1
License Needed : 2 core(s)
```

```
Matched Type      : NTX64
Matched Architecture : EM64T_3000
Matched Model      : Intel_EM64T
CPU Factor         : 60.0
```

Format of the demo license file

- LSF licenses are stored in a text file. The default name of the license file is `license.dat`.
- The `license.dat` file for an LSF license normally contains the same products defined in `lsf.cluster.cluster_name`.
- The `license.dat` file for a demo license contains a FEATURE line for each LSF product. Each feature contains an expiry date and ends with the string `DEMO`.
- The FEATURE line contains an encrypted key to prevent tampering.
- A demo license does not require a server daemon or vendor daemon, so it does not contain `SERVER` or `DAEMON` lines, only `FEATURE` lines.

```

Product      Version      Expiry date      DEMO license
-----
FEATURE lsf_base lsf_id 8.000 24-Jan-2011 0 9C4CF8EDE0ML096AAF?? "Platform" DEMO
FEATURE lsf_manager lsf_id 8.000 24-Jan-2011 0 BC0CE84D6165BFD64E4A "Platform" DEMO
FEATURE lsf_make lsf_id 8.000 24-Jan-2011 0 8CCC08FDF035t75CC878 "Platform" DEMO
  
```

LSF vendor daemon

Number of licenses
(0 indicates an
unlimited number of
licenses)

Encrypted license key

Format of the permanent license file

A permanent license file has the same format as other products licensed with FlexNet. If you are already familiar with FlexNet license files, you can skip this section.

In addition to the information presented in the demo license file, the permanent license file includes the following:

- A **SERVER** line for each FlexNet server host. The **SERVER** line contains the following server information:
 - Host name
 - Hardware host ID
 - TCP port number used by the FlexNet license server daemon (l mgrd)

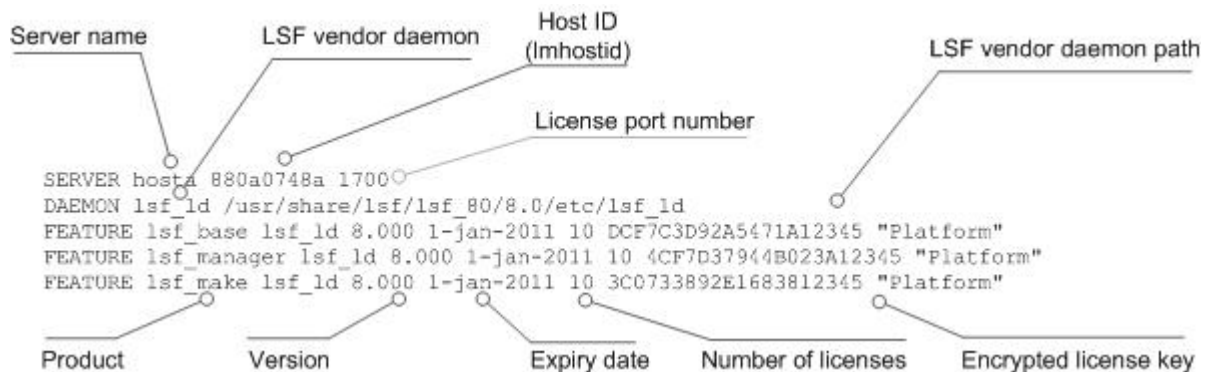
```
SERVER hostA 880a0748a 1700
```

- A **DAEMON** line for each software vendor, which gives the file path name of the LSF license vendor daemon (lsf_ld, normally installed in LSF_SERVERDIR).

```
DAEMON lsf_ld /usr/local/lsf/etc/lsf_ld
```

- Each **FEATURE** line in the license ends with **Platform**, instead of **DEMO**.

For permanent licenses, the licenses granted by the **FEATURE** line can be accessed only through license server hosts listed on the **SERVER** lines.



The license server daemon is configured to run on host a, using TCP port 1700. It allows 10 single-processor hosts to run Platform LSF

How Platform LSF permanent licensing works

Platform LSF uses the FlexNet license management product from Flexera Software to control its licenses. LSF licenses are controlled centrally through the LSF master LIM.

FlexNet license server

Permanent LSF licenses are managed by the FlexNet license server daemon (`lmgrd`). The FlexNet license server daemon runs on a license server host you choose (for failover purposes, the daemon can run on multiple hosts).

The `lmgrd` daemon starts the LSF vendor license daemon `lsf_lmd`, which periodically keeps track of how many LSF licenses are checked out and who has them. Only one `lsf_lmd` can run on a host. If `lsf_lmd` stops running, `lmgrd` immediately stops serving LSF licenses to all LSF hosts.

The LIM on the LSF master hosts contacts the license server host to get the necessary LSF licenses. It then propagates licenses to all LSF server hosts and client hosts. Multiple LSF clusters can get licenses from the same license server host.

The `TIMEOUT ALL` parameter in the FlexNet license option file changes timeout values, including how quickly the master host releases licenses during failover. LSF supports a minimum timeout value of 15 minutes.

Platform LSF license checkout

Only the master LIM can check out licenses. No other part of LSF has any contact with the FlexNet license server daemon. Once LIM on the master host identifies itself as the master, it reads the `LSF_CONFDIR/lsf.cluster.cluster_name` file to get the host information to calculate the total number of licenses needed. LSF software is licensed per core, not per host or per cluster, so hosts with multicore processors require multiple LSF licenses.

After the cluster is properly licensed, the master LIM contacts the license server daemon periodically to confirm the availability of checked out LSF licenses.

LIM distributes the licenses needed this way:

1. Calculates the total number of licenses needed for the master LIM.
2. Before slave LIMs contact the master, calculates the total number of licenses needed for all LSF server hosts and checks them out. When the slave LIMs start, they contact the master host to get the licenses they need.
3. Checks out licenses needed for client hosts listed in `LSF_CONFDIR/lsf.cluster.cluster_name`. If the license checkout fails for any host, that host is unlicensed. The master LIM tries to check out the license later.

Platform LSF license grace period

If the master LIM finds the license server daemon has gone down or is unreachable, LSF has a grace period before the whole cluster is unlicensed. As long as the master LIM that originally received the licenses is not restarted or shut down, the LSF cluster can run up to 60 hours without licenses. If you reconfigure LSF after the license server daemon becomes unavailable, you lose the grace period and the cluster is unlicensed because the original LIM that carries the correct license information is killed and restarted during reconfiguration. This prevents LSF from becoming a single point of failure and enables LSF to function reliably over an extended period of time (for example, over a long weekend) should the license server daemon fail.

Location of the Platform LSF license file for a demo license

For a demo license, each LSF host must be able to read the license file.

The installation program `lsfi nst al l` puts the LSF license file in a shared directory where it is available to all LSF hosts.

Unlicensed cluster

When LSF cannot contact a license server daemon, LSF commands are automatically resubmitted, not aborted.

Install a permanent license for the first time

If you are switching from a demo license to a permanent license, follow these instructions to set up the permanent license. You can discard the old demo license; LSF cannot use both licenses at the same time.

1. Acquire your permanent license.
2. When you receive your license file, save it as `license.dat`.
3. Edit the DAEMON line in the license file to point to the LSF vendor license daemon `lsf_ld`.

The LSF vendor license daemon is installed in `LSF_SERVERDIR` (defined in `lsf.conf` or set in your environment). For example:

```
DAEMON lsf_ld /usr/share/lsf/cluster_name/8.0/etc/lsf_ld
```

The `lsf_ld` binary should be available to the FlexNet server using this path.

4. Verify that the LSF products enabled by the PRODUCTS line in `LSF_CONFDIR/lsf.cluster.cluster_name` are licensed by features in the license file.

For example, if the PRODUCTS line contains:

```
PRODUCTS=LSF_Make LSF_MultiCluster
```

then your license must include FEATURE lines such as:

```
FEATURE lsf_make lsf_ld 8.000 1-jun-0000 10 DCF7C3D92A5471A12345 "Platform"
FEATURE lsf_multicluster lsf_ld 8.000 1-jun-0000 10 4CF7D37944B023A12345 "Platform"
```

If you do not have licenses for some products in the PRODUCTS line, contact Platform Computing or your Platform LSF vendor. To continue installing your permanent license, remove the unlicensed products from the PRODUCTS line.

5. Make sure the file is in a location where it can be accessed by the license server daemons.
6. Set the `LSF_LICENSE_FILE` parameter to point to your license file.
7. Start the license server daemon.
8. To allow the new permanent license to take effect, reconfigure the cluster:

```
lsadmin reconfig
badmin mbdrestart
```

9. After the cluster starts, use the following commands to make sure LSF is up and running:

```
lsid
bhosts
```

Getting a permanent license

To install Platform LSF for production use, you must get a permanent license from Platform or your LSF vendor.

Platform creates a permanent license that is keyed to the license server host or hosts. Some host types have a built-in hardware host ID; on others, the hardware address of the primary LAN interface is used.

For a permanent license to be created, you must supply a server host name and the hardware host identifier for each license server host at your site.

Send the following information to Platform Computing or your Platform LSF vendor.

- Host name of the license server host
- Host identifier of the license server host
- Products required
- Number of licenses required for your cluster

Acquire the FlexNet license server host identifier

When an LSF license is managed by FlexNet, you must provide a hardware host name and host identifier for the FlexNet license server host at your site.

If you do not already use FlexNet to manage other applications, you must choose a host as the FlexNet license server host before you request your license.

1. On the FlexNet server host, run `lmhostid`.

```
lmhostid - Copyright (C) 1989-2010 Flexera Software, Inc. The FlexNet host ID of
this machine is "68044d20"
```

In this example, send the code "68044d20" to Platform.

View and edit the license file

Your LSF license is a text file (normally named `license.dat`).

1. If you receive your license from Platform as text, create a new file and copy the text into the file.
2. Use any text editor such as `vi` or `emacs` to open a copy of your license file for viewing or editing.
 - You can modify lines in the license, such as the path in the `DAEMON` line when you install a new permanent license.
 - You should check that the license includes the correct features before you install it.
 - You might want to merge the contents of the LSF license into a single file that includes licenses for all applications that use FlexNet.
3. Make carriage returns visible or view the text without word wrap.

Each line in the text file must have no extra characters. You can accidentally corrupt your license file if you view it or copy it from email and then save it with hidden line breaks.

Caution:

Do not combine demo license lines with permanent license lines.

Location of the Platform LSF license file for a permanent license

For a permanent license, the FlexNet license daemon `lmgrd` and the LSF vendor daemon `lsf_l d` must be able to read the LSF license file. You can put the license file on the license server host or in a shared directory.

Daemons on the LSF master host do not need any access to the permanent license file.

LSF_LICENSE_FILE parameter

The `LSF_LICENSE_FILE` parameter in `LSF_CONFDIR/lsf.conf` points to the LSF license file.

The installation program `lsfinstall` configures the `LSF_LICENSE_FILE` parameter automatically for demo licenses only. You must set `LSF_LICENSE_FILE` manually if you do either of the following:

- Install a permanent license
- Install a DEMO or permanent license manually and change the location of the license file

To configure `LSF_LICENSE_FILE`, specify the full path name to the license file. A permanent license file should also be visible to the FlexNet license server host using the same path.

The value for `LSF_LICENSE_FILE` can be either of the following:

- The full path name to the license file.
 - UNIX example:
`LSF_LICENSE_FILE=/usr/share/lsf/cluster1/conf/license.dat`
 - Windows examples:
`LSF_LICENSE_FILE= C:\licenses\license.dat`
`LSF_LICENSE_FILE=\\HostA\licenses\license.dat`
- For a permanent license, the name of the license server host and TCP port number used by the `lmgrd` daemon, in the format `port@host_name`. For example:
`LSF_LICENSE_FILE="1700@hostD"`
- For a license with redundant servers, use a comma (,) to separate each `port@host_name` or configure the full path of the license file. For `port@host_name`, the port number must be the same as that specified in the `SERVER` line of the license file. For example:
`LSF_LICENSE_FILE="port@hostA, port@hostB, port@hostC"`
- For a license with distributed servers, use a pipe (|) to separate the `port@host_names` on UNIX, Linux and Windows. The port number must be the same as that specified in the `SERVER` line of the license file. For example:
`LSF_LICENSE_FILE="port@hostA|port@hostB|port@hostC"`

For example, after you run `lsfinstall`, the default setting is:

- If you installed LSF with a default installation, the license file is installed in the LSF configuration directory (`LSF_CONFDIR/license.dat`).
- If you installed LSF with a custom installation, the license installation directory is the one you specified. The default is the LSF configuration directory (`LSF_SERVERDIR` for the custom installation).
- If you installed FlexNet separately from LSF to manage other software licenses, the default FlexNet installation puts the license file a location you specify, usually:
 - UNIX: `/usr/share/flexm/licenses/license.dat`
 - Windows: `C:\flexm\license.dat`

`LSF_LICENSE_FILE` can also be the name of the license server host and the port number used by `lmgrd` in the form `port_number@host_name`. For example, if your license file contains the line:

```
SERVER hosta 68044d20 1700
```

`LSF_LICENSE_FILE` would be:

```
LSF_LICENSE_FILE="1700@hosta"
```

Troubleshooting

If this parameter points to an older or incorrect license key, correct the problem using one of these two methods:

- Change the path to point to the location of the new key.
- Put the new key in the location specified by the path (make a backup copy of your old license key before you overwrite it).

Licensing Platform LSF products and features

All LSF software requires a license. Some LSF features are enabled by the license file alone, but other products must also be included in the cluster configuration file. However, if you already have the FEATURE line in your license file, you can install or enable the corresponding products later on.

The following strings are examples of what can be listed in the PRODUCTS line in the Parameters section of the `lsf.cluster.cluster_name` file, to indicate which LSF products that the cluster should run. This is not a comprehensive list of Platform product license names. Any valid Platform LSF license product name can be on the PRODUCTS line, according to which Platform products you've purchased:

- LSF_Base
- LSF_Manager
- LSF_MultiCluster

If these products are listed in the cluster configuration, the LSF license must also include FEATURE lines for these products.

In addition, there are some “extra” licensed features that do not have a matching item in the PRODUCTS line. Do not remove features from your license unless instructed to do so by Platform. For example, the following string is valid in the license file, but should not be used in the PRODUCTS line:

LSF_Client

LSF client hosts are licensed per host, not per core, so there is no difference between licensing a single-processor host and a multi-processor host.

FlexNet license server host

A permanent LSF license is tied to the host ID of a particular license server host and cannot be used on another host.

If you are already running FlexNet to support other software licenses, you can use the existing license server host to manage LSF also. In this case, you will add your Platform LSF license key to the existing FlexNet license file.

If you are not already using FlexNet, or prefer to administer LSF license management separately, you must choose a host to run the license daemons.

It is possible to run multiple license server hosts for failover purposes.

Selecting a license server host

The FlexNet license server daemon normally runs on one host. LSF tolerates failure of the license server daemon for up to 60 hours, as long as the master LIM is not restarted or shut down.

If you are installing a permanent license, choose a reliable host as the license server host to ensure that the LSF licenses are always available. Although the license server host can be an LSF host, it is usually a host outside of the cluster. The license daemons create very little load, so they can be run on the host that is the dedicated file server for the Platform LSF software. This permits the licenses to be available whenever the LSF software is available.

Restriction:

You should not make the license server host the same as the master host for the cluster. If you do this, and the master host goes down, the backup

master that takes over will not be able to check license tokens out from the license server daemon on the original master which has failed.

FlexNet software for the license server host

Permanent (server-based) LSF licenses work with FlexNet version 10.8.5 or later.

If your FlexNet license server host is of the same host type as one or more LSF hosts, the FlexNet software is included in the LSF distribution and automatically installed under `LSF_SERVERDIR`, which is a shared directory (so there is no requirement to copy any software to your FlexNet license server host; just include `LSF_SERVERDIR` in your `PATH` environment variable on the license server host so that you can access the files and start the daemons).

Update a license

This information does not apply when switching your demo license to a permanent license.

1. Contact Platform Computing or your Platform LSF vendor for an updated license.

Because you already have a license, you receive new lines to put into your existing file.

2. Update your license lines by `FEATURE` (upgrading LSF or installing new products) or `INCREMENT`.

a) Feature:

- Ensure you have just one `FEATURE` line for each LSF product.
- If this is the first time you have installed the product, append the `FEATURE` line to your existing license file (if you wish, you can insert it anywhere after the `SERVER` line).
- If you already have a license for the product, replace the old `FEATURE` line with the new line.
- If you want LSF 7.x and LSF 8.x clusters to share a license file, make sure your license includes the `FEATURE` line for `lsf_batch` version 7.x.

b) Increment:

Always append an `INCREMENT` line, do not overwrite or delete existing license lines in the process.

- If this is the first increment, add the `INCREMENT` line for each product after the `FEATURE` line for that product.
- If you already have an `INCREMENT` line for the product, add the second `INCREMENT` line after the first, and so on.

FlexNet basics

This section is for users installing a permanent license, as FlexNet is not used with demo licenses.

FlexNet is used by many UNIX software packages because it provides a simple and flexible method for controlling access to licensed software. A single FlexNet license server daemon can handle licenses for many software packages, even if those packages come from different vendors. This reduces the system's administration load because you do not need to install a new license manager every time you get a new package.

Start the license daemons

FlexNet uses license daemons to manage permanent licenses.

This is a procedure that describes how to start the FlexNet license daemons.

1. Log on to the license server host as LSF administrator.

Important:

Do not run `l mgrd` as root.

2. If you have an old `lsf_l d` running, run `l mdown` to kill it.

You can only have one `lsf_l d` daemon running on a host.

3. Run the `l mgrd` command in `LSF_SERVERDIR` to start the license server daemon:

```
l mgrd -c /usr/share/lsf/cluster_name/conf/license.dat -l /usr/share/lsf/cluster_name/log/  
license.log
```

The `-c` option specifies the license file (or license file list, if you have multiple license server hosts).

The `-l` option specifies the debug log path.

Note:

You should include `LSF_SERVERDIR` in your `PATH` environment variable. You may want to include the full command line in your system startup files on the license server host, so that `l mgrd` starts automatically during system reboot.

Check the license server status

If you are using a permanent LSF license, check the status of the license server daemon. This check tells if you started your license server daemon.

The `l mstat` command is in `LSF_SERVERDIR`. For example:

```
/usr/share/lsf/cluster_name/8.0/etc/l mstat
```

1. Run `l mstat -a -c LSF_LICENSE_FILE` from the FlexNet license server and also from the LSF master host.

The output of `l mstat` gives the status of:

- The license server daemon (`l mgrd`)
- The LSF vendor daemon (`lsf_l d`)
- The number of available licenses for each product in the license file

FlexNet log file

Read this to familiarize yourself with the FlexNet log file.

The FlexNet license server daemons log messages about the state of the license server hosts, and when licenses are checked in or out. This log helps to resolve problems with the license server hosts and to track license use. The log file grows over time. You can remove or rename the existing FlexNet log file at any time.

You must choose a location for the log file when you start the license daemon. If you already have FlexNet server running for other products and Platform LSF licenses are added to the existing license file, then the log messages for FlexNet should go to the same log file you set up for other products. If FlexNet is dedicated to managing LSF licenses, you can put the FlexNet log in the same directory as your other system logs, or in the `/tmp` directory.

License management utilities

FlexNet provides several utility programs for managing software licenses. These utilities and their man pages are included in the Platform LSF software distribution.

Because these utilities can be used to shut down the FlexNet license server daemon, and can prevent licensed software from running, they are installed in the `LSF_SERVERDIR` directory. For security reasons, this directory should only be accessible to LSF administrators. Set the file permissions so that only root and members of group 0 can use them.

LSF installs the following FlexNet utilities in `LSF_SERVERDIR`:

- `lmcksum`: Calculate check sums of the license key information
- `lmdown`: Shut down the FlexNet server
- `lmhostid`: Display the hardware host ID
- `lmremove`: Remove a feature from the list of checked out features
- `lmreread`: Tell the license daemons to re-read the license file
- `lmstat`: Display the status of the license server daemons and checked out licenses
- `lmver`: Display the FlexNet version information for a program or library

Multiple FlexNet license server hosts

This section applies to permanent licenses only. Read this section if you are interested in the various ways you can distribute your licenses. This is valuable if you are interested in having some form of backup in case of failure.

Although it is not necessary, you may want to understand how the FlexNet license server behaves prior to setting up your license server hosts.

If you are concerned about the reliability of your license server host, you can distribute the LSF licenses across multiple FlexNet license server hosts. If one license server host goes down, LSF will not lose all of the available licenses. There are two ways to configure multiple license server hosts:

- Multiple license files with multiple license server hosts.
- Single license file with three redundant license server hosts.

Enable multiple license server hosts

Configuring multiple license server hosts is optional. It provides a way to keep LSF running if a license server host goes down.

Distributing licenses over multiple server hosts provides a fallback, in case your license server daemons fail.

With this method, you run multiple license server daemons, each with its own license file. Each license file has a `SERVER` line keyed to the license server host it is assigned to. The cluster is partially licensed as long as any one license server daemon is running, and fully licensed when all license server daemons are running. When a license server host is unavailable, the licenses managed by that host are unavailable. You decide how many LSF licenses to put on each license server host.

1. Obtain multiple license files, with your total number of licenses divided appropriately among the license server hosts. Provide the following information for each license server host:

- Host name and FlexNet host ID
- The products and number of licenses you want to be managed by this host

2. Specify the location of all the licenses in `LSF_LICENSE_FILE`, not just one.

Use a pipe (`|`) to separate the `port@host_names` distributed license servers on UNIX, Linux and Windows. List the primary license server host first (the one you want LSF to contact first)

3. Start `l mgrd` on all license server hosts.
4. To allow the new permanent licenses to take effect, reconfigure the cluster with the commands:

```
lsadmin reconfig
```

```
badmin mbdrestart
```

Enable redundant license server hosts

Configuring multiple license server hosts is optional. It provides a way to keep LSF running if a license server host goes down. There are two ways to configure multiple license servers.

A permanent license key is tied to a particular license server host with a specific host ID. If that host is down, the license service is not available and LSF becomes unlicensed if the master LIM is shut down or restarted.

To prevent down time, you can configure three hosts as license server hosts. The license server daemon (`l mgrd`) and LSF vendor license daemon (`l sf_l d`) run on each license server host. With three redundant

server hosts, if any one host is down, the other two continue to serve licenses. If any two hosts are down, the license service stops.

1. Obtain a license file that contains three SERVER lines. You must provide the following information for each license server host:
 - Host name and FlexNet host ID
2. Specify the location of all the licenses in LSF_LICENSE_FILE, not just one. Use a comma (,) to separate each location or configure the full path of the license file. List the primary license server host first (the one you want LSF to contact first).
3. Start lmgrd on all license server hosts, not just one.
4. To allow the new permanent licenses to take effect, reconfigure the cluster with the commands:

lsadmin reconfig

badmin mbdrestart

Partial licensing

Not all hosts in the cluster need to be licensed for the same set of LSF products. Partial licensing allows you to run some LSF products on specific hosts in the cluster, instead of licensing all the hosts in the cluster.

For example, only some of your hosts might need the license for Platform Make.

Partial licensing allows you to purchase only as many licenses as you need, rather than licensing the entire cluster for products that are only needed by a few hosts. You can save money by distributing your licenses efficiently.

Compatibility

Many LSF products do not support partial licensing, only full licensing. Those products must be enabled for the entire cluster, or not at all.

Partial licensing is supported for:

- Platform Make
- Platform Session Scheduler

Configure partial licensing

When you configure partial licensing and define the licenses for each host, you also define the order in which your licenses are given out to hosts.

1. If the product keyword (LSF_Make or LSF_Session_Scheduler) is in the PRODUCTS line in LSF_CONFDIR/lsf.cluster.*cluster_name*, remove it.

This disables full licensing. Partial licensing will not take effect if full licensing is configured.

2. To enable partial licenses, edit the Host section of LSF_CONFDIR/lsf.cluster.*cluster_name* and add the product keyword in the RESOURCES column for specific hosts.

- Platform Make
 - Add LSF_Make in the RESOURCES column.
 - If you configure partial licensing for LSF_Make, the same host will automatically be configured to use an LSF_Base license also.
- Platform Session Scheduler
 - Add LSF_Session_Scheduler in the RESOURCES column.
 - If you configure partial licensing for LSF_Session_Scheduler, the same host will automatically be configured to use LSF_Base and LSF_Manager licenses also.

When the LSF cluster starts, the master LIM reads the lsf.cluster.*cluster_name* file and determines the LSF products that each host is licensed to use.

For a permanent license, the license manager retrieves the appropriate licenses for the cluster, and distributes the licenses to the hosts in the order they are listed in lsf.cluster.*cluster_name*.

Display licensed products

Use the `lshosts -l` command to view what products are licensed for any host in the cluster.

In this example, hostA is licensed for LSF_Base and LSF_Manager, which means the host can run Platform LSF only.

lshosts -l hostA

```
HOST_NAME: hostA
type      model  cpuf  ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86   PC6000 116.1 2      1      2016M 1983M 72917M 0      Yes    1      1      2
```

RESOURCES: Not defined

RUN_WINDOWS: (always open)

Licenses enabled: (LSF_Base LSF_Manager)

LOAD_THRESHOLDS:

```
 r15s r1m r15m ut pg io ls it tmp swp mem tmp2 nio console
-      3.5 - - - - - - - - - - - - 0.0
```


Floating clients

In LSF, you can have both static client hosts and floating client hosts. LSF floating client hosts are hosts that are not all active at the same time.

For floating clients, an `lsf_client` license is shared among several client hosts at different times as it can also be assigned dynamically to any host that submits a request to LSF.

Licensing for client hosts

If you purchased an `lsf_client` license, the static client hosts must be listed in `lsf.cluster.cluster_name`. The license is fixed to the hosts specified in `lsf.cluster.cluster_name` and whenever client hosts change, you must update it with the new host list.

LSF floating client hosts are dynamic. They are not listed in `lsf.cluster.cluster_name` and since LSF does not take into account their host names, they can change dynamically and licenses will be distributed to the clients that request to use LSF. When you submit a job from any unlicensed host, and if there are any static licenses free, the host will check out a license and submit your job to LSF. However, once a host checks out a static client license, it keeps that license for the rest of the day, until midnight. A host that becomes a floating client behaves like a fixed client all day, then at 12 midnight it releases the license. At that time, the host turns back into a normal, unlicensed host, and the static client license becomes available to any other host that needs it. Restarting `mlim` can also release the license for floating client hosts.

If you purchased an `lsf_client` license, the following behavior applies within each license distribution period (triggered when a new host is added into the cluster or some host status changes):

- A static client has the highest priority to get a license from the license server
- If there are still client licenses remaining on the license server, they can be used for floating clients
- A number of remaining client licenses are reserved for floating clients according to the number configured for the `FLOAT_CLIENT` parameter
- Unavailable server hosts cannot check out `lsf_client` licenses but can check out `lsf_base` licenses with a 1:1 ratio

If you did not purchase an `lsf_client` license, the following behavior applies:

- Floating clients and static clients can check out an `lsf_base` license instead of an `lsf_client` license
- If there are `lsf_base` licenses remaining after the server hosts check them out, unavailable server hosts will get the licenses first
- If there are still `lsf_base` licenses available, static clients can check out licenses
- Lastly, floating client hosts can check out remaining `lsf_base` licenses up to the number configured for the `FLOAT_CLIENT` parameter

Note:

If the license file contains an `lsf_client` license, static clients and floating clients cannot use `lsf_base` licenses.

FLOAT_CLIENT parameter

If you have floating client hosts, configure `FLOAT_CLIENTS` in the parameter section of the `lsf.cluster.cluster_name` file. This parameter represents the maximum number of floating clients that you expect to have.

Floating client hosts and host types/models

This differentiates between client hosts and floating client hosts in terms of the restrictions on host types or models.

For LSF static client hosts, you can list the host type and model in `lsf.cluster.cluster_name` and by default, restrict running applications on different host types.

For floating client hosts, host types and models are not included in the client information. By default, any job submissions made from floating client hosts are allowed dispatch to any host type or model.

In the same way as client and server hosts, you can specify a specific model or type when you submit a job from a floating client host.

For example:

```
bsub sleep
```

The command above is interpreted as:

- `-R "type==local"` on a client host
- `-R "type==any"` on a floating client host

Security issues with floating clients

If you have floating clients in your cluster, it is important that you read this section to inform yourself of the security issues. There are measures to compensate for these security issues.

With LSF static clients, when you list client hosts in `lsf.cluster.cluster_name`, there is a level of security defined since you specify the exact hosts that will be used by the LSF system. Host authentication is done in this way.

With LSF floating clients, you should be aware of the security issues:

- Hosts that are not specified in `lsf.cluster.cluster_name` can submit requests. This means any host can submit requests.
- Remote machines make it easier for users to submit commands with a fake user ID. As a result, if an authorized user uses the user ID `lsfadmin`, the user will be able to run commands that affect the entire cluster or shut it down and cause problems in the LSF system.

Configure security for LSF floating clients

To resolve these security issues, the LSF administrator can limit which client hosts submit requests in the cluster by adding a domain or a range of domains in `lsf.cluster.cluster_name` with the parameter `FLOAT_CLIENTS_ADDR_RANGE`.

FLOAT_CLIENTS_ADDR_RANGE parameter

This optional parameter specifies an IP address or range of addresses of domains from which floating client hosts can submit requests. Multiple ranges can be defined, separated by spaces. The IP address can have either a dotted quad notation (IPv4) or IP Next Generation (IPv6) format. LSF supports both formats; you do not have to map IPv4 addresses to an IPv6 format.

Note:

You must uncomment `FLOAT_CLIENTS_ADDR_RANGE` (remove the `#` symbol before the parameter) to have it take effect.

If the value of this parameter is undefined, there is no security and any host can be an LSF floating client.

If a value is defined, security is enabled. When this parameter is defined, client hosts that do not belong to the domain will be denied access. However, if there is an error in the configuration of this variable, by default, no host will be allowed to be an LSF floating client.

If a requesting host belongs to an IP address that falls in the specified range, the host will be accepted to become an LSF floating client.

Address ranges are validated at configuration time so they must conform to the required format. If any address range is not in the correct format, no host will be accepted as an LSF floating client and a error message will be logged in the LIM log.

Conventions

- IP addresses are separated by spaces, and considered "OR" alternatives.
- The * character indicates any value is allowed.
- The - character indicates an explicit range of values. For example 1-4 indicates 1,2,3,4 are allowed.
- Open ranges such as *-30, or 10-*, are allowed.
- If a range is specified with less fields than an IP address such as 10.161, it is considered as 10.161.*.*.
- This parameter is limited to 2048 characters.

Examples

```
FLOAT_CLI_ENTS_ADDR_RANGE=100
```

All IPv4 and IPv6 hosts with a domain address starting with 100 will be allowed access.

- To specify only IPv4 hosts, set the value to **100.***
- To specify only IPv6 hosts, set the value to **100:***

```
FLOAT_CLI_ENTS_ADDR_RANGE=100- 110. 34. 1- 10. 4- 56
```

All client hosts belonging to a domain with an address having the first number between 100 and 110, then 34, then a number between 1 and 10, then, a number between 4 and 56 will be allowed access. Example: 100.34.9.45, 100.34.1.4, 102.34.3.20, etc. No IPv6 hosts are allowed.

```
FLOAT_CLI_ENTS_ADDR_RANGE=100. 172. 1. 13 100. *. 30- 54 124. 24- *. 1. *- 34
```

All client hosts belonging to a domain with the address 100.172.1.13 will be allowed access. All client hosts belonging to domains starting with 100, then any number, then a range of 30 to 54 will be allowed access. All client hosts belonging to domains starting with 124, then from 24 onward, then 1, then from 0 to 34 will be allowed access. No IPv6 hosts are allowed.

```
FLOAT_CLI_ENTS_ADDR_RANGE=12. 23. 45. *
```

All client hosts belonging to domains starting with 12.23.45 are allowed. No IPv6 hosts are allowed.

```
FLOAT_CLI_ENTS_ADDR_RANGE=100. *43
```

The * character can only be used to indicate any value. In this example, an error will be inserted in the LIM log and no hosts will be accepted to become LSF floating clients. No IPv6 hosts are allowed.

```
FLOAT_CLI_ENTS_ADDR_RANGE=100. *43 100. 172. 1. 13
```

Although one correct address range is specified, because *43 is not correct format, the entire line is considered not valid. An error will be inserted in the LIM log and no hosts will be accepted to become LSF floating clients. No IPv6 hosts are allowed.

```
FLOAT_CLI_ENTS_ADDR_RANGE = 3ffe
```

All client IPv6 hosts with a domain address starting with 3ffe will be allowed access. No IPv4 hosts are allowed.

`FLOAT_CLIENTS_ADDR_RANGE = 3ffe:ffe: : 88bb: *`

Expands to 3ffe:ffe:0:0:0:0:88bb:*. All IPv6 client hosts belonging to domains starting with 3ffe:ffe::88bb:* are allowed. No IPv4 hosts are allowed.

`FLOAT_CLIENTS_ADDR_RANGE = 3ffe-4fff:ffe: : 88bb: aa-ff 12. 23. 45. *`

All IPv6 client hosts belonging to domains starting with 3ffe up to 4fff, then ffe::88bb, and ending with aa up to ff are allowed. All IPv4 client hosts belonging to domains starting with 12.23.45 are allowed.

`FLOAT_CLIENTS_ADDR_RANGE = 3ffe- *: ffe: : 88bb: *-ff`

All IPv6 client hosts belonging to domains starting with 3ffe up to ffff and ending with 0 up to ff are allowed. No IPv4 hosts are allowed.

Check that security is enabled

Take this step after you have configured security. You are shown how to check that security has been configured properly.

After you configure `FLOAT_CLIENTS_ADDR_RANGE`, check the master LIM log file on the LSF master host (`LSF_LOGDIR/l i m. l o g. master_ host_name`) to make sure this parameter is correctly set. If this parameter is not set or is wrong, this will be indicated in the log file.

Administration commands

Since LSF floating client hosts are not listed in `l s f . c l u s t e r . cluster_name`, some administration commands will not work if issued from LSF floating client hosts. Always run administration commands from server hosts.

Troubleshooting licensing

"lsadmin reconfig" gives "User permission denied" message

If you ran `lsfinstall` as a non-root user to install a multi-user cluster, the LSF administration commands `lsadmin` and `badmi` might give the error message "User permission denied".

Use the following commands to change the ownership for `lsadmin` and `badmi` to root and the file permission mode to `-rwsr-xr-x`:

```
chown root lsadmin badmi
chmod 4755 lsadmin badmi
```

Now the user ID bit for the owner is setuid. If `lsadmin` and `badmi` are in a directory shared through NFS, the directory must be shared and mounted with setuid enabled. Do not mount with the nosuid flag. If your site does not permit this, copy `lsadmin` and `badmi` to `/usr/bin` or `/bin`.

Primary cluster administrator receives email "Your cluster has experienced license overuse" message

This occurs when your cluster is using more licenses than you have purchased. LSF allows for some overuse due to the peak usage of the cluster.

See the `lsf.cluster_name.license.acct` file for details of the peak license usage of your cluster:

OK

Peak usage is less than the maximum license availability

OVERUSE

Peak usage is more than the maximum license availability

If your cluster experiences frequent license violations or overuse, contact Platform Computing or your Platform LSF vendor to get more licenses, or plan your cluster to reduce the license usage during peak periods.

lsadmin command fails with "ls_gethostinfo: Host does not have a software license"

This may occur when you have installed the new key but have an old (unlicensed) LIM running on the LSF master.

1. On the LSF master, enter the command:

```
ps -ef | grep lim
```

2. Kill the LIM, using one of the following commands:

- **kill `lim_PID`**
- **kill -9 `lim_PID`**

3. After the old LIM has died, start the new LIM on the master host using one of the following methods:

- **lsadmin limstartup**
- **LSF_SERVERDIR/lim** as root.

Platform LSF commands give "Host does not have a software license"

You may see this message after running `l si d`, `l shost s`, or other `l s*` commands.

Typical problems and their solutions:

If you experience this problem ...	Do the following:
Your demo license (not tied to FlexNet server) has expired.	Check the <code>l i cense. dat</code> file to check the expiry date. If your license has expired, contact your account manager to obtain a new demo key or a permanent license.
Your license file may be formatted incorrectly. One of the following things may be responsible: The license file may have more than one FEATURE on a line. The license file was edited in Windows and incorrect line ending characters (^M) exist in the file.	Each FEATURE must be on its own line, and should only have UNIX line breaks. On UNIX or Linux, run <code>dos2uni x</code> to remove the Windows line breaks (^M characters) from the license file. If the license key is tied to a FlexNet server, restart <code>l mgrd</code> . Restart the master LIM.
The LSF master host is unable to communicate with the FlexNet server.	Check the network communication by entering the command: <code>ping FlexNet_server</code>
License daemons (<code>l mgrd</code> and <code>l sf_ l d</code>) are not running on the FlexNet server.	Check if <code>l mgrd</code> and <code>l sf_ l d</code> are running by typing: <code>ps -ef egrep 'lmgrd lsf_ld'</code> on the FlexNet server. If not: Check the <code>l i cense. log</code> file for error messages. Start <code>l mgrd</code> . Restart the master LIM.

Platform LSF commands fail with "ls_initdebug: Unable to open file lsf.conf"

You might see this message after running `l si d`. This message indicates that the LSF commands cannot access the `l sf. conf` file or `l sf. conf` does not exist in `LSF_ENVDIR`.

Solution:

- Use `LSF_CONFDIR/csrhc. l sf` or `LSF_CONFDIR/profi le. l sf` to set up your LSF environment, or
- If you know the location of `l sf. conf`, set the `LSF_ENVDIR` environment variable to point to the directory containing the `l sf. conf` file.

lmgrd fails with message "Port already in use"

The port number defined in `LSF_LICENSE_FILE` and `l i cense. dat` is being used by another application (by default, LSF uses port number 1700).

Possible causes:

If you experience this problem ...	Do the following:
l m g r d is already running for this license	Use <code>ps -ef</code> and make sure that <code>l m g r d</code> and <code>l s f _ l d</code> are not running.
l m g r d has been stopped and the operating system has not cleared the port	Wait a few minutes for the OS to clear this port.
Another process is using the same port (this is not likely)	<p data-bbox="643 394 1354 449">If the port number is being used by another application, execute the following to change the port number used by LSF:</p> <ol data-bbox="643 470 1404 989" style="list-style-type: none"> <li data-bbox="643 470 1404 701">1. Edit <code>l i c e n s e . d a t</code> and change the port number in the line: <pre data-bbox="678 512 1114 535">SERVER FlexNet_server 3f8b6a3 1700</pre> <p data-bbox="678 554 1404 701">The fourth field on the SERVER line of <code>l i c e n s e . d a t</code> specifies the TCP port number that the FlexNet server uses. Choose an unused port number. The default port set by FlexNet is 1700. Platform LSF usually uses port numbers in the range 3879 to 3882, so the numbers from 3883 forward are good alternate choices.</p> <li data-bbox="643 701 1404 989">2. In <code>l s f . c o n f</code>: <ul data-bbox="678 751 1404 989" style="list-style-type: none"> <li data-bbox="678 751 1404 869">• If <code>LSF_LICENSE_FILE</code> is defined as follows: <code>LSF_LICENSE_FILE=port_number@FlexNet_server</code> (for example: <code>1700@hostA</code>), the port number must be changed accordingly. <li data-bbox="678 869 1404 961">• If <code>LSF_LICENSE_FILE</code> points to the license file path (for example: <code>LSF_LICENSE_FILE=/usr/local/lsf/conf/l i c e n s e . d a t</code>), no changes are required. <li data-bbox="678 961 878 989">• Restart <code>l m g r d</code>.

Cluster Version Management and Patching on UNIX and Linux

Scope

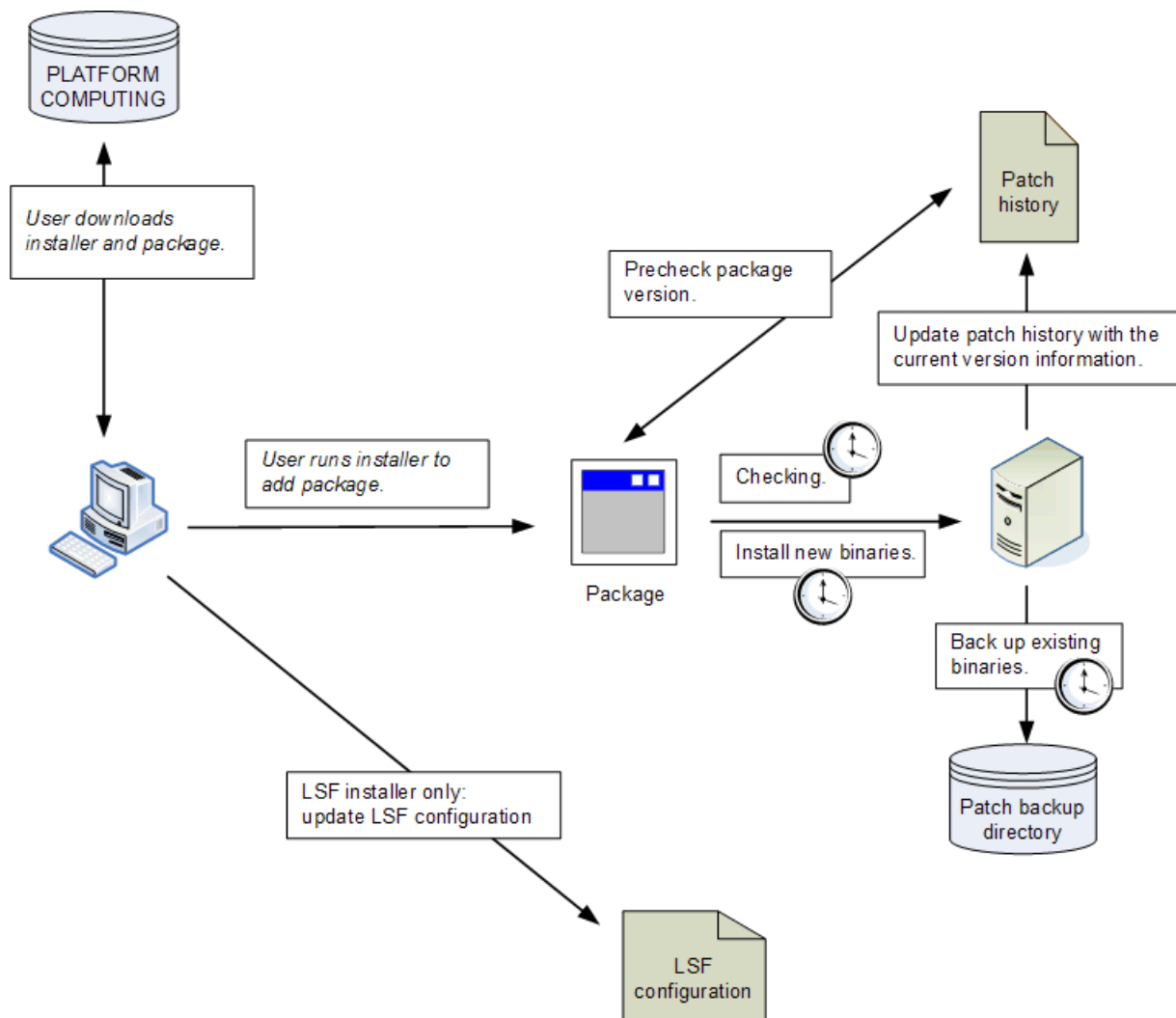
Operating system	<ul style="list-style-type: none">• Supports UNIX hosts within a single cluster
Limitations	<p>pversions supports LSF Update 1 and later</p> <p>patchinstall supports LSF Update 1 and later</p> <p>For installation of a new cluster, see <i>Installing Platform LSF on UNIX and Linux</i>.</p>

Important:

For LSF 8 Update 2 through 6, you cannot use the steps in this chapter. You must follow the steps in “Migrating to LSF Version 8 on UNIX and Linux” to manually migrate your LSF 8 cluster to LSF 8.

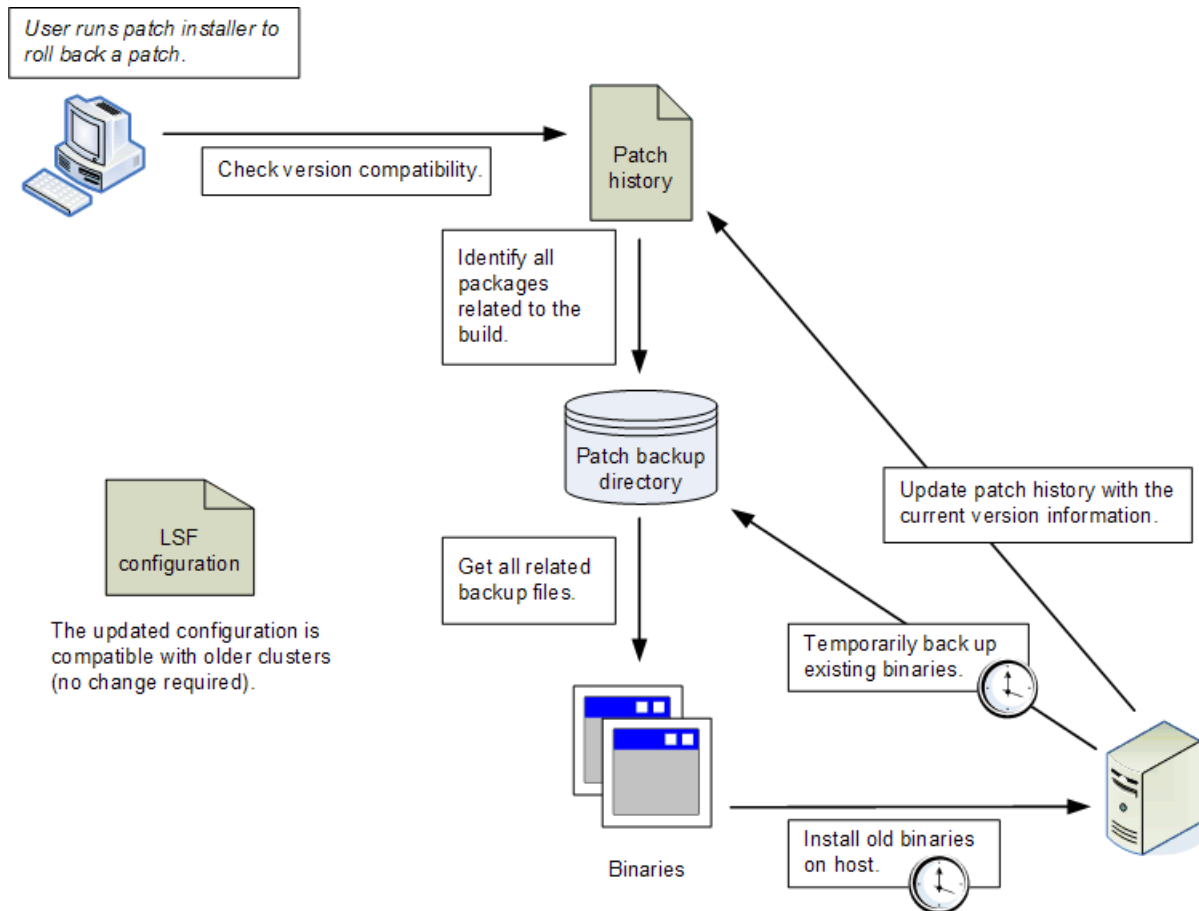
Patch installation interaction diagram

Patches may be installed using the patch installer or LSF installer. The same mechanism is used.



Patch rollback interaction diagram

Use the patch installer to roll back the most recent patch in the cluster.



Version management components

Patches and distributions

Products and versioning

Platform products and components may be separately licensed and versioned. For example, LSF and the Platform Application Center are licensed together, but delivered as separate distributions and patched separately.

Product version is a number identifying the release, such as LSF version 7.0.6. The final digit changes whenever you patch the cluster with a new update release.

In addition to the product version, build date, build number, and binary type are used to identify the distributions. Build number helps identify related distributions for different binary types and is important when rolling back the cluster.

Patching the cluster is optional and clusters with the same product version may have different patches installed, so a complete description of the cluster includes information about the patches installed.

Like installation, patching the cluster sometimes requires you to download packages for each binary type.

Types of distributions

Upgrades, patches, and hot fixes are used to update the software in an existing cluster.

- *Product upgrades* deliver a new version of the software with valuable new features. Upgrades require a new license.
- *Patches* deliver small changes and bug fixes that may result in a minor version change. Patches do not require a new license.
- *Hot fixes* deliver temporary solutions for emergency problems. Hot fixes do not require a new license.

Types of patches

This document describes installing and removing patches. Patches include fixes, fix packs, and update releases. These do not require a new license.

- *Update releases*—are full distributions available to all customers at regular intervals and include all fixes intended for general use. Your cluster should always use the latest update release. The same package can be used to patch a cluster or create a new cluster. Each update has a different version number (for example, LSF7 Update 6 is 7.0.6).
- *Fixes*—are partial distributions delivered as needed to resolve customer issues (identified by a specific fix number). Platform Support will advise you if you need install any fixes in your cluster. Installing or removing this type of patch does not change the version of the cluster.
- *Fix packs (FP)*—contain two or more related fixes in one distribution for your convenience.

Version command pversions

The version command `pversions` is a tool provided to query the patch history and deliver information about cluster and product version and patch levels.

The version command includes functionality to query a cluster or check contents of a package.

For clusters version 7.0 or earlier, the version command is not available.

For clusters version 7 Update 1 (7.0.1) or later, the command is available under `install` directory under the LSF installation directory (*LSF_TOP*/8.0/`install/pversions`). It is not located with other LSF commands and may not be in your path by default.

Command environment

Both `patchinstall` and `pversions` on UNIX need environment information to identify your cluster.

Before you run the command, set your environment using `profile.lsf` or `cshrc.lsf`. You may have already done this to administer your cluster.

As a workaround, use the `-f` option in the command line and specify a file that defines your environment. For more information, see the command reference.

Patch installer

The patch installer `patchinstall` is a tool provided to install patches on an existing cluster.

The patch installer includes functionality to query a cluster, check contents of a package and compatibility with the cluster, and patch or roll back a cluster.

Installers

The patch installer installs all patches and never modifies configuration. A partial distribution (FP or fix) can only be installed by the patch installer.

The LSF installer installs full distributions and can modify configuration. The LSF installer incorporates the patch installer so the process of updating the files is the same as the patch installer. However, the LSF installer should be used to install an update because the update may require configuration changes that `lsfinstall` can do automatically.

The LSF installer may change with each update. You should not install a new update using the old `lsfinstall` program or `install.config` template; make sure your installers match the version of the distribution you are installing.

Patch installer accessibility

For clusters version 7.0 or earlier, you must obtain the patch installer separately from Platform, and run the `patchinstall` command from your download directory.

For clusters version 7 Update 1 (7.0.1) or later, the patch installer is available under `install` directory under the LSF installation directory. This location may not be in your path, so run the `patchinstall` command from this directory (*LSF_TOP*/8.0/`install/patchinstall`).

Order of installation

If you have to install multiple patches, start with the most recent update, which includes all previous fixes. Install on all UNIX hosts to bring the whole cluster up to date. Then install fixes or fix packs as needed.

Silent install

The silent install option is used for automated installations.

For `lsfinstall`, enable silent install by the `LSF_QUIET_INST` parameter in `install.config`. Silent install hides some messages.

For `patchinstall`, enable silent install by the `--silent` option in the command line. Silent install shows all messages but does not prompt for confirmations.

Windows-UNIX clusters and Windows clusters

If your cluster has both Windows and UNIX, patch the UNIX hosts in the cluster using the patch installer. Patch the Windows hosts using Windows tools.

The Windows patch files should be installed in order from oldest to newest on every Windows host if you have more than one to install.

To install a Windows patch, double click the .msp file for the OS you want and follow the wizard. You may be asked to reboot after installing. Follow the Windows prompts if applicable.

Note:

You can also install silently.

Patch history and backups

History

The patch history is a record of information about patches installed with the patch installer or the LSF installer, including products and patches installed, dates, and location of backups required for rollback purposes.

The `pversions` command retrieves and displays the version information. The patch installer rollback feature retrieves the backup information.

History directory

The patch history information is kept in the patch history directory. The directory location is `LSF_TOP/patch` by default.

The patch history directory is configurable during installation. See the `PATCH_HISTORY_DIR` parameter in `install.config`.

Backups

The patch installer backs up the current installation before attempting to replace files with the newer versions. The backups are saved so that rollback is possible later on.

Patches change relatively few files, but for an update release, all the files in the cluster are backed up, so the amount of space required is large. The more patches you install, the more space is required to save multiple backups.

Backup directory

The patch backup files are kept in the patch backup directory. The directory location is `LSF_TOP/patch/backup` by default.

The patch backup directory is configurable during installation. See the `PATCH_BACKUP_DIR` parameter in `install.config`.

Maintenance

Over time, the backups accumulate. You may choose to manually delete old backups, starting with the oldest. Remember that rollback is performed one patch at a time, so your cluster's rollback functionality stops at the point where a backup file is unavailable.

If the backup directory runs out of space, your installations and rollbacks fail.

You can change your backup directory by setting `PATCH_BACKUP_DIR` in `patch.conf`, but you must copy the contents of the old directory to the new directory manually (or there can be no rollback).

Update release backup control

You can disable backups when installing update releases. In this case, your update is installed without backing up the cluster first, so you cannot remove the update using the rollback functionality.

You might choose this feature to save disk space, to speed up the install process, or if you have your own methods of backing up the cluster.

Backup is always done before installing fixes, so you can always roll back if a fix does not behave as expected.

Multiple daemon files

To make changes without affecting running daemons, the patch installer must move some files to another directory instead of overwriting.

For each file, a new directory is created in parallel with the file. The directory is called `daemons_old`.

Running jobs may require the old files even after you restart the updated cluster.

Cluster patch behavior

When...	Actions...	The result...
Normal behavior.	The installer replaces current files with new.	<ul style="list-style-type: none"> • Success, cluster is updated.
Installing an update and the patch history is missing (files are not found in the directory defined by the parameter <code>PATCH_HISTORY_DIR</code> in <code>patch.conf</code>)	<p>The installer creates new history files in the directory.</p> <p>The installer cannot determine compatibility but installs anyway because an update is a full distribution.</p>	<ul style="list-style-type: none"> • Cluster is modified but if the update is not compatible (a previous version instead of newer version), the cluster may not work properly.
Installing a fix and the patch history is missing (files are not found in the directory defined by the parameter <code>PATCH_HISTORY_DIR</code> in <code>patch.conf</code>)	For a fix, the installer cannot determine compatibility.	<ul style="list-style-type: none"> • No update, cluster remains in same state • Error presented on screen and logged in <code>patch.log</code> and <code>patch.err</code>
The installer is partway through the installation when there is a problem. The cluster contains some older files and some newer files.	If the installer cannot complete, it reverses the update actions, removing the newer files and returning the older ones.	<ul style="list-style-type: none"> • No update, cluster remains in same state. • Error presented on screen and logged
Installing a fix and a file in the cluster is newer than the file in the patch (build number in cluster is larger than build number of patch).	Prompt user to overwrite or preserve file. Install other files in the patch as usual.	<ul style="list-style-type: none"> • Each build of a file is backwards compatible, so this patch works properly with the newer file. • Overwriting the newer file may break functionality of a newer patch in the cluster.
Installing a fix and a file in the cluster has been modified since the last patch (current file size does not match size recorded in patch history).	Prompt user to overwrite or exit.	<ul style="list-style-type: none"> • Overwriting a corrupt file results in correct behavior. • Overwriting a customized file breaks existing functionality. You can modify the updated file manually after installation. • Patch functionality depends on updated content in the new file, so you cannot install the patch if you do not overwrite the file.

Cluster rollback behavior

When...	Actions...	The result...
Normal behavior.	The installer replaces current files with previous backup.	<ul style="list-style-type: none"> • Success, cluster reverts to previous state.
The patch history is missing (files are not found in the directory defined by the parameter <code>PATCH_HISTORY_DIR</code> in <code>patch.conf</code>)	Without the history, the installer cannot determine which backups to use. Since there is nothing to replace them with, the installer does not remove the current files.	<ul style="list-style-type: none"> • No rollback, cluster remains in same state. • Error presented on screen and logged
You did not specify the most recent patch.	The history indicates that the patch is not the newest backup. The installer must use the most recent backup to roll back.	<ul style="list-style-type: none"> • No rollback, cluster remains in same state. • Error presented on screen and logged
The backups are missing (expected files are not found in the directory defined by the parameter <code>PATCH_BACKUP_DIR</code> in <code>patch.conf</code>).	Since there is nothing to replace them with, the installer does not remove the current files.	<ul style="list-style-type: none"> • No rollback, cluster remains in same state. • Error presented on screen and logged
The installer is partway through the roll back when there is a problem. The cluster contains some older files and some newer files.	If the installer cannot complete, it reverses the rollback actions, removing the older files and returning the newer ones.	<ul style="list-style-type: none"> • No rollback, cluster remains in same state. • Error presented on screen and logged

Version management log files

File	Description
patch.log	<p>This file:</p> <ul style="list-style-type: none">• Created by the patch installer (not created if you use <code>lsfi nstall</code>)• Created when you install a patch or update release• Created in current working directory (or if you do not have write permission there, logs to <code>/tmp</code>)• Logs installation steps
precheck.log	<p>This file:</p> <ul style="list-style-type: none">• Created by the patch installer• Created when you install or check a patch• Created in current working directory (or if you do not have write permission there, logs to <code>/tmp</code>)• Logs precheck steps
install.log	<p>This file:</p> <ul style="list-style-type: none">• Created by the LSF installer (not created if you use <code>patchi nstall</code>)• Created when you install a new cluster or update release• Created in current working directory (or if you do not have write permission there, logs to <code>/tmp</code>)• Logs installation steps

Version management commands

Commands to modify cluster

Command	Description
<code>lsfi nstall</code>	<p>This command:</p> <ul style="list-style-type: none"> Creates a new cluster (using any full distribution including update releases) Patches a cluster with an update release (a full distribution) by installing binaries and updating configuration
<code>patchi nstall</code>	<p>This command:</p> <ul style="list-style-type: none"> Patches a cluster by installing binaries from a full or partial distribution (does not update configuration, so <code>lsfi nstall</code> is recommended for an update release)
<code>patchi nstall -r</code>	<p>This command:</p> <ul style="list-style-type: none"> Rolls back a cluster by removing binaries (does not roll back configuration, so rollback of updates may not be recommended)

Commands to monitor cluster

Command	Description
<code>pversi ons</code>	<p>This command:</p> <ul style="list-style-type: none"> Displays product version information for the entire cluster, including patch levels Displays detailed information for specific builds or files in the cluster; for example, see what files were modified after installing a patch
<code>file_name -V</code>	<p>This command:</p> <ul style="list-style-type: none"> Displays detailed information for a specific file in the cluster (specify the installed file, for example <code>l i m - V</code>)

Commands to check uninstalled packages

Command	Description
<code>pversi ons -c</code>	<p>This command:</p> <ul style="list-style-type: none"> Displays detailed information about the contents of an uninstalled package
<code>patchi nstall -c</code>	<p>This command:</p> <ul style="list-style-type: none"> Tests if an uninstalled package is compatible with the cluster

Install update releases on UNIX and Linux

To install an update release to the cluster.

Important:

For LSF 8 Update 2 through 6, you cannot use the steps in this chapter. You must follow the steps in “Migrating to LSF Version 8 on UNIX and Linux” to manually migrate your LSF 8 cluster to LSF 8.

1. Download and extract the new version of `lsfinstall`.

For example,

```
zcat lsfsf8_lsfinstall.tar.Z | tar xvf -
```

2. Prepare the `install.config` file using the new template and information from your original installation. The new template may have new parameters for you to set.
3. Download the patches and put the distribution files in the same directory as `lsfinstall`.

If hosts in your cluster have multiple binary types, you may require multiple distribution files to patch the entire cluster.

4. Run the new LSF installer.

For example,

```
lsfinstall -f install.config
```

Specify the patches to install and let the installer finish.

5. Restart the cluster.

This makes changes to daemons take effect.

6. Optional. Run **pversions** to determine the state of the cluster.
7. Optional. Free some space by deleting the contents of backup directories under EGO and LSF installation directories.

Install fixes on UNIX and Linux

To install fixes or fix packs to update the cluster.

1. Download the patches from Platform and put the distribution files on any host.

For example,

```
//HostB/downloads/pkg1
```

```
//HostB/downloads/pkg2
```

If hosts in your cluster have multiple binary types, you may require multiple distribution files to patch the entire cluster.

2. Log on to a host in the cluster.
3. Set your environment (if you cannot do this, prepare a configuration file and use the `-f` option in the `pversions` and `patchinstall` commands).

```
source LSF_TOP/conf/cshrc.lsf
(for csh or tcsh)
```

```
. LSF_TOP/conf/profile.lsf
(for sh, ksh, or bash)
```

4. Run the patch installer tool and specify the patches to install.

For example,

```
LSF_TOP/8.0/install/patchinstall //HostB/downloads/pkg1 //HostB/downloads/pkg2
```

Let the patch installer finish.

5. If you were prompted to do so, restart the cluster.

Patches that affect running daemons require you to restart manually.

6. Optional. Run `LSF_TOP/8.0/install/pversions` to determine the state of the cluster.
7. Optional. If you were prompted to restart the cluster and have done so, you can free some space by deleting the contents of backup directories under EGO and LSF installation directories.

Roll back patches on UNIX and Linux

Removes patches installed using `patchinstall`, and returns the cluster to a previous state.

1. Log on to a host in the cluster.
2. Set your environment (if you cannot, prepare a configuration file and use `-f` option in `pversions` and `patchinstall` commands).

```
source LSF_TOP/conf/cshrc.lsf  
(for csh or tcsh)
```

```
. LSF_TOP/conf/profile.lsf  
(for sh, ksh, or bash)
```

3. Run `LSF_TOP/8.0/install/pversions` to determine the state of the cluster and find the build number of the last patch installed (roll back one patch at a time).
4. Run `patchinstall` with `-r` and specify the build number of the last patch installed (the patch to be removed).

```
patchinstall -r 12345
```

5. If you were prompted to do so, restart the cluster.
Patches that affect running daemons require you to restart manually.
6. If necessary, modify LSF cluster configuration manually. This may be necessary to roll back an update.
7. Optional. Run `LSF_TOP/8.0/install/pversions` to determine the state of the cluster.

To roll back multiple builds, repeat as required until the cluster is in the state you want.

Enable Platform LSF HPC Features

Enable Platform LSF HPC Features

You can install HPC features on UNIX or Linux hosts. Refer to the installation guide for more information.

When you install, some changes are made for you automatically.

A number of shared resources are added to `lsf.shared` that are required by HPC features. When you upgrade HPC features, you should add the appropriate resource names under the RESOURCES column of the Host section of `lsf.cluster.cluster_name`.

What HPC feature installation does

- Installs HPC binary and configuration files
- Automatically configures the following files:
 - `lsb.hosts`
 - `lsb.modules`
 - `lsb.resources`
 - `lsb.queues`
 - `lsb.cluster.cluster_name`
 - `lsb.conf`
 - `lsb.shared`

lsb.hosts

For the default host, `lsfinstall` enables "!" in the MXJ column of the HOSTS section of `lsb.hosts`. For example:

```
Begin Host
HOST_NAME MXJ    rlm    pg    ls    tmp    DISPATCH_WINDOW # Keywords
#hostA      ()  3.5/4.5  15/   12/15  0      ()              # Example
default     !    ()      ()    ()     ()     ()              #pset host
HPPA11      !    ()      ()    ()     ()     ()
End Host
```

lsb.modules

- Adds the external scheduler plugin module names to the PluginModule section of `lsb.modules`:

```
Begin PluginModule
SCH_PLUGIN RB_PLUGIN SCH_DISABLE_PHASES
schmod_default ()      ()
schmod_fcfs    ()      ()
schmod_fairshare ()     ()
schmod_limit   ()      ()
schmod_reserve ()      ()
schmod_preemption ()    ()
schmod_advrsv  ()      ()
...
schmod_cpuset  ()      ()
schmod_pset    ()      ()
schmod_crayx1  ()      ()
schmod_crayxt3 ()      ()
End PluginModule
```

Note:

The HPC plugin names must be configured after the standard LSF plugin names in the PluginModule list.

lsb.resources

For IBM POE jobs, `lsfinstall` configures the ReservationUsage section in `lsb.resources` to reserve HPS resources on a per-slot basis.

Resource usage defined in the ReservationUsage section overrides the cluster-wide `RESOURCE_RESERVE_PER_SLOT` parameter defined in `lsb.params` if it also exists.

```
Begin ReservationUsage
RESOURCE METHOD
adapter_windows PER_SLOT
ntbl_windows PER_SLOT
csss PER_SLOT
```

```
css0 PER_SLOT
End ReservationUsage
```

lsb.queues

- Configures `hpc_ibm` queue for IBM POE jobs and the `hpc_ibm_tv` queue for debugging IBM POE jobs through Etnus TotalView®.

```
Begin Queue
QUEUE_NAME = hpc_ibm
PRIORITY = 30
NICE = 20
# ...
RES_REQ = select[ poe > 0 ]
EXCLUSIVE = Y
REQUEUE_EXIT_VALUES = 133 134 135
DESCRIPTION = This queue is to run POE jobs ONLY.
End Queue
```

```
Begin Queue
QUEUE_NAME = hpc_ibm_tv
PRIORITY = 30
NICE = 20
# ...
RES_REQ = select[ poe > 0 ]
REQUEUE_EXIT_VALUES = 133 134 135
TERMINATE_WHEN = LOAD PREEMPT WINDOW
RERUNNABLE = NO
INTERACTIVE = NO
DESCRIPTION = This queue is to run POE jobs ONLY.
End Queue
```

- Configures `hpc_linux` queue for LAM/MPI and MPICH-GM jobs and `hpc_linux_tv` queue for debugging LAM/MPI and MPICH-GM jobs through Etnus TotalView®.

```
Begin Queue
QUEUE_NAME = hpc_linux
PRIORITY = 30
NICE = 20
# ...
DESCRIPTION = for linux.
End Queue
```

```
Begin Queue
QUEUE_NAME = hpc_linux_tv
PRIORITY = 30
NICE = 20
# ...
TERMINATE_WHEN = LOAD PREEMPT WINDOW
RERUNNABLE = NO
INTERACTIVE = NO
DESCRIPTION = for linux TotalView Debug queue.
End Queue
```

By default, LSF sends a SIGUSR2 signal to terminate a job that has reached its run limit or deadline. Since LAM/MPI does not respond to the SIGUSR2 signal, you should configure the `hpc_linux` queue with a custom job termination action specified by the `JOB_CONTROLS` parameter.

Note:

To make the one of the LSF queues the default queue, set `DEFAULT_QUEUE` in `lsb.params`.

Use the `bqueues -l` command to view the queue configuration details. Before using HPC features, see the Platform LSF Configuration Reference to understand queue configuration parameters in `lsb.queues`.

lsf.cluster.*cluster_name*

- Removes lsf_data and lsf_parallel from the PRODUCTS line of lsf.cluster.*cluster_name* if they are already there.
- For IBM POE jobs, configures the ResourceMap section of lsf.cluster.*cluster_name* to map the following shared resources for POE jobs to all hosts in the cluster:

```
Begin ResourceMap
RESOURCENAME      LOCATION
adapter_windows   [ default ]
ntbl_windows       [ default ]
poe                [ default ]
dedicated_tasks    (0@[ default ])
ip_tasks           (0@[ default ])
us_tasks           (0@[ default ])
End ResourceMap
```

lsf.conf

- LSB_SUB_COMMANDNAME=Y to lsf.conf to enable the LSF_SUB_COMMANDLINE environment variable required by esub.
- LSF_ENABLE_EXTSCHEDULER=Y
- LSF uses an external scheduler for topology-aware external scheduling.
- LSB_CPUSET_BESTCPUS=Y
- LSF schedules jobs based on the shortest CPU radius in the processor topology using a best-fit algorithm for SGI cpuset allocation.

Note:

LSF_IRIX_BESTCPUS is obsolete.

- On SGI hosts, sets the full path to the SGI vendor MPI library libxmpi.so:
 - On SGI IRIX: LSF_VPLUGIN="/usr/lib32/libxmpi.so"
 - On SGI Altix: LSF_VPLUGIN="/usr/lib/libxmpi.so"

You can specify multiple paths for LSF_VPLUGIN, separated by colons (:). For example, the following configures both /usr/lib32/libxmpi.so for SGI IRIX, and /usr/lib/libxmpi.so for SGI IRIX:

```
LSF_VPLUGIN="/usr/lib32/libxmpi.so:/usr/lib/libxmpi.so"
```

- On HP-UX hosts, sets the full path to the HP vendor MPI library libmpirm.sl


```
LSF_VPLUGIN="/opt/mpi/lib/pa1.1/libmpirm.sl"
```
- LSB_RLA_PORT=*port_number*

Where *port_number* is the TCP port used for communication between the Platform LSF HPC topology adapter (RLA) and sbatchd.

The default port number is 6883.

- LSB_SHORT_HOSTLIST=1

Displays an abbreviated list of hosts in bjobs and bhist for a parallel job where multiple processes of a job are running on a host. Multiple processes are displayed in the following format:

```
processes*hostA
```

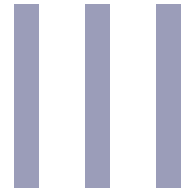
lsf.shared

Defines the following shared resources required by HPC features in `lsf.shared`:

Resource	TYPE	INTERVAL	INCREASING	DESCRIPTION	# Keywords
Begin Resource					
RESOURCENAME	TYPE	INTERVAL	INCREASING	DESCRIPTION	# Keywords
pset	Boolean	()	()	(PSET)	
slurm	Boolean	()	()	(SLURM)	
cpuset	Boolean	()	()	(CPUSET)	
mpich_gm	Boolean	()	()	(MPI CH GM MPI)	
lammpi	Boolean	()	()	(LAM MPI)	
mpichp4	Boolean	()	()	(MPI CH P4 MPI)	
mvapi ch	Boolean	()	()	(Infiniband MPI)	
sca_mpi mon	Boolean	()	()	(SCALI MPI)	
ibmmpi	Boolean	()	()	(IBM POE MPI)	
hpmpi	Boolean	()	()	(HP MPI)	
sgimpi	Boolean	()	()	(SGI MPI)	
intel mpi	Boolean	()	()	(Intel MPI)	
crayxt3	Boolean	()	()	(Cray XT3 MPI)	
crayx1	Boolean	()	()	(Cray X1 MPI)	
fluent	Boolean	()	()	(fluent availability)	
ls_dyna	Boolean	()	()	(ls_dyna availability)	
nastran	Boolean	()	()	(nastran availability)	
pvm	Boolean	()	()	(pvm availability)	
openmp	Boolean	()	()	(openmp availability)	
ansys	Boolean	()	()	(ansys availability)	blast
Boolean ()	()		(blast availability)		
gaussian	Boolean	()	()	(gaussian availability)	
lion	Boolean	()	()	(lion availability)	
scitegic	Boolean	()	()	(scitegic availability)	
schroedinger	Boolean	()	()	(schroedinger availability)	
hammer	Boolean	()	()	(hammer availability)	
adapter_windows	Numeric	30	N	(free adapter windows on css0 on IBM SP)	
ntbl_windows	Numeric	30	N	(free ntbl windows on IBM HPS)	
poe	Numeric	30	N	(poe availability)	
css0	Numeric	30	N	(free adapter windows on css0 on IBM SP)	
csss	Numeric	30	N	(free adapter windows on csss on IBM SP)	
dedicated_tasks	Numeric	()	Y	(running dedicated tasks)	
ip_tasks	Numeric	()	Y	(running IP tasks)	
us_tasks	Numeric	()	Y	(running US tasks)	
End Resource					

Note:

You should add the appropriate resource names under the RESOURCES column of the Host section of `lsf.cluster.cluster_name`.



Monitoring Your Cluster

Achieving Performance and Scalability

Optimize performance in large sites

As your site grows, you must tune your LSF cluster to support a large number of hosts and an increased workload.

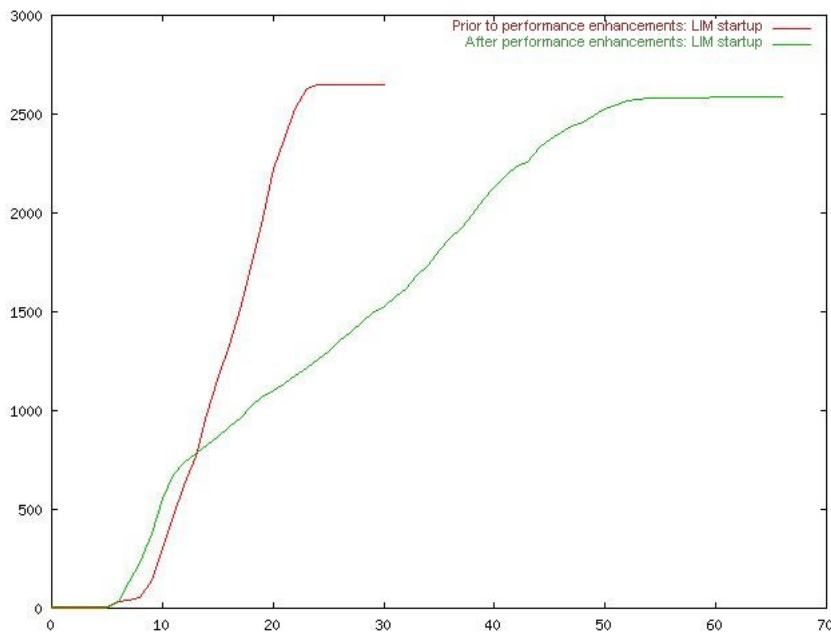
This chapter discusses how to efficiently tune querying, scheduling, and event logging in a large cluster that scales to 5000 hosts and 100,000 jobs at any one time.

What's new in Platform LSF performance?

LSF provides parameters for tuning your cluster, which you will learn about in this chapter. However, before you calculate the values to use for tuning your cluster, consider the following enhancements to the general performance of LSF daemons, job dispatching, and event replaying:

- Both scheduling and querying are much faster
- Switching and replaying the events log file, `lsb.events`, is much faster. The length of the events file no longer impacts performance
- Restarting and reconfiguring your cluster is much faster
- Job submission time is constant. It does not matter how many jobs are in the system. The submission time does not vary.
- The scalability of load updates from the slaves to the master has increased
- Load update intervals are scaled automatically

The following graph shows the improvement in LIM startup after the LSF performance enhancements:



Tune UNIX for large clusters

The following hardware and software specifications are requirements for a large cluster that supports 5,000 hosts and 100,000 jobs at any one time.

Hardware recommendation

LSF master host:

- 4 processors, one each for:
 - `mbat chd`
 - `mbschd`
 - `lim`
 - Operating system
- 10 GB Ram

Software requirement

To meet the performance requirements of a large cluster, increase the file descriptor limit of the operating system.

The file descriptor limit of most operating systems used to be fixed, with a limit of 1024 open files. Some operating systems, such as Linux and AIX, have removed this limit, allowing you to increase the number of file descriptors.

Increase the file descriptor limit

1. To achieve efficiency of performance in LSF, follow the instructions in your operating system documentation to increase the number of file descriptors on the LSF master host.

Tip:

To optimize your configuration, set your file descriptor limit to a value at least as high as the number of hosts in your cluster.

The following is an example configuration. The instructions for different operating systems, kernels, and shells are varied. You may have already configured the host to use the maximum number of file descriptors that are allowed by the operating system. On some operating systems, the limit is configured dynamically.

Your cluster size is 5000 hosts. Your master host is on Linux, kernel version 2.4:

1. Log in to the LSF master host as the root user.
2. Add the following line to your `/etc/rc.d/rc.local` startup script:

```
echo -n "5120" > /proc/sys/fs/file-max
```

3. Restart the operating system to apply the changes.
4. In the bash shell, instruct the operating system to use the new file limits:

```
# ulimit -n unlimited
```

Tune Platform LSF for large clusters

To enable and sustain large clusters, you need to tune LSF for efficient querying, dispatching, and event log management.

Manage scheduling performance

For fast job dispatching in a large cluster, configure the following parameters:

`LSB_MAX_JOB_DISPATCH_PER_SESSION` in `lsf.conf`

The maximum number of jobs the scheduler can dispatch in one scheduling session

Some operating systems, such as Linux and AIX, let you increase the number of file descriptors that can be allocated on the master host. You do not need to limit the number of file descriptors to 1024 if you want fast job dispatching. To take advantage of the greater number of file descriptors, you must set `LSB_MAX_JOB_DISPATCH_PER_SESSION` to a value greater than 300.

Set `LSB_MAX_JOB_DISPATCH_PER_SESSION` to one-half the value of `MAX_SBD_CONNS`. This setting configures `mbatchd` to dispatch jobs at a high rate while maintaining the processing speed of other `mbatchd` tasks.

`MAX_SBD_CONNS` in `lsb.params`

The maximum number of open file connections between `mbatchd` and `sbatchd`.

Specify a value equal to the number of hosts in your cluster plus a buffer. For example, if your cluster includes 4000 hosts, set:

`MAX_SBD_CONNS=4100`

Highly recommended for large clusters to decrease the load on the master LIM. Forces the client `sbatchd` to contact the local LIM for host status and load information. The client `sbatchd` only contacts the master LIM or a LIM on one of the `LSF_SERVER_HOSTS` if `sbatchd` cannot find the information locally.

Enable fast job dispatch

1. Log in to the LSF master host as the root user.
2. Increase the system-wide file descriptor limit of your operating system if you have not already done so.
3. In `lsb.params`, set `MAX_SBD_CONNS` equal to the number of hosts in the cluster plus a buffer.
4. In `lsf.conf`, set the parameter `LSB_MAX_JOB_DISPATCH_PER_SESSION` to a value greater than 300 and less than or equal to one-half the value of `MAX_SBD_CONNS`.

For example, for a cluster with 4000 hosts:

```
LSB_MAX_JOB_DISPATCH_PER_SESSION = 2050
MAX_SBD_CONNS=4100
```

5. In `lsf.conf`, define the parameter `LSF_SERVER_HOSTS` to decrease the load on the master LIM.
6. In the shell you used to increase the file descriptor limit, shut down the LSF batch daemons on the master host:

badm in hshutdown

7. Run `badm in mbdrestart` to restart the LSF batch daemons on the master host.
8. Run `badm in hrestart all` to restart every `sbatchd` in the cluster:

Note:

When you shut down the batch daemons on the master host, all LSF services are temporarily unavailable, but existing jobs are not affected. When `mbat chd` is later started by `sbat chd`, its previous status is restored and job scheduling continues.

Enable continuous scheduling

1. To enable the scheduler to run continuously, define the parameter `JOB_SCHEDULING_INTERVAL=0` in `l sb. params`.

Limit the number of batch queries

In large clusters, job querying can grow quickly. If your site sees a lot of high traffic job querying, you can tune LSF to limit the number of job queries that `mbat chd` can handle. This helps decrease the load on the master host.

If a job information query is sent after the limit has been reached, an error message ("*Batch system concurrent query limit exceeded*") is displayed and `mbat chd` keeps retrying, in one second intervals. If the number of job queries later drops below the limit, `mbat chd` handles the query.

1. Define the maximum number of concurrent jobs queries to be handled by `mbat chd` in the parameter `MAX_CONCURRENT_JOB_QUERY` in `l sb. params`:
 - If `mbat chd` is not using multithreading, the value of `MAX_CONCURRENT_JOB_QUERY` is always the maximum number of job queries in the cluster.
 - If `mbat chd` is using multithreading (defined by the parameter `LSB_QUERY_PORT` in `l sf. conf`), the number of job queries in the cluster can temporarily become higher than the number specified by `MAX_CONCURRENT_JOB_QUERY`.

This increase in the total number of job queries is possible because the value of `MAX_CONCURRENT_JOB_QUERY` actually sets the maximum number of queries that can be handled by each child `mbat chd` that is forked by `mbat chd`. When the new child `mbat chd` starts, it handles new queries, but the old child `mbat chd` continues to run until all the old queries are finished. It is possible that the total number of job queries can be as high as `MAX_CONCURRENT_JOB_QUERY` multiplied by the number of child daemons forked by `mbat chd`.

Syntax:

```
MAX_CONCURRENT_JOB_QUERY=max_query
```

Where:

```
max_query
```

Specifies the maximum number of job queries that can be handled by `mbat chd`. Valid values are positive integers between 1 and 100. The default value is unlimited.

Examples

```
MAX_CONCURRENT_JOB_QUERY=20
```

Specifies that no more than 20 queries can be handled by `mbat chd`.

```
MAX_CONCURRENT_JOB_QUERY=101
```

Incorrect value. The default value will be used. An unlimited number of job queries will be handled by `mbat chd`.

Improve the speed of host status updates

- Configure the parameter `LSB_SYNC_HOST_STAT_LIM` in the file `l sb. params`.

This also improves the speed with which LSF reschedules jobs: the sooner LSF knows that a host has become unavailable, the sooner LSF reschedules any rerunnable jobs executing on that host.

For example, during maintenance operations, the cluster administrator might need to shut down half of the hosts at once. LSF can quickly update the host status and reschedule any rerunnable jobs that were running on the unavailable hosts.

When you define this parameter, `mbat chd` periodically obtains the host status from the master LIM, and then verifies the status by polling each `sbat chd` at an interval defined by the parameters `MBD_SLEEP_TIME` and `LSB_MAX_PROBE_SBD`.

Manage your user's ability to move jobs in a queue

Control whether users can use `bt op` and `bbot` to move jobs to the top and bottom of queues

- Set `JOB_POSITION_CONTROL_BY_ADMIN=Y` in `l sb. params`.

Remember:

You must be an LSF administrator to set this parameter.

When set, only the LSF administrator (including any queue administrators) can use `bbot` and `bt op` to move jobs within a queue. A user attempting to use `bbot` or `bt op` receives the error "User permission denied."

Manage the number of pending reasons

Condense all the host-based pending reasons into one generic pending reason for efficient, scalable management of pending reasons.

- Set `CONDENSE_PENDING_REASONS=Y` in `l sb. params`.

If a job has no other main pending reason, `bj obs -p` or `bj obs -l` will display the following:

Individual host based reasons

If you condense host-based pending reasons, but require a full pending reason list, you can run the following command:

badmin diagnose *<job_ID>*

Remember:

You must be an LSF administrator or a queue administrator to run this command.

Achieve efficient event switching

Periodic switching of the event file can weaken the performance of `mbat chd`, which automatically backs up and rewrites the events file after every 1000 batch job completions. The old `l sb. events` file is moved to `l sb. events. 1`, and each old `l sb. events. n` file is moved to `l sb. events. n+1`.

1. Change the frequency of event switching with the following two parameters in `l sb. params`:
 - `MAX_JOB_NUM` specifies the number of batch jobs to complete before `l sb. events` is backed up and moved to `l sb. events. 1`. The default value is 1000

- `MIN_SWITCH_PERIOD` controls how frequently `mbat chd` checks the number of completed batch jobs

The two parameters work together. Specify the `MIN_SWITCH_PERIOD` value in seconds.

Tip:

For large clusters, set the `MIN_SWITCH_PERIOD` to a value equal to or greater than 600. This causes `mbat chd` to fork a child process that handles event switching, thereby reducing the load on `mbat chd`. `mbat chd` terminates the child process and appends delta events to new events after the `MIN_SWITCH_PERIOD` has elapsed. If you define a value less than 600 seconds, `mbatchd` will not fork a child process for event switching.

Example

This instructs `mbat chd` to check if the events file has logged 1000 batch job completions every two hours. The two parameters can control the frequency of the events file switching as follows:

- After two hours, `mbat chd` checks the number of completed batch jobs. If 1000 completed jobs have been logged, it switches the events file
- If 1000 jobs complete after five minutes, `mbat chd` does not switch the events file until till the end of the two-hour period

Automatic load updates

Periodically, the LIM daemons exchange load information. In large clusters, let LSF automatically load the information by dynamically adjusting the period based on the load.

Important:

For automatic tuning of the loading interval, make sure the parameter `EXINTERVAL` in `lsf.cluster.cluster_name` file is *not* defined. Do not configure your cluster to load the information at specific intervals.

Manage I/O performance of the info directory

In large clusters, the large numbers of jobs results in a large number of job files stored in the `LSF_SHAREDIR/cluster_name/logdir/info` directory at any time. When the total size of the job files reaches a certain point, you will notice a significant delay when performing I/O operations in the `info` directory due to file server directory limits dependent on the file system implementation.

By dividing the total file size of the `info` directory among subdirectories, your cluster can process more job operations before reaching the total size limit of the job files.

1. Use `MAX_INFO_DIRS` in `lsb.params` to create subdirectories and enable `mbat chd` to distribute the job files evenly throughout the subdirectories.

```
MAX_INFO_DIRS=num_subdirs
```

Where `num_subdirs` specifies the number of subdirectories that you want to create under the `LSF_SHAREDIR/cluster_name/logdir/info` directory. Valid values are positive integers between 1 and 1024. By default, `MAX_INFO_DIRS` is not defined.

2. Run `badm in reconfig` to create and use the subdirectories.

Note:

If you enabled duplicate event logging, you must run `badmin mbdrestart` instead of `badmin reconfig` to restart `mbatchd`.

3. Run `bparams -l` to display the value of the `MAX_INFO_DIRS` parameter.

Example

```
MAX_INFO_DIRS=10
```

`mbatchd` creates ten subdirectories from `LSB_SHAREDIR/cluster_name/1` to `LSB_SHAREDIR/cluster_name/10`.

Processor binding for Platform LSF job processes

Rapid progress of modern processor manufacture technologies has enabled the low cost deployment of LSF on hosts with multicore and multithread processors. The default soft affinity policy enforced by the operating system scheduler may not give optimal job performance. For example, the operating system scheduler may place all job processes on the same processor or core leading to poor performance. Frequently switching processes as the operating system schedules and reschedules work between cores can cause cache invalidations and cache miss rates to grow large.

Processor binding for LSF job processes takes advantage of the power of multiple processors and multiple cores to provide hard processor binding functionality for sequential LSF jobs and parallel jobs that run on a single host.

Restriction:

Processor binding is supported on hosts running Linux with kernel version 2.6 or higher.

For multi-host parallel jobs, LSF sets two environment variables (`$LSB_BIND_JOB` and `$LSB_BIND_CPU_LIST`) but does not attempt to bind the job to any host.

When processor binding for LSF job processes is enabled on supported hosts, job processes of an LSF job are bound to a processor according to the binding policy of the host. When an LSF job is completed (exited or done successfully) or suspended, the corresponding processes are unbound from the processor.

When a suspended LSF job is resumed, the corresponding processes are bound again to a processor. The process is not guaranteed to be bound to the same processor it was bound to before the job was suspended.

The processor binding affects the whole job process group. All job processes forked from the root job process (the job RES) are bound to the same processor.

Processor binding for LSF job processes does not bind daemon processes.

If processor binding is enabled, but the execution hosts do not support processor affinity, the configuration has no effect on the running processes. Processor binding has no effect on a single-processor host.

Processor, core, and thread-based CPU binding

By default, the number of CPUs on a host represents the number of cores a machine has. For LSF hosts with multiple cores, threads, and processors, `ncpus` can be defined by the cluster administrator to consider one of the following:

- Processors
- Processors and cores

- Processors, cores, and threads

Globally, this definition is controlled by the parameter `EGO_DEFINE_NCPUS` in `lsf.conf` or `ego.conf`. The default behavior for `ncpus` is to consider the number of cores (`EGO_DEFINE_NCPUS=cores`).

Note:

When `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the resource requirement string keyword `ncpus` refers to the number of slots instead of the number of CPUs, however `lshosts` output will continue to show `ncpus` as defined by `EGO_DEFINE_NCPUS` in `lsf.conf`.

Binding job processes randomly to multiple processors, cores, or threads, may affect job performance. Processor binding configured with `LSF_BIND_JOB` in `lsf.conf` or `BIND_JOB` in `lsb.appl` cat i o n s, detects the `EGO_DEFINE_NCPUS` policy to bind the job processes by processor, core, or thread (PCT).

For example, if a host's PCT policy is set to processor (`EGO_DEFINE_NCPUS=procs`) and the binding option is set to `BALANCE`, the first job process is bound to the first physical processor, the second job process is bound to the second physical processor and so on.

If host's PCT policy is set to core level (`EGO_DEFINE_NCPUS=cores`) and the binding option is set to `BALANCE`, the first job process is bound to the first core on the first physical processor, the second job process is bound to the first core on the second physical processor, the third job process is bound to the second core on the first physical processor and so on.

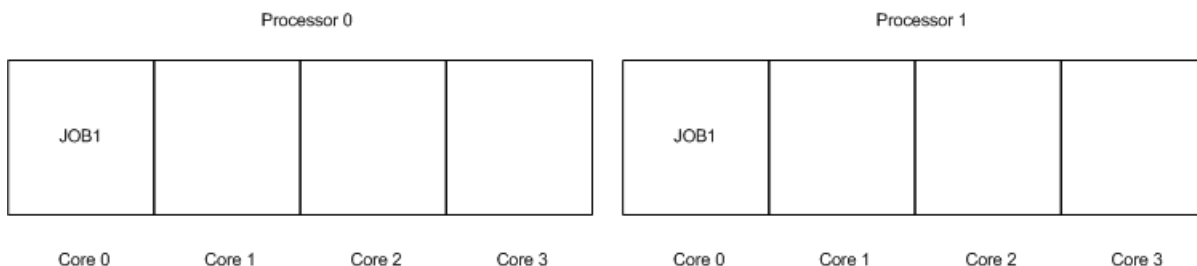
If host's PCT policy is set to thread level (`EGO_DEFINE_NCPUS=threads`) and the binding option is set to `BALANCE`, the first job process is bound to the first thread on the first physical processor, the second job process is bound to the first thread on the second physical processor, the third job process is bound to the second thread on the first physical processor and so on.

BIND_JOB=BALANCE

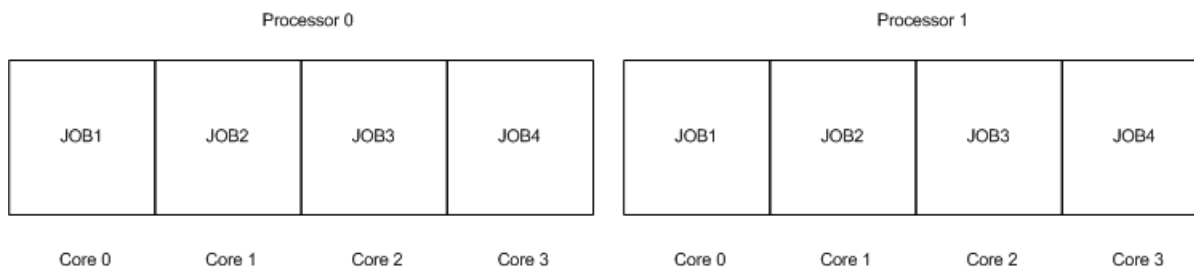
The `BIND_JOB=BALANCE` option instructs LSF to bind the job based on the load of the available processors/cores/threads. For each slot:

- If the PCT level is set to processor, the lowest loaded physical processor runs the job.
- If the PCT level is set to core, the lowest loaded core on the lowest loaded processor runs the job.
- If the PCT level is set to thread, the lowest loaded thread on the lowest loaded core on the lowest loaded processor runs the job.

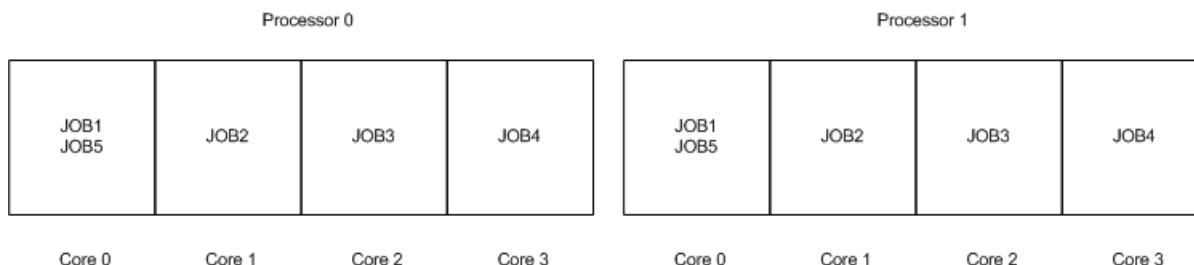
If there is a single 2 processor quad core host and you submit a parallel job with `-n 2 -R"span [hosts=1] "` when the PCT level is core, the job is bound to the first core on the first processor and the first core on the second processor:



After submitting another three jobs with `-n 2 -R"span[hosts=1]":`



If `PARALLEL_SCHED_BY_SLOT=Y` is set in `lsb.params`, the job specifies a maximum and minimum number of job slots instead of processors. If the `MXJ` value is set to 16 for this host (there are 16 job slots on this host), LSF can dispatch more jobs to this host. Another job submitted to this host is bound to the first core on the first processor and the first core on the second processor:



BIND_JOB=PACK

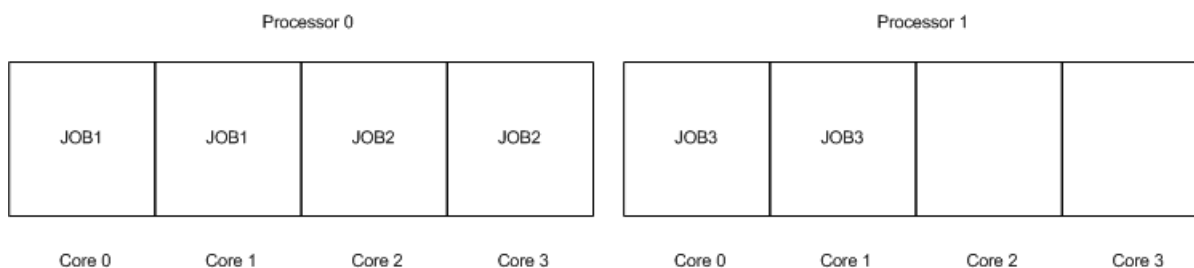
The `BIND_JOB=PACK` option instructs LSF to try to pack all the processes onto a single processor. If this cannot be done, LSF tries to use as few processors as possible. Email is sent to you after job dispatch and when job finishes. If no processors/cores/threads are free (when the `PCT` level is processor/core/thread level), LSF tries to use the `BALANCE` policy for the new job.

LSF depends on the order of processor IDs to pack jobs to a single processor.

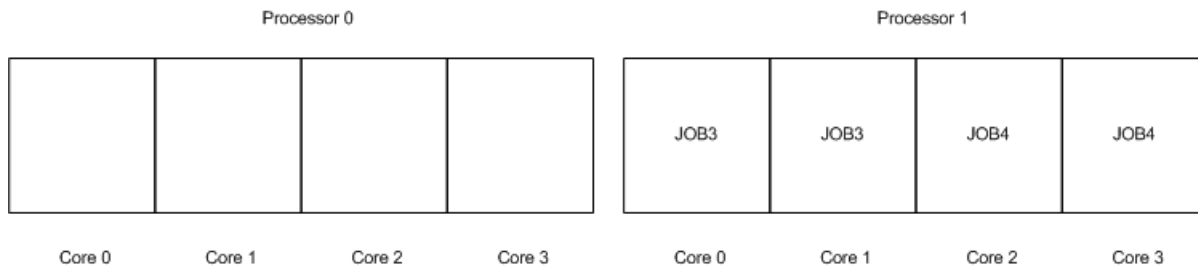
If `PCT` level is processor (default value after installation), there is no difference between `BALANCE` and `PACK`.

This option binds jobs to a single processor where it makes sense, but does not oversubscribe the processors/cores/threads. The other processors are used when they are needed. For instance, when the `PCT` level is core level, if we have a single 4 processor quad core host and we had bound 4 sequential jobs onto the first processor, the 5th-8th sequential job is bound to the second processor.

If you submit three single-host parallel jobs with `-n 2 -R"span[hosts=1]"` when the `PCT` level is core level, the first job is bound to the first and second cores of the first processor, the second job is bound to the third and fourth cores of the first processor. Binding the third job to the first processor oversubscribes the cores in the first processor, so the third job is bound to the first and second cores of the second processor:



After JOB1 and JOB2 finished, if you submit one single-host parallel jobs with **-n 2 -R "span[hosts=1]**, the job is bound to the third and fourth cores of the second processor:



BIND_JOB=ANY

BIND_JOB=ANY binds the job to the first N available processors/cores/threads with no regard for locality. If the PCT level is core, LSF binds the first N available cores regardless of whether they are on the same processor or not. LSF arranges the order based on APIC ID.

If PCT level is processor (default value after installation), there is no difference between ANY and BALANCE.

For example, with a single 2-processor quad core host and the below table is the relationship of APIC ID and logic processor/core id:

APC ID	Processor ID	Core ID
0	0	0
1	0	1
2	0	2
3	0	3
4	1	0
5	1	1
6	1	2
7	1	3

If the PCT level is core level and you submits two jobs to this host with **-n 3 -R "span[hosts=1]"**, then the first job is bound to the first, second and third core of the first physical processor, the second job is bound to the fourth core of the first physical processor and the first, second core in the second physical processor.

BIND_JOB=USER

BIND_JOB=USER binds the job to the value of \$LSB_USER_BIND_JOB as specified in the user submission environment. This allows the Administrator to delegate binding decisions to the actual user. This value must be one of Y, N, NONE, BALANCE, PACK, or ANY. Any other value is treated as ANY.

BIND_JOB=USER_CPU_LIST

`BIND_JOB=USER_CPU_LIST` binds the job to the explicit logic CPUs specified in environment variable `$LSB_USER_BIND_CPU_LIST`. LSF does not check that the value is valid for the execution host(s). It is the user's responsibility to correctly specify the CPU list for the hosts they select.

The correct format of `$LSB_USER_BIND_CPU_LIST` is a list which may contain multiple items, separated by comma, and ranges. For example, 0,5,7,9-11.

If the value's format is not correct or there is no such environment variable, jobs are not bound to any processor.

If the format is correct and it cannot be mapped to any logic CPU, the binding fails. But if it can be mapped to some CPUs, the job is bound to the mapped CPUs. For example, with a two-processor quad core host and the logic CPU ID is 0-7:

1. If user1 specifies 9,10 into `$LSB_USER_BIND_CPU_LIST`, his job is not bound to any CPUs.
2. If user2 specifies 1,2,9 into `$LSB_USER_BIND_CPU_LIST`, his job is bound to CPU 1 and 2.

If the value's format is not correct or it does not apply for the execution host, the related information is added to the email sent to users after job dispatch and job finish.

If user specifies a minimum and a maximum number of processors for a single-host parallel job, LSF may allocate processors between these two numbers for the job. In this case, LSF binds the job according to the CPU list specified by the user.

BIND_JOB=NONE

`BIND_JOB=NONE` is functionally equivalent to the former `BIND_JOB=N` where the processor binding is disabled.

Feature interactions

- Existing CPU affinity features

Processor binding of LSF job processes will not take effect on a master host with the following parameters configured.

- `MBD_QUERY_CPUS`
- `LSF_DAEMONS_CPUS`
- `EGO_DAEMONS_CPUS`
- IRIX cpusets

Processor binding cannot be used with IRIX cpusets. If an execution host is configured as part of a cpuset, processor binding is disabled on that host.

- Job requeue, rerun, and migration

When a job is requeued, rerun or migrated, a new job process is created. If processor binding is enabled when the job runs, the job processes will be bound to a processor.

- `badmi n hrestart`

`badmi n hrestart` restarts a new `sbatchd`. If a job process has already been bound to a processor, after `sbatchd` is restarted, processor binding for the job processes are restored.

- `badmi n reconfi g`

If the `BIND_JOB` parameter is modified in an application profile, `badmi n reconfi g` only affects pending jobs. The change does not affect running jobs.

- MultiCluster job forwarding model

In a MultiCluster environment, the behavior is similar to the current application profile behavior. If the application profile name specified in the submission cluster is not defined in the execution cluster, the job is rejected. If the execution cluster has the same application profile name, but does not enable processor binding, the job processes are not bound at the execution cluster.

Enable processor binding for Platform LSF job processes

LSF supports the following binding options for sequential jobs and parallel jobs that run on a single host:

- BALANCE
- PACK
- ANY
- USER
- USER_CPU_LIST
- NONE

1. Enable processor binding cluster-wide or in an application profile.

- Cluster-wide configuration (`lsf.conf`)

Define `LSF_BIND_JOB` in `lsf.conf` to enable processor binding for all execution hosts in the cluster. On the execution hosts that support this feature, job processes will be hard bound to selected processors.

- Application profile configuration (`lsb.application`)

Define `BIND_JOB` in an application profile configuration in `lsb.application` to enable processor binding for all jobs submitted to the application profile. On the execution hosts that support this feature, job processes will be hard bound to selected processors.

If `BIND_JOB` is not set in an application profile in `lsb.application`, the value of `LSF_BIND_JOB` in `lsf.conf` takes effect. The `BIND_JOB` parameter configured in an application profile overrides the `lsf.conf` setting.

Job ID limit

By default, LSF assigns job IDs up to 6 digits. This means that no more than 999999 jobs can be in the system at once. The job ID limit is the highest job ID that LSF will ever assign, and also the maximum number of jobs in the system.

LSF assigns job IDs in sequence. When the job ID limit is reached, the count rolls over, so the next job submitted gets job ID "1". If the original job 1 remains in the system, LSF skips that number and assigns job ID "2", or the next available job ID. If you have so many jobs in the system that the low job IDs are still in use when the maximum job ID is assigned, jobs with sequential numbers could have different submission times.

Increase the maximum job ID

You cannot lower the job ID limit, but you can raise it to 10 digits. This allows longer term job accounting and analysis, and means you can have more jobs in the system, and the job ID numbers will roll over less often.

Use `MAX_JOBID` in `lsb.params` to specify any integer from 999999 to 2147483646 (for practical purposes, you can use any 10-digit integer less than this value).

Increase the job ID display length

By default, `bj obs` and `bhist` display job IDs with a maximum length of 7 characters. Job IDs greater than 9999999 are truncated on the left.

Use `LSB_JOBID_DISP_LENGTH` in `lsf.conf` to increase the width of the JOBID column in `bjobs` and `bhist` display. When `LSB_JOBID_DISP_LENGTH=10`, the width of the JOBID column in `bj obs` and `bhist` increases to 10 characters.

Monitor performance metrics in real time

Enable metric collection

Set `SCHED_METRIC_ENABLE=Y` in `l sb. params` to enable performance metric collection.

Start performance metric collection dynamically:

`badmin perfmon start sample_period`

Optionally, you can set a sampling period, in seconds. If no sample period is specified, the default sample period set in `SCHED_METRIC_SAMPLE_PERIOD` in `l sb. params` is used.

Stop sampling:

`badmin perfmon stop`

`SCHED_METRIC_ENABLE` and `SCHED_METRIC_SAMPLE_PERIOD` can be specified independently. That is, you can specify `SCHED_METRIC_SAMPLE_PERIOD` and not specify `SCHED_METRIC_ENABLE`. In this case, when you turn on the feature dynamically (using `badmin perfmon start`), the sampling period valued defined in `SCHED_METRIC_SAMPLE_PERIOD` will be used.

`badmin perfmon start` and `badmin perfmon stop` override the configuration setting in `l sb. params`. Even if `SCHED_METRIC_ENABLE` is set, if you run `badmin perfmon start`, performance metric collection is started. If you run `badmin perfmon stop`, performance metric collection is stopped.

Tune the metric sampling period

Set `SCHED_METRIC_SAMPLE_PERIOD` in `l sb. params` to specify an initial cluster-wide performance metric sampling period.

Set a new sampling period in seconds:

`badmin perfmon setperiod sample_period`

Collecting and recording performance metric data may affect the performance of LSF. Smaller sampling periods will result in the `l sb. streams` file growing faster.

Display current performance

Run `badmin perfmon view` to view real time performance metric information. The following metrics are collected and recorded in each sample period:

- The number of queries handled by `mbatchd`
- The number of queries for each of jobs, queues, and hosts. (`bj_obs`, `bqueues`, and `bhosts`, as well as other daemon requests)
- The number of jobs submitted (divided into job submission requests and jobs actually submitted)
- The number of jobs dispatched
- The number of jobs completed
- The numbers of jobs sent to remote cluster
- The numbers of jobs accepted by from cluster

`badmin perfmon view`

```
Performance monitor start time:  Fri Jan 19 15:07:54
End time of last sample period:  Fri Jan 19 15:25:55
Sample period :                  60 Seconds
-----
```

Metrics	Last	Max	Min	Avg	Total
Total queries	0	25	0	8	159
Jobs information queries	0	13	0	2	46
Hosts information queries	0	0	0	0	0
Queue information queries	0	0	0	0	0
Job submission requests	0	10	0	0	10
Jobs submitted	0	100	0	5	100
Jobs dispatched	0	0	0	0	0
Jobs completed	0	13	0	5	100
Jobs sent to remote cluster	0	12	0	5	100
Jobs accepted from remote cluster	0	0	0	0	0
<hr/>					
File Descriptor Metrics		Free	Used	Total	
<hr/>					
MBD file descriptor usage		800	424	1024	

Performance metrics information is calculated at the end of each sampling period. Running `badmin perfmon` before the end of the sampling period displays metric data collected from the sampling start time to the end of last sample period.

If no metrics have been collected because the first sampling period has not yet ended, `badmin perfmon view` displays:

badmin perfmon view

```
Performance monitor start time: Thu Jan 25 22:11:12
End time of last sample period: Thu Jan 25 22:11:12
Sample period : 120 Seconds
```

No performance metric data available. Please wait until first sample period ends.

badmin perfmon output

Sample Period	Current sample period
Performance monitor start time	The start time of sampling
End time of last sample period	The end time of last sampling period
Metric	The name of metrics
Total	This is accumulated metric counter value for each metric. It is counted from Performance monitor start time to End time of last sample period.
Last Period	Last sampling value of metric. It is calculated per sampling period. It is represented as the metric value per period, and normalized by the following formula.
	$LastPeriod = \frac{\text{Metric Counter Value of Last Period}}{\text{Sample Period Interval}} \times \text{Sample Period}$
Max	Maximum sampling value of metric. It is re-evaluated in each sampling period by comparing Max and Last Period. It is represented as the metric value per period.
Min	Minimum sampling value of metric. It is re-evaluated in each sampling period by comparing Min and Last Period. It is represented as the metric value per period.
Avg	Average sampling value of metric. It is recalculated in each sampling period. It is represented as the metric value per period, and normalized by the following formula.

$$Avg = \frac{Total}{LastPeriodEndTime - SampleStartTime} \times SamplePeriod$$

Reconfigure your cluster with performance metric sampling enabled

badmimbdrestart

If performance metric sampling is enabled dynamically with `badmimperfmonstart`. You must enable it again after running `badmimbdrestart`. If performance metric sampling is enabled by default, `StartTime` will be reset to the point `mbatchd` is restarted.

badmimreconfig

If `SCHED_METRIC_ENABLE` and `SCHED_METRIC_SAMPLE_PERIOD` parameters are changed, `badmimreconfig` is the same as `badmimbdrestart`.

Performance metric logging in lsb.streams

By default, collected metrics must be written to `lsb.streams`. However, performance metric can still be turned on even if `ENABLE_EVENT_STREAM=N` is defined. In this case, no metric data will be logged.

- If `EVENT_STREAM_FILE` is defined and is valid, collected metrics should be written to `EVENT_STREAM_FILE`.
- If `ENABLE_EVENT_STREAM=N` is defined, metrics data will not be logged.

Job arrays and job packs

Every job submitted in a job array or job pack is counted individually, except for the `Job submission requests` metric. The entire job array or job pack counts as just one job submission request.

Job rerun

Job rerun occurs when execution hosts become unavailable while a job is running, and the job will be put to its original queue first and later will be dispatched when a suitable host is available. So in this case, only one submission request, one job submitted, and n jobs dispatched, n jobs completed are counted (n represents the number of times the job reruns before it finishes successfully).

Job queue

Queued jobs may be dispatched, run, and exit due to some special errors again and again. The job data always exists in the memory, so LSF only counts one job submission request and one job submitted, and counts more than one job dispatched.

For jobs completed, if a job is queued with `brequeue`, LSF counts two jobs completed, since queuing a job first kills the job and later puts the job into pending list. If the job is automatically queued, LSF counts one job completed when the job finishes successfully.

Job replay

When job replay is finished, submitted jobs are not counted in job submission and job submitted, but are counted in job dispatched and job finished.

Event Generation

Event generation

LSF detects events occurring during the operation of LSF daemons. LSF provides a program which translates LSF events into SNMP traps. You can also write your own program that runs on the master host to interpret and respond to LSF events in other ways. For example, your program could:

- Page the system administrator
- Send email to all users
- Integrate with your existing network management software to validate and correct the problem

On Windows, use the Windows Event Viewer to view LSF events.

SNMP trap program

If you use the LSF SNMP trap program as the event handler, see the SNMP documentation for instructions on how to enable event generation.

Enable event generation for custom programs

If you use a custom program to handle the LSF events, take the following steps to enable event generation.

1. Write a custom program to interpret the arguments passed by LSF.
2. To enable event generation, define `LSF_EVENT_RECEIVER` in `lsf.conf`. You must specify an event receiver even if your program ignores it.

The event receiver maintains cluster-specific or changeable information that you do not want to hard-code into the event program. For example, the event receiver could be the path to a current log file, the email address of the cluster administrator, or the host to send SNMP traps to.

3. Set `LSF_EVENT_PROGRAM` in `lsf.conf` and specify the name of your custom event program. If you name your event program `genevent` (`genevent.exe` on Windows) and place it in `LSF_SERVERDIR`, you can skip this step.
4. Reconfigure the cluster with the commands `lsadm in reconf ig` and `badmi n reconf ig`.

Events list

The following daemon operations cause `mbat chd` or the master LIM to call the event program to generate an event. Each LSF event is identified by a predefined number, which is passed as an argument to the event program. Events 1-9 also return the name of the host on which an event occurred.

1. LIM goes down (detected by the master LIM). This event may also occur if LIM temporarily stops communicating to the master LIM.
2. RES goes down (detected by the master LIM).
3. `sbat chd` goes down (detected by `mbat chd`).
4. An LSF server or client host becomes unlicensed (detected by the master LIM).
5. A host becomes the new master host (detected by the master LIM).
6. The master host stops being the master (detected by the master LIM).
7. `mbat chd` comes up and is ready to schedule jobs (detected by `mbat chd`).
8. `mbat chd` goes down (detected by `mbat chd`).
9. `mbat chd` receives a reconfiguration request and is being reconfigured (detected by `mbat chd`).
10. `LSB_SHAREDIR` becomes full (detected by `mbat chd`).
11. The administrator opens a host.
12. The administrator closes a host.
13. The administrator opens a queue.
14. The administrator closes a queue.
15. `mbschd` goes down.
16. A license is being overused.

Arguments passed to the Platform LSF event program

If `LSF_EVENT_RECEIVER` is defined, a function called `ls_postevent()` allows specific daemon operations to generate LSF events. This function then calls the LSF event program and passes the following arguments:

- The event receiver (`LSF_EVENT_RECEIVER` in `lsf.conf`)
- The cluster name
- The LSF event number (LSF events list or `LSF_EVENT_XXXX` macros in `lsf.h`)
- The event argument (for events that take an argument)

Example

For example, if the event receiver is the string `xxx` and LIM goes down on Host A in Cluster 1, the function returns:

```
xxx Cluster1 1 HostA
```

The custom LSF event program can interpret or ignore these arguments.

18

Tuning the Cluster

Tune LIM

LIM provides critical services to all LSF components. In addition to the timely collection of resource information, LIM provides host selection and job placement policies. If you are using Platform MultiCluster, LIM determines how different clusters should exchange load and resource information. You can tune LIM policies and parameters to improve performance.

LIM uses load thresholds to determine whether to place remote jobs on a host. If one or more LSF load indices exceeds the corresponding threshold (too many users, not enough swap space, etc.), then the host is regarded as busy and LIM will not recommend jobs to that host. You can also tune LIM load thresholds.

Adjust LIM Parameters

There are two main goals in adjusting LIM configuration parameters: improving response time, and reducing interference with interactive use. To improve response time, tune LSF to correctly select the best available host for each job. To reduce interference, tune LSF to avoid overloading any host.

LIM policies are advisory information for applications. Applications can either use the placement decision from LIM, or make further decisions based on information from LIM.

Most of the LSF interactive tools use LIM policies to place jobs on the network. LSF uses load and resource information from LIM and makes its own placement decisions based on other factors in addition to load information.

Files that affect LIM are `lsf.shared`, `lsf.cluster.cluster_name`, where `cluster_name` is the name of your cluster.

RUNWINDOW parameter

LIM thresholds and run windows affect the job placement advice of LIM. Job placement advice is not enforced by LIM.

The `RUNWINDOW` parameter defined in `lsf.cluster.cluster_name` specifies one or more time windows during which a host is considered available. If the current time is outside all the defined time windows, the host is considered locked and LIM will not advise any applications to run jobs on the host.

Load thresholds

Load threshold parameters define the conditions beyond which a host is considered busy by LIM and are a major factor in influencing performance. No jobs will be dispatched to a busy host by LIM's policy. Each of these parameters is a load index value, so that if the host load goes beyond that value, the host becomes busy.

LIM uses load thresholds to determine whether to place remote jobs on a host. If one or more LSF load indices exceeds the corresponding threshold (too many users, not enough swap space, etc.), then the host is regarded as busy and LIM will not recommend jobs to that host.

Thresholds can be set for any load index supported internally by the LIM, and for any external load index.

If a particular load index is not specified, LIM assumes that there is no threshold for that load index. Define looser values for load thresholds if you want to aggressively run jobs on a host.

Load indices that affect LIM performance

Load index	Description
r15s	15-second CPU run queue length
r1m	1-minute CPU run queue length
r15m	15-minute CPU run queue length
pg	Paging rate in pages per second
swp	Available swap space
it	Interactive idle time
ls	Number of users logged in

Compare LIM load thresholds

Tune LIM load thresholds, compare the output of `lsl load` to the thresholds reported by `lshosts -l`.

1. Run `lshosts -l`
2. Run `lsl load`

The `lsl load` and `lsmom` commands display an asterisk * next to each load index that exceeds its threshold.

Example

Consider the following output from `lshosts -l` and `lsl load`:

```
lshosts -l
HOST_NAME:  hostD
...
LOAD_THRESHOLDS:
  r15s  r1m  r15m  ut   pg   io   ls   it   tmp  swp  mem
  -      3.5  -      -    15   -    -    -    -    2M   1M

HOST_NAME:  hostA
...
LOAD_THRESHOLDS:
  r15s  r1m  r15m  ut   pg   io   ls   it   tmp  swp  mem
  -      3.5  -      -    15   -    -    -    -    2M   1M

lsl load
HOST_NAME  status  r15s  r1m  r15m  ut   pg   ls   it  tmp  swp  mem
hostD      ok      0.0  0.0  0.0   0%   0.0  6    0  30M  32M  10M
hostA      busy    1.9  2.1  1.9  47% *69.6 21   0  38M  96M  60M
```

In this example, the hosts have the following characteristics:

- hostD is ok.
- hostA is busy — The pg (paging rate) index is 69.6, above the threshold of 15.

LIM reports a host as busy

If LIM often reports a host as busy when the CPU utilization and run queue lengths are relatively low and the system is responding quickly, the most likely cause is the paging rate threshold. Try raising the pg threshold.

Different operating systems assign subtly different meanings to the paging rate statistic, so the threshold needs to be set at different levels for different host types. In particular, HP-UX systems need to be configured with significantly higher pg values; try starting at a value of 50.

There is a point of diminishing returns. As the paging rate rises, eventually the system spends too much time waiting for pages and the CPU utilization decreases. Paging rate is the factor that most directly affects perceived interactive response. If a system is paging heavily, it feels very slow.

Interactive jobs

If you find that interactive jobs slow down system response while LIM still reports your host as ok, reduce the CPU run queue lengths (r15s, r1m, r15m). Likewise, increase CPU run queue lengths if hosts become busy at low loads.

Multiprocessor systems

On multiprocessor systems, CPU run queue lengths (r15s, r1m, r15m) are compared to the effective run queue lengths as displayed by the `lsload -E` command.

CPU run queue lengths should be configured as the load limit for a single processor. Sites with a variety of uniprocessor and multiprocessor machines can use a standard value for r15s, r1m and r15m in the configuration files, and the multiprocessor machines will automatically run more jobs.

Note that the normalized run queue length displayed by `lsload -N` is scaled by the number of processors.

How Platform LSF works with LSF_MASTER_LIST

The files `lsf.shared` and `lsf.cluster.cluster_name` are shared only among LIMs listed as candidates to be elected master with the parameter `LSF_MASTER_LIST`.

The preferred master host is no longer the first host in the cluster list in `lsf.cluster.cluster_name`, but the first host in the list specified by `LSF_MASTER_LIST` in `lsf.conf`.

Whenever you reconfigure, only master LIM candidates read `lsf.shared` and `lsf.cluster.cluster_name` to get updated information. The elected master LIM sends configuration information to slave LIMs.

The order in which you specify hosts in `LSF_MASTER_LIST` is the preferred order for selecting hosts to become the master LIM.

Non-shared file considerations

Generally, the files `lsf.cluster.cluster_name` and `lsf.shared` for hosts that are master candidates should be identical.

When the cluster is started up or reconfigured, LSF rereads configuration files and compares `lsf.cluster.cluster_name` and `lsf.shared` for hosts that are master candidates.

In some cases in which identical files are not shared, files may be out of sync. This section describes situations that may arise should `lsf.cluster.cluster_name` and `lsf.shared` for hosts that are master candidates not be identical to those of the elected master host.

LSF_MASTER_LIST host eligibility

LSF only rejects candidate master hosts listed in `LSF_MASTER_LIST` from the cluster if the number of load indices in `lsf.cluster.cluster_name` or `lsf.shared` for master candidates is different from the number of load indices in the `lsf.cluster.cluster_name` or `lsf.shared` files of the elected master.

A warning is logged in the log file `lim.log`. `master_host_name` and the cluster continues to run, but without the hosts that were rejected.

If you want the hosts that were rejected to be part of the cluster, ensure the number of load indices in `lsf.cluster.cluster_name` and `lsf.shared` are identical for all master candidates and restart LIMs on the master and all master candidates:

lsadmin limrestart hostA hostB hostC

Failover with ineligible master host candidates

If the elected master host goes down and if the number of load indices in `lsf.cluster.cluster_name` or `lsf.shared` for the new elected master is different from the number of load indices in the files of the master that went down, LSF will reject all master candidates that do not have the same number of load indices in their files as the newly elected master. LSF will also reject all slave-only hosts. This could cause a situation in which only the newly elected master is considered part of the cluster.

A warning is logged in the log file `lim.log`. `new_master_host_name` and the cluster continues to run, but without the hosts that were rejected.

To resolve this, from the current master host, restart all LIMs:

lsadmin limrestart all

All slave-only hosts will be considered part of the cluster. Master candidates with a different number of load indices in their `lsf.cluster.cluster_name` or `lsf.shared` files will be rejected.

When the master that was down comes back up, you need to ensure load indices defined in `lsf.cluster.cluster_name` and `lsf.shared` for all master candidates are identical and restart LIMs on all master candidates.

Improve performance of mbatchd query requests on UNIX

You can improve mbatchd query performance on UNIX systems using the following methods:

- **Multithreading**—On UNIX platforms that support thread programming, you can change default mbatchd behavior to use multithreading and increase performance of query requests when you use the `bj obs` command. Multithreading is beneficial for busy clusters with many jobs and frequent query requests. This may indirectly increase overall mbatchd performance.
- **Hard CPU affinity**—You can specify the master host CPUs on which mbatchd child query processes can run. This improves mbatchd scheduling and dispatch performance by binding query processes to specific CPUs so that higher priority mbatchd processes can run more efficiently.

mbatchd without multithreading

Ports

By default, mbatchd uses the port defined by the parameter `LSB_MBD_PORT` in `lsf.conf` or looks into the system services database for port numbers to communicate with LIM and job request commands.

It uses this port number to receive query requests from clients.

Service requests

For every query request received, mbatchd forks a child mbatchd to service the request. Each child mbatchd processes the request and then exits.

Configure mbatchd to use multithreading

When mbatchd has a dedicated port specified by the parameter `LSB_QUERY_PORT` in `lsf.conf`, it forks a child mbatchd which in turn creates threads to process query requests.

As soon as mbatchd has forked a child mbatchd, the child mbatchd takes over and listens on the port to process more query requests. For each query request, the child mbatchd creates a thread to process it.

The child mbatchd continues to listen to the port number specified by `LSB_QUERY_PORT` and creates threads to service requests until the job status changes, a new job is submitted, or until the time specified in `MBD_REFRESH_TIME` in `lsb.params` has passed.

Specify a time interval, in seconds, when mbatchd will fork a new child mbatchd to service query requests to keep information sent back to clients updated. A child mbatchd processes query requests creating threads.

`MBD_REFRESH_TIME` has the following syntax:

`MBD_REFRESH_TIME=seconds [min_refresh_time]`

where *min_refresh_time* defines the minimum time (in seconds) that the child mbatchd will stay to handle queries. The valid range is 0 - 300. The default is 5 seconds.

- If `MBD_REFRESH_TIME` is < *min_refresh_time*, the child mbatchd exits at `MBD_REFRESH_TIME` even if the job changes status or a new job is submitted before `MBD_REFRESH_TIME` expires.
- If `MBD_REFRESH_TIME` > *min_refresh_time*
 - the child mbatchd exits at *min_refresh_time* if a job changes status or a new job is submitted before the *min_refresh_time*

- the child `mbatchd` exits after the *min_refresh_time* when a job changes status or a new job is submitted
- If `MBD_REFRESH_TIME > min_refresh_time` and no job changes status or a new job is submitted, the child `mbatchd` exits at `MBD_REFRESH_TIME`

The default for *min_refresh_time* is 10 seconds.

If you use the `bj obs` command and do not get up-to-date information, you may want to decrease the value of `MBD_REFRESH_TIME` or `MIN_REFRESH_TIME` in `l sb. params` to make it likely that successive job queries could get the newly-submitted job information.

Note:

Lowering the value of `MBD_REFRESH_TIME` or `MIN_REFRESH_TIME` increases the load on `mbatchd` and might negatively affect performance.

1. Specify a query-dedicated port for the `mbatchd` by setting `LSB_QUERY_PORT` in `l sf. conf`.
2. Optional: Set an interval of time to indicate when a new child `mbatchd` is to be forked by setting `MBD_REFRESH_TIME` in `l sb. params`. The default value of `MBD_REFRESH_TIME` is 5 seconds, and valid values are 0-300 seconds.
3. Optional: Use `NEWJOB_REFRESH=Y` in `l sb. params` to enable a child `mbatchd` to get up to date new job information from the parent `mbatchd`.

Set a query-dedicated port for `mbatchd`

To change the default `mbatchd` behavior so that `mbatchd` forks a child `mbatchd` that can create threads, specify a port number with `LSB_QUERY_PORT` in `l sf. conf`.

Tip:

This configuration only works on UNIX platforms that support thread programming.

1. Log on to the host as the primary LSF administrator.
2. Edit `l sf. conf`.
3. Add the `LSB_QUERY_PORT` parameter and specify a port number that will be dedicated to receiving requests from hosts.
4. Save the `l sf. conf` file.
5. Reconfigure the cluster:

badmim mbdrestart

Specify an expiry time for child `mbatchds` (optional)

Use `MBD_REFRESH_TIME` in `l sb. params` to define how often `mbatchd` forks a new child `mbatchd`.

1. Log on to the host as the primary LSF administrator.
2. Edit `l sb. params`.
3. Add the `MBD_REFRESH_TIME` parameter and specify a time interval in seconds to fork a child `mbatchd`.

The default value for this parameter is 5 seconds. Valid values are 0 to 300 seconds.

4. Save the `l sb. params` file.

5. Reconfigure the cluster as follows:

badmin reconfig

Specify hard CPU affinity

You can specify the master host CPUs on which `mbat chd` child query processes can run (hard CPU affinity). This improves `mbat chd` scheduling and dispatch performance by binding query processes to specific CPUs so that higher priority `mbat chd` processes can run more efficiently.

When you define this parameter, LSF runs `mbat chd` child query processes *only* on the specified CPUs. The operating system can assign other processes to run on the same CPU, however, if utilization of the bound CPU is lower than utilization of the unbound CPUs.

1. Identify the CPUs on the master host that will run `mbat chd` child query processes.

- Linux: To obtain a list of valid CPUs, run the command

/proc/cpuinfo

- Solaris: To obtain a list of valid CPUs, run the command

psrinfo

2. In the file `lsb.params`, define the parameter `MBD_QUERY_CPUS`.

For example, if you specify:

MBD_QUERY_CPUS=1 2

the `mbat chd` child query processes will run only on CPU numbers 1 and 2 on the master host.

You can specify CPU affinity only for master hosts that use one of the following operating systems:

- Linux 2.6 or higher
- Solaris 8 or higher

If failover to a master host candidate occurs, LSF maintains the hard CPU affinity, provided that the master host candidate has the same CPU configuration as the original master host. If the configuration differs, LSF ignores the CPU list and reverts to default behavior.

3. Verify that the `mbat chd` child query processes are bound to the correct CPUs on the master host.
 - a) Start up a query process by running a query command such as `bj obs`.
 - b) Check to see that the query process is bound to the correct CPU.
 - Linux: Run the command **taskset -p <pid>**
 - Solaris: Run the command **ps -AP**

Configure mbatchd to push new job information to child mbatchd

`LSB_QUERY_PORT` must be defined in `lsf.conf`.

If you have enabled multithreaded `mbatchd` support, the `bjobs` command may not display up-to-date information if two consecutive query commands are issued before a child `mbatchd` expires because child `mbatchd` job information is not updated. Use `NEWJOB_REFRESH=Y` in `lsb.params` to enable a child `mbat chd` to get up to date new job information from the parent `mbat chd`.

When `NEWJOB_REFRESH=Y` the parent `mbat chd` pushes new job information to a child `mbat chd`. Job queries with `bj obs` display new jobs submitted after the child `mbat chd` was created.

1. Log on to the host as the primary LSF administrator.

2. Edit l sb. params.
3. Add NEWJOB_REFRESH=Y.

You should set MBD_REFRESH_TIME in l sb. params to a value greater than 10 seconds.

4. Save the l sb. params file.
5. Reconfigure the cluster as follows:

badmin reconfig

Authentication and Authorization

LSF uses authentication and authorization to ensure the security of your cluster. The authentication process verifies the identity of users, hosts, and daemons, depending on the security requirements of your site. The authorization process enforces user account permissions.

Change authentication method

During LSF installation, the authentication method is set to external authentication (`eauth`), which offers the highest level of security.

- Set `LSF_AUTH` in `lsf.conf`.
 - For external authentication (the default), set `LSF_AUTH=eauth`
 - For authentication using the `identd` daemon, set `LSF_AUTH=ident`
 - For privileged port authentication, leave `LSF_AUTH` undefined

Note:

If you change the authentication method while LSF daemons are running, you must shut down and restart the daemons on all hosts in order to apply the changes.

When the external authentication (`eauth`) feature is enabled, you can also configure LSF to authenticate daemons by defining the parameter `LSF_AUTH_DAEMONS` in `lsf.conf`.

All authentication methods supported by LSF depend on the security of the `root` account on all hosts in the cluster.

Authentication options

Authentication method	Description	Configuration	Behavior
External authentication	<ul style="list-style-type: none"> A framework that enables you to integrate LSF with any third-party authentication product—such as Kerberos or DCE Security Services—to authenticate users, hosts, and daemons. This feature provides a secure transfer of data within the authentication data stream between LSF clients and servers. Using external authentication, you can customize LSF to meet the security requirements of your site. 	LSF_AUTH=eauth	<ul style="list-style-type: none"> LSF uses the default <code>eauth</code> executable located in <code>LSF_SERVERDIR</code>. The default executable provides an example of how the <code>eauth</code> protocol works. You should write your own <code>eauth</code> executable to meet the security requirements of your cluster. For a detailed description of the external authentication feature and how to configure it, see External Authentication on page 309.
Identification daemon (<code>identd</code>)	<ul style="list-style-type: none"> Authentication using the <code>identd</code> daemon available in the public domain. 	LSF_AUTH=ident	<ul style="list-style-type: none"> LSF uses the <code>identd</code> daemon available in the public domain. LSF supports both RFC 931 and RFC 1413 protocols.
Privileged ports (<code>setuid</code>)	<ul style="list-style-type: none"> User authentication between LSF clients and servers on UNIX hosts only. An LSF command or other executable configured as <code>setuid</code> uses a reserved (privileged) port number (1-1024) to contact an LSF server. The LSF server accepts requests received on a privileged port as coming from the <code>root</code> user and then runs the LSF command or other executable using the real user account of the user who issued the command. 	LSF_AUTH not defined	<ul style="list-style-type: none"> For UNIX hosts only, LSF clients (API functions) use reserved ports 1-1024 to communicate with LSF servers. The number of user accounts that can connect concurrently to remote hosts is limited by the number of available privileged ports. LSF_AUTH must be deleted or commented out and LSF commands must be installed as <code>setuid</code> programs owned by <code>root</code>.

UNIX user and host authentication

The primary LSF administrator can configure additional authentication for UNIX users and hosts by defining the parameter `LSF_USE_HOSTEQUIV` in the `lsf.conf` file. With `LSF_USE_HOSTEQUIV` defined, `mbatchd` on the master host and `RES` on the remote host call the `ruserok(3)` function to verify that the originating host is listed in the `/etc/hosts.equiv` file and that the host and user account are listed in the `$HOME/.rhosts` file. Include the name of the local host in both files. This additional level of authentication works in conjunction with `eauth`, privileged ports (`setuid`), or `identd` authentication.

Caution:

Using the `/etc/hosts.equiv` and `$HOME/.rhosts` files grants permission to use the `rlogin` and `rsh` commands without requiring a password.

SSH

SSH is a network protocol that provides confidentiality and integrity of data using a secure channel between two networked devices. Use SSH to secure communication between submission, execution, and display hosts.

A frequently used option is to submit jobs with SSH X11 forwarding (`bsub -XF`), which allows a user to log into an X-Server client, access the submission host through the client, and run an interactive X-Window job, all through SSH.

Strict checking protocol in an untrusted environment

To improve security in an untrusted environment, the primary LSF administrator can enable the use of a strict checking communications protocol. When you define `LSF_STRICT_CHECKING` in `lsf.conf`, LSF authenticates messages passed between LSF daemons and between LSF commands and daemons. This type of authentication is *not* required in a secure environment, such as when your cluster is protected by a firewall.

Important:

You must shut down the cluster before adding or deleting the `LSF_STRICT_CHECKING` parameter.

Authentication failure

If authentication fails (the user's identity cannot be verified), LSF displays the following error message after a user issues an LSF command:

User permission denied

This error has several possible causes depending on the authentication method used.

Authentication method	Possible cause of failure
auth	<ul style="list-style-type: none">External authentication failed
identd	<ul style="list-style-type: none">The identification daemon is not available on the local or submitting host
setuid	<ul style="list-style-type: none">The LSF applications are not installed setuidThe NFS directory is mounted with the nosuid option
ruserok	<ul style="list-style-type: none">The client (local) host is not found in either the <code>/etc/hosts.equiv</code> or the <code>\$HOME/.rhosts</code> file on the master or remote host

Operating system authorization

By default, an LSF job or command runs on the execution host under the user account that submits the job or command, with the permissions associated with that user account. Any UNIX or Windows user account with read and execute permissions for LSF commands can use LSF to run jobs—the LSF administrator does not need to define a list of LSF users. User accounts must have the operating system permissions required to execute commands on remote hosts. When users have valid accounts on all hosts in the cluster, they can run jobs using their own account permissions on any execution host.

All external executables invoked by the LSF daemons, such as `esub`, `eexec`, `elim`, `eauth`, and `pre-` and `post-execution` commands, run under the `lsfadmin` user account.

Windows passwords

Windows users must register their Windows user account passwords with LSF by running the command `lspasswd`. If users change their passwords, they must use this command to update LSF. A Windows job does not run if the password is not registered in LSF. Passwords must be 31 characters or less.

For Windows password authorization in a non-shared file system environment, you must define the parameter `LSF_MASTER_LIST` in `lsf.conf` so that jobs run with correct permissions. If you do not define this parameter, LSF assumes that the cluster uses a shared file system environment.

LSF authorization

As an LSF administrator, you have the following authorization options:

- Enable one or more types of user account mapping
- Specify the user account used to run `eauth` and `eexec` executables or pre- and post-execution commands
- Control user access to LSF resources and functionality

Enable user account mapping

You can configure different types of user account mapping so that a job or command submitted by one user account runs on the remote host under a different user account.

Type of account mapping	Description
Between-host	Enables job submission and execution within a cluster that has different user accounts assigned to different hosts. Using this feature, you can map a local user account to a different user account on a remote host.
Cross-cluster	Enables cross-cluster job submission and execution for a MultiCluster environment that has different user accounts assigned to different hosts. Using this feature, you can map user accounts in a local cluster to user accounts in one or more remote clusters.
UNIX/Windows	Enables cross-platform job submission and execution in a mixed UNIX/Windows environment. Using this feature, you can map Windows user accounts, which include a domain name, to UNIX user accounts, which do not include a domain name, for user accounts with the same user name on both operating systems.

For a detailed description of the user account mapping features and how to configure them, see [UNIX/Windows User Account Mapping](#) on page 199.

Specify a user account

To change the user account for ...	Define the parameter ...	In the file ...
<code>eauth</code>	<code>LSF_EAUTH_USER</code>	<code>lsf.sudoers</code>
<code>eexec</code>	<code>LSF_EEXEC_USER</code>	
Pre- and post execution commands	<code>LSB_PRE_POST_EXEC_USER</code>	

Control user access to LSF resources and functionality

If you want to ...	Define ...	In the file ...	Section ...
Specify the user accounts with cluster administrator privileges	<code>ADMINISTRATORS</code>	<code>lsf.cluster. cluster_name</code>	ClusterAdmins
Allow the root user to run jobs on a remote host	<code>LSF_ROOT_REX</code>	<code>lsf.conf</code>	N/A

If you want to ...	Define ...	In the file ...	Section ...
Allow specific user accounts to use @ for host redirection with <code>lscsh</code>	LSF_SHELL_AT_USE RS	<code>lsf.conf</code>	N/A
Allow user accounts other than root to start LSF daemons	LSF_STARTUP_USER S LSF_STARTUP_PATH	<code>lsf.sudoers</code>	N/A
Note: For information about how to configure the LSF daemon startup control feature, see Platform LSF Daemon Startup Control on page 33.			

Authorization failure

Symptom	Probable cause	Solution
User receives an email notification that LSF has placed a job in the USUSP state.	The job cannot run because the Windows password for the job is not registered with LSF.	<p>The user should</p> <ul style="list-style-type: none"> Register the Windows password with LSF using the command <code>lspasswd</code>. Use the <code>brresume</code> command to resume the suspended job.
<p>LSF displays one of the following error messages:</p> <ul style="list-style-type: none"> <code>findHostbyAddr/<proc>: Host <host>/<port> is unknown by <myhostname></code> <code>function: Gethostbyaddr_(<host>/<port>) failed: error</code> <code>main: Request from unknown host <host>/<port>: error</code> <code>function: Received request from non-LSF host <host>/<port></code> 	The LSF daemon does not recognize <i>host</i> as part of the cluster. These messages can occur if you add <i>host</i> to the configuration files without reconfiguring all LSF daemons.	<p>Run the following commands after adding a host to the cluster:</p> <ul style="list-style-type: none"> <code>lsadmin reconfig</code> <code>badmin mbdrestart</code> <p>If the problem still occurs, the host might have multiple addresses. Match all of the host addresses to the host name by either:</p> <ul style="list-style-type: none"> Modifying the system hosts file (<code>/etc/hosts</code>). The changes affect all software programs on your system. Creating an LSF hosts file (<code>EGO_CONFDIR/hosts</code>). Only LSF resolves the addresses to the specified host.
<ul style="list-style-type: none"> <code>doacceptconn: getpwnam (<username>@<host>/<port>) failed: error</code> <code>doacceptconn: User <username> has uid <uid1> on client host <host>/<port>, uid <uid2> on RES host; assume bad user</code> <code>authRequest: username/uid <userName>/<uid>@<host>/<port> does not exist</code> <code>authRequest: Submitter's name <clname>@<clhost> is different from name <lname> on this host</code> 	RES assumes that a user has the same UNIX user name and user ID on all LSF hosts. These messages occur if this assumption is violated.	If the user is allowed to use LSF for interactive remote execution, make sure the user's account has the same user ID and user name on all LSF hosts.
<ul style="list-style-type: none"> <code>doacceptconn: root remote execution permission denied</code> <code>authRequest: root job submission rejected</code> 	The root user tried to execute or submit a job but <code>LSF_ROOT_REX</code> is not defined in <code>lsf.conf</code> .	To allow the root user to run jobs on a remote host, define <code>LSF_ROOT_REX</code> in <code>lsf.conf</code> .

Symptom	Probable cause	Solution
<ul style="list-style-type: none"> resControl: operation permission denied, uid = <uid> 	<p>The user with user ID <i>uid</i> is not allowed to make RES control requests. By default, only the LSF administrator can make RES control requests.</p>	<p>To allow the root user to make RES control requests, define LSF_ROOT_REX in <i>lsf.conf</i>.</p>
<ul style="list-style-type: none"> do_restartReq: Failed to get hData of host <host_name>/<host_addr> 	<p><i>mbatchd</i> received a request from <i>sbatchd</i> on host <i>host_name</i>, but that host is not known to <i>mbatchd</i>. Either</p> <ul style="list-style-type: none"> The configuration file has been changed but <i>mbatchd</i> has not been reconfigured. <i>host_name</i> is a client host but <i>sbatchd</i> is running on that host. 	<p>To reconfigure <i>mbatchd</i>, run the command <code>badmi n reconf i g</code></p> <p>To shut down <i>sbatchd</i> on <i>host_name</i>, run the command <code>badmi n hshut down host_name</code></p>

Submitting Jobs with SSH

Secure Shell (SSH) is a network protocol that provides confidentiality and integrity of data using a secure channel between two networked devices.

About SSH

SSH uses public-key cryptography to authenticate the remote computer and allow the remote computer to authenticate the user, if necessary.

SSH is typically used to log into a remote machine and execute commands, but it also supports tunneling, forwarding arbitrary TCP ports and X11 connections. SSH uses a client-server protocol.

SSH uses private/public key pairs to log into another host. Users no longer have to supply a password every time they log on to a remote host.

SSH is used when running any of the following:

- Remote log on to a lightly loaded host (`l s l o g i n`)
- An interactive job (`b s u b - I S | - I S p | I S s`)
- An interactive X-window job with X11 forwarding (`b s u b - X F`)
- An interactive X-window job, without X11 forwarding (`b s u b - I X`)
- An externally submitted job (`e s u b`)

X-Window job options

Depending on your requirements for X-Window jobs, you can choose either `b s u b - X F` (recommended) or `b s u b - I X`. Both options encrypt the X-Server and X-Clients.

Mode	Benefits	Drawbacks
<code>b s u b - X F</code> (X11 forwarding): Recommended	<ul style="list-style-type: none"> • Any password required can be typed in when needed. • Does not require the X-Server host to have the SSH daemon installed. 	<ul style="list-style-type: none"> • The user must enable X11 forwarding in the client. • Submission and execution hosts must be UNIX.
<code>b s u b - I X</code> (interactive X-window)	<ul style="list-style-type: none"> • The execution host contacts the X-Server host directly (no user steps required). • Hosts can be any OS that OpenSSH supports. 	<ul style="list-style-type: none"> • Requires the SSH daemon installed on the X-Server host. • Must use private keys with no passwords set.

Scope

Table 2: SSH X11 forwarding (-XF)

Applicability	Details
Dependencies	<ul style="list-style-type: none"> • OpenSSH 3.9p1 and up is supported. • OpenSSL 0.9.7a and up is supported. • You must have SSH correctly installed on all hosts in the cluster. • You must use an SSH client to log on to the submission host from the display host. • You must install and run the X-Server program on the display host.
Operating system	<ul style="list-style-type: none"> • Only UNIX for submission and execution hosts. The display host can be any operating system.

Applicability	Details
Limitations	<ul style="list-style-type: none"> • You cannot run with <code>bsub -K</code>, <code>-IX</code>, or <code>-r</code>. • You cannot <code>bmod</code> a job submitted with X11 forwarding. • Cannot be used with job arrays, job chunks, or user account mapping. • Jobs submitted with X11 forwarding cannot be checked or modified by <code>esubs</code>. • Can only run on UNIX hosts (submission and execution hosts).

Table 3: Interactive X-window without X11 forwarding (-IX)

Applicability	Details
Dependencies	<ul style="list-style-type: none"> • You must have OpenSSH correctly installed on all hosts in the cluster. • You must generate public/private key pairs and add the content of the public key to the <code>authorized_keys</code> file on remote hosts. For more information, refer to your SSH documentation. • For X-window jobs: <ul style="list-style-type: none"> • You must set the <code>DISPLAY</code> environment variable to <code>X-serverHost:0.0</code>, where <code>X-serverHost</code> is the name of the X-window server. Ensure the X-server can access itself. Run, for example, <code>xhost +local host</code>.
Operating system	<ul style="list-style-type: none"> • Any OS that also supports OpenSSH.
Limitations	<ul style="list-style-type: none"> • Cannot be used with job arrays or job chunks. • Private user keys must have no password set. • You cannot run with <code>-K</code>, <code>-r</code>, or <code>-XF</code>.

Configuration to enable SSH

No LSF configuration is needed to enable SSH X11 forwarding.

Remote log on to a lightly loaded host (l s l o g i n):

Configuration file	Level	Syntax	Behavior
l s f . c o n f	System	LSF_LSLOGIN_SSH=Y y	A user with SSH configured can log on to a remote host without providing a password. All communication between local and remote hosts is encrypted.

Configuration to modify SSH (X11 forwarding)

Configuration file	Level	Syntax	Behavior
lsf.conf	System	LSB_SSH_XFORWARD_CMD	For X11 forwarding, you can modify the default value with an SSH command (full PATH and options allowed).

SSH commands

Commands to submit

Command	Behavior
<code>bsub -IS</code>	Submits a batch interactive job under a secure shell (ssh).
<code>bsub -ISp</code>	Submits a batch interactive job under a secure shell and creates a pseudo-terminal when the job starts.
<code>bsub -ISs</code>	Submits a batch interactive job under a secure shell and creates a pseudo-terminal with shell mode support when the job starts. Use for interactive shells or applications that redefine the CTRL-C and CTRL-Z keys (for example, jove).
<code>bsub -IX</code>	Submits an interactive X-window job., secured using SSH.
<code>bsub -XF</code>	Submits a job with SSH X11 forwarding.
<code>bsub -XF -I</code>	Submits an interactive job with SSH X11 forwarding. The session displays throughout the job lifecycle.

Commands to monitor

Command	Behavior
<code>netstat -an</code>	Displays all active TCP connections and the TCP and UDP ports on which the computer is listening.
<code>bjobs -l</code>	Displays job information, including any jobs submitted with SSH X11 forwarding.
<code>bhist -l</code>	Displays historical job information, including any jobs submitted with SSH X11 forwarding.

Troubleshoot SSH X11 forwarding (-XF)

SSH X11 forwarding must already be working outside LSF.

1. Enable the following flags in `lsf.conf`:

- `LSF_NIOS_DEBUG=1`
- `LSF_LOG_MASK="LOG_DEBUG"`

Troubleshoot SSH (-IX)

Use the SSH command on the job execution host to connect it securely with the job submission host.

If the host fails to connect, you can perform the following steps to troubleshoot.

1. Check the SSH version on both hosts.

If the hosts have different SSH versions, a message displays identifying a protocol version mismatch.

2. Check that public and private key pairs are correctly configured.

More information on configuring key pairs is here: <http://sial.org/howto/openssh/publickey-auth/>.

3. Check the domain name.

```
$ ssh -f -L 6000:localhost:6000 domain_name.example.com date
```

```
$ ssh -f -L 6000:localhost:6000 domain_name date
```

If these commands return errors, troubleshoot the domain name with the error information returned.

The execution host should connect without passwords and pass phrases.

```
$ ssh sahpi a03
$ ssh sahpi a03. example. com
```


External Authentication

The external authentication feature provides a framework that enables you to integrate LSF with any third-party authentication product—such as Kerberos or DCE Security Services—to authenticate users, hosts, and daemons. This feature provides a secure transfer of data within the authentication data stream between LSF clients and servers. Using external authentication, you can customize LSF to meet the security requirements of your site.

About external authentication (eauth)

The external authentication feature uses an executable file called `eauth`. You can write an `eauth` executable that authenticates users, hosts, and daemons using a site-specific authentication method such as Kerberos or DCE Security Services client authentication. You can also specify an external encryption key (recommended) and the user account under which `eauth` runs.

Important:

LSF uses an internal encryption key by default. To increase security, configure an external encryption key by defining the parameter `LSF_EAUTH_KEY` in `lsf.sudoers`.

During LSF installation, a default `eauth` executable is installed in the directory specified by the parameter `LSF_SERVERDIR` in `lsf.conf`. The default executable provides an example of how the `eauth` protocol works. You should write your own `eauth` executable to meet the security requirements of your cluster.

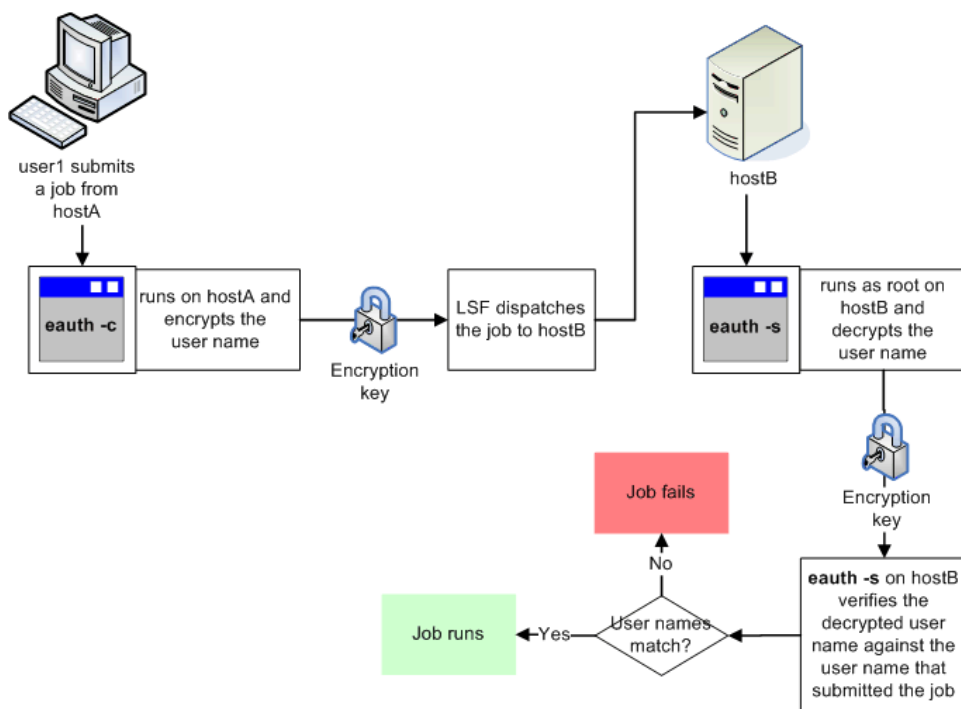


Figure 14: Default behavior (`eauth` executable provided with LSF)

The `eauth` executable uses corresponding processes `eauth -c host_name` (client) and `eauth -s` (server) to provide a secure data exchange between LSF daemons on client and server hosts. The variable *host_name* refers to the host on which `eauth -s` runs; that is, the host called by the command. For `bsub`, for example, the *host_name* is `NULL`, which means the authentication data works for any host in the cluster.

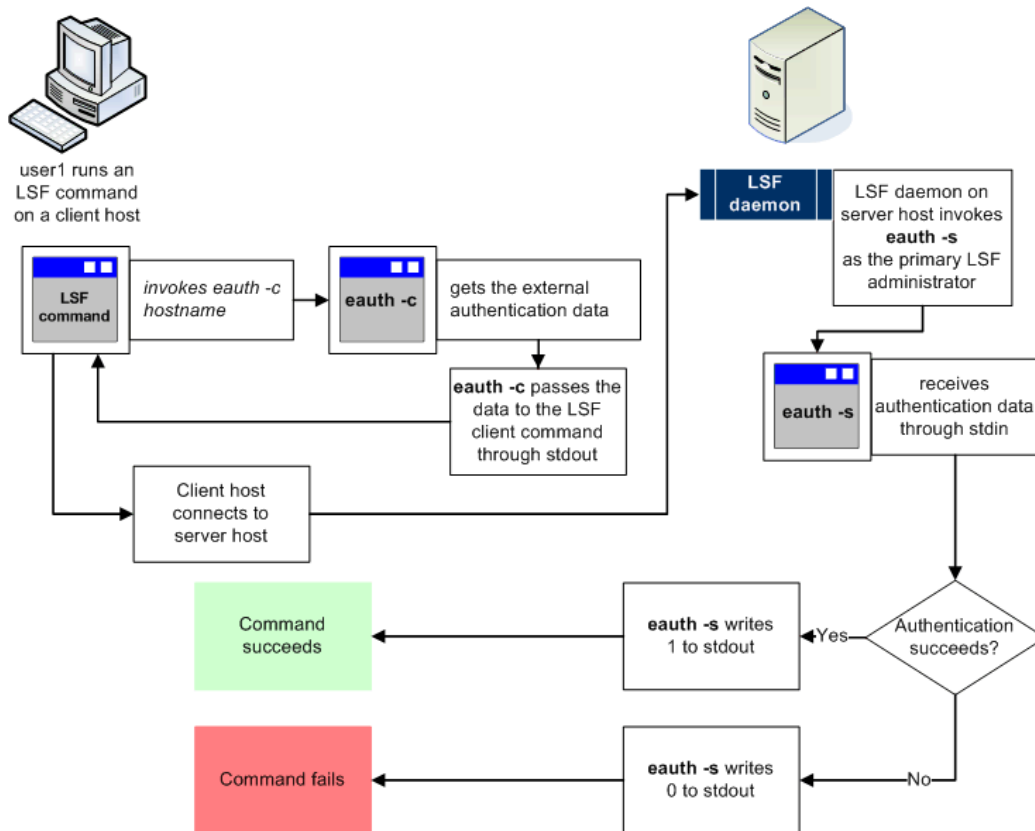


Figure 15: How eauth works

One `eauth -s` process can handle multiple authentication requests. If `eauth -s` terminates, the LSF daemon invokes another instance of `eauth -s` to handle new authentication requests.

The standard input stream to `eauth -s` is a text string with the following format:

```
uid gid user_name client_addr client_port user_auth_data_len eauth_client eauth_server
aux_data_file aux_data_status user_auth_data
```

where

The variable ...	Represents the ...
<i>uid</i>	User ID of the client user
<i>gid</i>	Group ID of the client user
<i>user_name</i>	User name of the client user
<i>client_addr</i>	IP address of the client host
<i>client_port</i>	Port number from which the client request originates
<i>user_auth_data_len</i>	Length of the external authentication data passed from the client host
<i>eauth_client</i>	Daemon or user that invokes <code>eauth -c</code>
<i>eauth_server</i>	Daemon that invokes <code>eauth -s</code>

The variable ...	Represents the ...
<i>aux_data_file</i>	Location of the temporary file that stores encrypted authentication data
<i>aux_data_status</i>	File in which <code>eaut h -s</code> stores authentication status. When used with Kerberos authentication, <code>eaut h -s</code> writes the source of authentication to this file if authentication fails. For example, if <code>mbat chd</code> to <code>mbat chd</code> authentication fails, <code>eaut h -s</code> writes "mbat chd" to the file defined by <i>aux_data_status</i> . If user to <code>mbat chd</code> authentication fails, <code>eaut h -s</code> writes "user" to the <i>aux_data_status</i> file.
<i>user_auth_data</i>	External authentication data passed from the client host

The variables required for the `eaut h` executable depend on how you implement external authentication at your site. For `eaut h` parsing, unused variables are marked by "".

User credentials

When an LSF user submits a job or issues a command, the LSF daemon that receives the request verifies the identity of the user by checking the user credentials. External authentication provides the greatest security of all LSF authentication methods because the user credentials are obtained from an external source, such as a database, and then encrypted prior to transmission. For Windows hosts, external authentication is the only truly secure type of LSF authentication.

Host credentials

LSF first authenticates users and then checks host credentials. LSF accepts requests sent from all hosts configured as part of the LSF cluster, including floating clients and any hosts that are dynamically added to the cluster. LSF rejects requests sent from a non-LSF host. If your cluster requires additional host authentication, you can write an `eaut h` executable that verifies both user and host credentials.

Daemon credentials

Daemon authentication provides a secure channel for passing credentials between hosts, mediated by the master host. The master host mediates authentication by means of the `eaut h` executable, which ensures secure passing of credentials between submission hosts and execution hosts, even though the submission host does not know which execution host will be selected to run a job.

Daemon authentication applies to the following communications between LSF daemons:

- `mbat chd` requests to `sbat chd`
- `sbat chd` updates to `mbat chd`
- PAM interactions with `res`
- `mbat chd` to `mbat chd` (in a MultiCluster environment)

Kerberos authentication

Kerberos authentication is an extension of external daemon authentication, providing authentication of LSF users and daemons during client-server interactions. The `eaut h` executable provided with the Platform integration package uses Kerberos Version 5 APIs for interactions between `mbat chd` and `sbat chd`, and between `pam` and `res`. When you use Kerberos authentication for a cluster or MultiCluster, authentication data is encrypted along the entire path from job submission through to job completion.

You can also use Kerberos authentication for delegation of rights (forwarding credentials) when a job requires a Kerberos ticket during job execution. LSF ensures that a ticket-granting ticket (TGT) can be

forwarded securely to the execution host. LSF also automatically renews Kerberos credentials by means of daemon wrapper scripts.

Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> • UNIX • Windows (except for Kerberos authentication)
Allows for	<ul style="list-style-type: none"> • Authentication of LSF users, hosts, and daemons • Authentication of any number of LSF users
Not required for	<ul style="list-style-type: none"> • Authorization of users based on account permissions
Dependencies	<ul style="list-style-type: none"> • UNIX and Windows user accounts must be valid on all hosts in the cluster, or the correct type of account mapping must be enabled: <ul style="list-style-type: none"> • For a mixed UNIX/Windows cluster, UNIX/Windows user account mapping must be enabled • For a cluster with a non-uniform user name space, between-host account mapping must be enabled • For a MultiCluster environment with a non-uniform user name space, cross-cluster user account mapping must be enabled • User accounts must have the correct permissions to successfully run jobs. • The owner of <code>lsf.sudoers</code> on Windows must be <code>Administrators</code>.

Configuration to enable external authentication

During LSF installation:

- The parameter LSF_AUTH in lsf.conf is set to eauth, which enables external authentication
- A default eauth executable is installed in the directory specified by the parameter LSF_SERVERDIR in lsf.conf

The default executable provides an example of how the eauth protocol works. You should write your own eauth executable to meet the security requirements of your cluster.

Configuration file	Parameter and syntax	Default behavior
lsf.conf	LSF_AUTH=eauth	<ul style="list-style-type: none">• Enables external authentication
	LSF_AUTH_DAEMONS=y Y	<ul style="list-style-type: none">• Enables daemon authentication when external authentication is enabled <div>Note: By default, daemon authentication is not enabled. If you enable daemon authentication and want to turn it off later, you must comment out or delete the parameter LSF_AUTH_DAEMONS.</div>

External authentication behavior

The following example illustrates how a customized `eauth` executable can provide external authentication of users, hosts, and daemons. In this example, the `eauth` executable has been customized so that corresponding instances of `eauth -c` and `eauth -s` obtain user, host, and daemon credentials from a file that serves as the external security system. The `eauth` executable can also be customized to obtain credentials from an operating system or from an authentication protocol such as Kerberos.

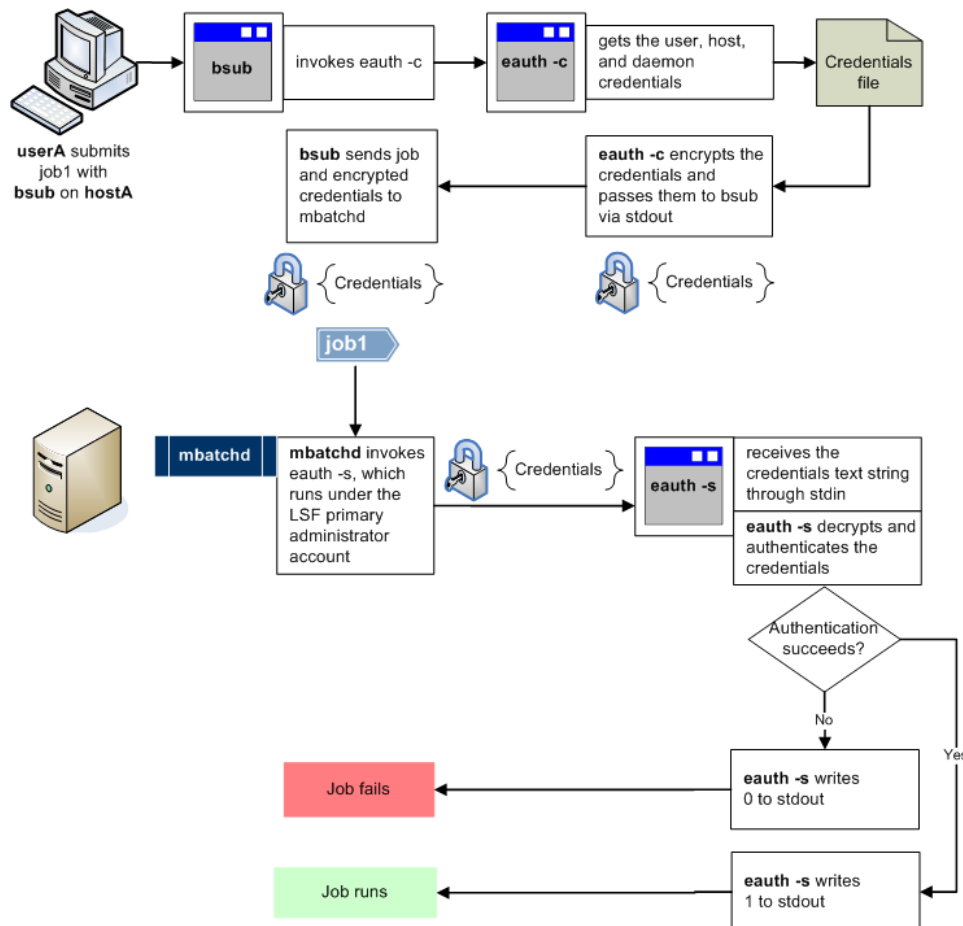


Figure 16: Example of external authentication

Authentication failure

When external authentication is enabled, the message

User permission denied

indicates that the `eauth` executable failed to authenticate the user's credentials.

Security

External authentication—and any other LSF authentication method—depends on the security of the root account on all hosts within the cluster. Limit access to the root account to prevent unauthorized use of your cluster.

Configuration to modify external authentication

You can modify external authentication behavior by writing your own `eauth` executable. There are also configuration parameters that modify various aspects of external authentication behavior by:

- Increasing security through the use of an external encryption key (recommended)
- Specifying a trusted user account under which the `eauth` executable runs (UNIX and Linux only)

You can also choose Kerberos authentication to provide a secure data exchange during LSF user and daemon authentication and to forward credentials to a remote host for use during job execution.

Configuration to modify security

File	Parameter and syntax	Descriptions
<code>lsf.sudoers</code>	<code>LSF_EAUTH_KEY=key</code>	<ul style="list-style-type: none"> • The <code>eauth</code> executable uses the external encryption key that you define to encrypt and decrypt the credentials. • The key must contain at least six characters and must use only printable characters. • For UNIX, you must edit the <code>lsf.sudoers</code> file on all hosts within the cluster and specify the same encryption key. You must also configure <code>eauth</code> as <code>setuid</code> to <code>root</code> so that <code>eauth</code> can read the <code>lsf.sudoers</code> file and obtain the value of <code>LSF_EAUTH_KEY</code>. • For Windows, you must edit the shared <code>lsf.sudoers</code> file.

Configuration to specify the eauth user account

On UNIX hosts, the `eauth` executable runs under the account of the primary LSF administrator. You can modify this behavior by specifying a different trusted user account. For Windows hosts, you do not need to modify the default behavior because `eauth` runs under the service account, which is always a trusted, secure account.

File	Parameter and syntax	Description
<code>lsf.sudoers</code>	<code>LSF_EAUTH_USER=user_name</code>	<ul style="list-style-type: none"> • UNIX only • The <code>eauth</code> executable runs under the account of the specified user rather than the account of the LSF primary administrator • You must edit the <code>lsf.sudoers</code> file on all hosts within the cluster and specify the same user name

Configuration to enable Kerberos authentication

To install and configure Kerberos authentication, refer to the information included with your Kerberos integration package provided by Platform Computing Inc..

Restriction:

Kerberos authentication is supported only for UNIX and Linux hosts, and only on the following operating systems:

- AIX 4
- Alpha 4.x
- IRIX 6.5
- Linux 2.x
- Solaris 2.x

Configuration file	Parameter and syntax	Behavior
lsf.conf	LSF_AUTH=eauth	<ul style="list-style-type: none"> • Enables external authentication
	LSF_AUTH_DAEMONS=y Y	<ul style="list-style-type: none"> • Enables daemon authentication when external authentication is enabled
	LSF_DAEMON_WRAP=y Y	<ul style="list-style-type: none"> • Required for Kerberos authentication • mbat chd, sbat chd, and RES run the executable LSF_SERVERDIR/daemons.wrap
lsf.sudoers	LSF_EAUTH_USER=root	<ul style="list-style-type: none"> • for Kerberos authentication, the eauth executable must run under the root account • You must edit the lsf.sudoers file on all hosts within the cluster and specify the same user name
	LSF_LOAD_PLUGINS=y Y	<ul style="list-style-type: none"> • Required for Kerberos authentication when plug-ins are used instead of the daemon wrapper script • LSF loads plug-ins from the directory LSB_LIBDIR
	LSF_EEXEC_USER=root	<ul style="list-style-type: none"> • Required for Kerberos authentication. The parameter LSF_DAEMON_WRAP must also be set to y or Y. • The eexec executable provided with the Kerberos integration runs under the root account

External authentication commands

Commands for submission

Command	Description
All LSF commands	<ul style="list-style-type: none"> If the parameter <code>LSF_AUTH=eauth</code> in the file <code>lsf.conf</code>, LSF daemons authenticate users and hosts—as configured in the <code>eauth</code> executable—before executing an LSF command If external authentication is enabled and the parameter <code>LSF_AUTH_DAEMONS=Y</code> in the file <code>lsf.conf</code>, LSF daemons authenticate each other as configured in the <code>eauth</code> executable

Commands to monitor

Not applicable: There are no commands to monitor the behavior of this feature.

Commands to control

Not applicable: There are no commands to control the behavior of this feature.

Commands to display configuration

Command	Description
<code>badmi n showconf</code>	<ul style="list-style-type: none"> Displays all configured parameters and their values set in <code>lsf.conf</code> or <code>ego.conf</code> that affect <code>mbatchd</code> and <code>sbatchd</code>. Use a text editor to view other parameters in the <code>lsf.conf</code> or <code>ego.conf</code> configuration files. In a MultiCluster environment, displays the parameters of daemons on the local cluster.

Use a text editor to view the `lsf.sudoers` configuration file.

Job Email and Job File Spooling

Email notification

When a batch job completes or exits, LSF by default sends a job report by email to the submitting user account. The report includes the following information:

- Standard output (`stdout`) of the job
- Standard error (`stderr`) of the job
- LSF job information such as CPU, process and memory usage

The output from `stdout` and `stderr` are merged together in the order printed, as if the job was run interactively. The default standard input (`stdin`) file is the null device. The null device on UNIX is `/dev/null`.

bsub mail options

-B

Sends email to the job submitter when the job is dispatched and begins running. The default destination for email is defined by `LSB_MAILTO` in `lsf.conf`.

-u user_name

If you want mail sent to another user, use the `-u user_name` option to the `bsub` command. Mail associated with the job will be sent to the named user instead of to the submitting user account.

-N

If you want to separate the job report information from the job output, use the `-N` option to specify that the job report information should be sent by email.

Output and error file options (-o output_file, -e error_file, -oo output_file, and -eo error_file)

The output file created by the `-o` and `-oo` options to the `bsub` command normally contains job report information as well as the job output. This information includes the submitting user and host, the execution host, the CPU time (user plus system time) used by the job, and the exit status.

If you specify a `-o output_file` or `-oo output_file` option and do not specify a `-e error_file` or `-eo error_file` option, the standard output and standard error are merged and stored in `output_file`. You can also specify the standard input file if the job needs to read input from `stdin`.

Note:

The file path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory, file name, and expanded values for `%J` (*job_ID*) and `%I` (*index_ID*).

The output files specified by the output and error file options are created on the execution host.

Disable job email

- specify `stdout` and `stderr` as the files for the output and error options (`-o`, `-oo`, `-e`, and `-eo`).

For example, the following command directs `stderr` and `stdout` to file named `/tmp/job_out`, and no email is sent.

```
bsub -o /tmp/job_out sleep 5
```

- On UNIX, for no job output or email specify `/dev/null` as the output file:

```
bsub -o /dev/null sleep 5
```

Example

The following example submits `myjob` to the night queue:

```
bsub -q night -i job_in -o job_out -e job_err myjob
```

The job reads its input from file `job_in`. Standard output is stored in file `job_out`, and standard error is stored in file `job_err`.

Size of job email

Some batch jobs can create large amounts of output. To prevent large job output files from interfering with your mail system, you can use the `LSB_MAILSIZE_LIMIT` parameter in `lsf.conf` to limit the size of the email containing the job output information.

By default, `LSB_MAILSIZE_LIMIT` is not enabled—no limit is set on size of batch job output email.

If the size of the job output email exceeds `LSB_MAILSIZE_LIMIT`, the output is saved to a file under `JOB_SPOOL_DIR`, or the default job output directory if `JOB_SPOOL_DIR` is undefined. The email informs users where the job output is located.

If the `-o` or `-oo` option of `bsub` is used, the size of the job output is not checked against `LSB_MAILSIZE_LIMIT`.

LSB_MAILSIZE environment variable

LSF sets `LSB_MAILSIZE` to the approximate size in KB of the email containing job output information, allowing a custom mail program to intercept output that is larger than desired. If you use the `LSB_MAILPROG` parameter to specify the custom mail program that can make use of the `LSB_MAILSIZE` environment variable, it is not necessary to configure `LSB_MAILSIZE_LIMIT`.

`LSB_MAILSIZE` is not recognized by the LSF default mail program. To prevent large job output files from interfering with your mail system, use `LSB_MAILSIZE_LIMIT` to explicitly set the maximum size in KB of the email containing the job information.

LSB_MAILSIZE values

The `LSB_MAILSIZE` environment variable can take the following values:

- A positive integer: if the output is being sent by email, `LSB_MAILSIZE` is set to the estimated mail size in KB.
- -1 :if the output fails or cannot be read, `LSB_MAILSIZE` is set to -1 and the output is sent by email using `LSB_MAILPROG` if specified in `lsf.conf`.
- Undefined: If you use the output or error options (`-o`, `-oo`, `-e`, or `-eo`) of `bsub`, the output is redirected to an output file. Because the output is not sent by email in this case, `LSB_MAILSIZE` is not used and `LSB_MAILPROG` is not called.

If the `-N` option is used with the output or error options of `bsub`, `LSB_MAILSIZE` is not set.

Directory for job output

The output and error options (-o, -oo, -e, and -eo) of the bsub and bmod commands can accept a file name or directory path. LSF creates the standard output and standard error files in this directory. If you specify only a directory path, job output and error files are created with unique names based on the job ID so that you can use a single directory for all job output, rather than having to create separate output directories for each job.

Note:

The directory path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows.

Specify a directory for job output

- Make the final character in the path a slash (/) on UNIX, or a double backslash (\\) on Windows.

If you omit the trailing slash or backslash characters, LSF treats the specification as a file name.

If the specified directory does not exist, LSF creates it on the execution host when it creates the standard error and standard output files.

By default, the output files have the following format:

Standard output: `output_directory/job_ID.out`

Standard error: `error_directory/job_ID.err`

Example

The following command creates the directory `/usr/share/l sf_out` if it does not exist, and creates the standard output file `job_ID.out` in this directory when the job completes:

```
bsub -o /usr/share/l sf_out/ myjob
```

The following command creates the directory `C: \l sf\work\l sf_err` if it does not exist, and creates the standard error file `job_ID.err` in this directory when the job completes:

```
bsub -e C:\l sf\work\l sf_err\ myjob
```

File spooling for job input, output, and command files

LSF enables *spooling* of job input, output, and command files by creating directories and files for buffering input and output for a job. LSF removes these files when the job completes.

You can make use of file spooling when submitting jobs with the `-is` and `-Zs` options to `bsub`. Use similar options in `bmod` to modify or cancel the spool file specification for the job. Use the file spooling options if you need to modify or remove the original job input or command files before the job completes. Removing or modifying the original input file does not affect the submitted job.

Note:

The file path for spooling job input, output, and command files can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory, file name, and expanded values for `%J` (*job_ID*) and `%I` (*index_ID*).

File spooling is not supported across MultiClusters.

Specify job input file

- Use `bsub -i input_file` and `bsub -is input_file` to get the standard input for the job from the file path name specified by *input_file*.

input_file can be an absolute path or a relative path to the current working directory, and can be any type of file though it is typically a shell script text file.

The `-is` option spools the input file to the directory specified by the `JOB_SPOOL_DIR` parameter in `lsb.params`, and uses the spooled file as the input file for the job.

Note:

With `bsub -i` you can use the special characters `%J` and `%I` in the name of the input file. `%J` is replaced by the job ID. `%I` is replaced by the index of the job in the array, if the job is a member of an array, otherwise by 0 (zero).

- Use `bsub -is` to change the original input file before the job completes. Removing or modifying the original input file does not affect the submitted job.

LSF first checks the execution host to see if the input file exists, and if so uses this file as the input file for the job. Otherwise LSF attempts to copy the file from the submission host to the execution host. For the file copy to be successful, you must allow remote copy (`rcp`) access, or you must submit the job from a server host where RES is running. The file is copied from the submission host to a temporary file in the directory specified by the `JOB_SPOOL_DIR` parameter in `lsb.params`, or your `$HOME/.lsbatch` directory on the execution host. LSF removes this file when the job completes.

Change job input file

- Use `bmod -i input_file` and `bmod -is input_file` to specify a new job input file.
- Use `bmod -in` and `bmod -isn` to cancel the last job input file modification made with either `-i` or `-is`.

Job spooling directory (JOB_SPOOL_DIR)

The JOB_SPOOL_DIR in lsb.params sets the job spooling directory. If defined, JOB_SPOOL_DIR should be:

- A shared directory accessible from the master host and the submission host.
- A valid path up to a maximum length up to 4094 characters on UNIX and Linux or up to 255 characters for Windows.
- Readable and writable by the job submission user.

Except for bsub -i s and bsub -Zs, if JOB_SPOOL_DIR is not accessible or does not exist, output is spooled to the default job output directory .lsbat ch.

For bsub -i s and bsub -Zs, JOB_SPOOL_DIR must be readable and writable by the job submission user. If the specified directory is not accessible or does not exist, bsub -i s and bsub -Zs cannot write to the default directory and the job will fail.

JOB_SPOOL_DIR specified:

- The job input file for bsub -i s is spooled to JOB_SPOOL_DIR/l sf_i ndi r. If the l sf_i ndi r directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.
- The job command file for bsub -Zs is spooled to JOB_SPOOL_DIR/l sf_cmddi r. If the l sf_cmddi r directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.

JOB_SPOOL_DIR not specified:

- The job input file for bsub -i s is spooled to LSB_SHAREDIR/*cluster_name*/l sf_i ndi r. If the l sf_i ndi r directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.
- The job command file for bsub -Zs is spooled to LSB_SHAREDIR/*cluster_name*/l sf_cmddi r. If the l sf_cmddi r directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.

If you want to use job file spooling without specifying JOB_SPOOL_DIR, the LSB_SHAREDIR/*cluster_name* directory must be readable and writable by all the job submission users. If your site does not permit this, you must manually create l sf_i ndi r and l sf_cmddi r directories under LSB_SHAREDIR/*cluster_name* that are readable and writable by all job submission users.

Specify a job command file (bsub -Zs)

- Use `bsub -Zs` to spool a job command file to the directory specified by the `JOB_SPOOL_DIR` parameter in `lsb.params`.

LSF uses the spooled file as the command file for the job.

Note:

The `bsub -Zs` option is not supported for embedded job commands because LSF is unable to determine the first command to be spooled in an embedded job command.

- Use `bmod -Zs` to change the command file after the job has been submitted.

Changing the original input file does not affect the submitted job.

- Use `bmod -Zsn` to cancel the last spooled command file and use the original spooled file.
- Use `bmod -Z` to modify a command submitted without spooling

Non-Shared File Systems

About directories and files

LSF is designed for networks where all hosts have shared file systems, and files have the same names on all hosts.

LSF includes support for copying user data to the execution host before running a batch job, and for copying results back after the job executes.

In networks where the file systems are not shared, this can be used to give remote jobs access to local data.

Supported file systems

UNIX

On UNIX systems, LSF supports the following shared file systems:

- Network File System (NFS). NFS file systems can be mounted permanently or on demand using `autofs`.
- Andrew File System (AFS)
- Distributed File System (DCE/DFS)

Windows

On Windows, directories containing LSF files can be shared among hosts from a Windows server machine.

Non-shared directories and files

LSF is usually used in networks with shared file space. When shared file space is not available, LSF can copy needed files to the execution host before running the job, and copy result files back to the submission host after the job completes.

Some networks do not share files between hosts. LSF can still be used on these networks, with reduced fault tolerance.

Use Platform LSF with non-shared file systems

1. Follow the complete installation procedure on every host to install all the binaries, man pages, and configuration files.
2. Update the configuration files on all hosts so that they contain the complete cluster configuration.

Configuration files must be the same on all hosts.

3. Choose one host to act as the LSF master host.
 - a) Install LSF configuration files and working directories on this host
 - b) Edit `lsf.cluster.cluster_name` and list this host first.
 - c) Use the parameter `LSF_MASTER_LIST` in `lsf.conf` to set master host candidates.

For Windows password authentication in a non-shared file system environment, you must define the parameter `LSF_MASTER_LIST` in `lsf.conf` so that jobs will run with correct permissions. If you do not define this parameter, LSF assumes that the cluster uses a shared file system environment.

Note:

Fault tolerance can be introduced by choosing more than one host as a possible master host, and using NFS to mount the LSF working directory on only these hosts. All the possible master hosts must be listed first in `lsf.cluster.cluster_name`. As long as one of these hosts is available, LSF continues to operate.

Remote file access with non-shared file space

LSF is usually used in networks with shared file space. When shared file space is not available, use the `bsub -f` command to have LSF copy needed files to the execution host before running the job, and copy result files back to the submission host after the job completes.

LSF attempts to run a job in the directory where the `bsub` command was invoked. If the execution directory is under the user's home directory, `sbat chd` looks for the path relative to the user's home directory. This handles some common configurations, such as cross-mounting user home directories with the `/net automount` option.

If the directory is not available on the execution host, the job is run in `/tmp`. Any files created by the batch job, including the standard output and error files created by the `-o` and `-e` options to `bsub`, are left on the execution host.

LSF provides support for moving user data from the submission host to the execution host before executing a batch job, and from the execution host back to the submitting host after the job completes. The file operations are specified with the `-f` option to `bsub`.

LSF uses the `lsrcp` command to transfer files. `lsrcp` contacts RES on the remote host to perform file transfer. If RES is not available, the UNIX `rcp` command is used or, if it is set, the command and options specified by setting `LSF_REMOTE_COPY_COMMAND` in `lsf.conf`.

Copy files from the submission host to execution host

1. Use `bsub -f "[local_file operator [remote_file]]"`

To specify multiple files, repeat the `-f` option.

local_file is the file on the submission host, *remote_file* is the file on the execution host.

local_file and *remote_file* can be absolute or relative file path names. You must specify at least one file name. When the file *remote_file* is not specified, it is assumed to be the same as *local_file*. Including *local_file* without the operator results in a syntax error.

Valid values for *operator* are:

>

local_file on the submission host is copied to *remote_file* on the execution host before job execution. *remote_file* is overwritten if it exists.

<

remote_file on the execution host is copied to *local_file* on the submission host after the job completes. *local_file* is overwritten if it exists.

<<

remote_file is appended to *local_file* after the job completes. *local_file* is created if it does not exist.

><, <>

Equivalent to performing the > and then the < operation. The file *local_file* is copied to *remote_file* before the job executes, and *remote_file* is copied back, overwriting *local_file*, after the job completes. <> is the same as ><

LSF tries to change the directory to the same path name as the directory where the `bsub` command was run. If this directory does not exist, the job is run in your home directory on the execution host.

Note:

Specify *remote_file* as a file name with no path when running in non-shared file systems; this places the file in the job's current working directory on the execution host. This way the job will work correctly even if the directory where the `bsub` command is run does not exist on the execution host.

Examples

To submit my job to LSF, with input taken from the file `/data/data3` and the output copied back to `/data/out3`, run the command:

```
bsub -f "/data/data3 > data3" -f "/data/out3 < out3" myjob data3 out3
```

To run the job `batch_update`, which updates the `batch_data` file in place, you need to copy the file to the execution host before the job runs and copy it back after the job completes:

```
bsub -f "batch_data <>" batch_update batch_data
```

Specify input file

1. Use `bsub -i input_file`.

If the input file specified is not found on the execution host it is copied from the submission host using the LSF remote file access facility and is removed from the execution host after the job finishes.

Copy output files back to the submission host

The output files specified with the `bsub -o` and `bsub -e` are created on the execution host, and are not copied back to the submission host by default.

1. Use the remote file access facility to copy these files back to the submission host if they are not on a shared file system.

For example, the following command stores the job output in the `job_out` file and copies the file back to the submission host:

```
bsub -o job_out -f "job_out <" myjob
```

File transfer mechanism (lsrscp)

The LSF remote file access mechanism (`bsub -f`) uses `lsrscp` to process the file transfer. The `lsrscp` command tries to connect to RES on the submission host to handle the file transfer.

Limitations to lsrscp

Because LSF client hosts do not run RES, jobs that are submitted from client hosts should only specify `bsub -f` if `rcp` is allowed. You must set up the permissions for `rcp` if account mapping is used.

File transfer using `lsrscp` is not supported in the following contexts:

- If LSF account mapping is used; `lsrscp` fails when running under a different user account
- LSF client hosts do not run RES, so `lsrscp` cannot contact RES on the submission host

See the *Authentication and Authorization* chapter for more information.

Workarounds

In these situations, use the following workarounds:

rcp and scp on UNIX

If `lsrscp` cannot contact RES on the submission host, it attempts to use `rcp` to copy the file. You must set up the `/etc/hosts.equiv` or `HOME/.rhosts` file in order to use `rcp`.

If `LSF_REMOTE_COPY_CMD` is set in `lsf.conf`, `lsrscp` uses that command instead of `rcp` to copy the file. You can specify `rcp`, `scp`, or a custom copy command and options in this parameter.

See the `rcp(1)` and `rsh(1)` man pages for more information on using the `rcp` command.

Custom file transfer mechanism

You can replace `lsrscp` with your own file transfer mechanism as long as it supports the same syntax as `lsrscp`. This might be done to take advantage of a faster interconnection network, or to overcome limitations with the existing `lsrscp`. `sbatchd` looks for the `lsrscp` executable in the `LSF_BINDIR` directory as specified in the `lsf.conf` file.

Sample script for file transfer

```
#!/bin/sh
# lsrscp_fallback_cmd - Sample shell script to perform file copy between hosts.
#
# This script can be used by lsrscp by configuring
# LSF_REMOTE_COPY_CMD in lsf.conf.
# We recommend placing this file in $LSF_BINDIR.
#
SHELL_NAME="lsrscp_fallback_cmd"
RCP="rcp"
SCP="scp"
SOURCE=$1
DESTINATION=$2
ENOENT=2
EACCES=13
ENOSPC=28
noFallback()
{
    echo "Do not try fallback commands"
    EXITCODE=0
}
tryRcpScpInOrder()
{
```



```

    echo "Trying rcp..."
    SRCP $SOURCE $DESTINATION
    EXITCODE=$?
    #The exit code of rcp only indicates whether a connection was made successfully or not.
    #An error will be returned if the hostname is not found
    #or the host refuses the connection. Otherwise, rcp is always successful.
    #So, we only try scp when the exit code is not zero. For other cases, we do nothing,
    #but the error message of rcp can be seen from terminal
    if [ $EXITCODE -ne 0 ]; then
        echo "Trying scp..."
        #If you don't configure SSH authorization and want users to input password,
        #remove the scp option of "-B -o 'strictHostKeyChecking no'"
        SSCP -B -o 'strictHostKeyChecking no' $SOURCE $DESTINATION
        EXITCODE=$?
    fi
}
tryScp()
{
    echo "Trying scp..."
    #If you don't configure SSH authorization and want users to input password,
    #remove the scp option of "-B -o 'strictHostKeyChecking no'"
    SSCP -B -o 'strictHostKeyChecking no' $SOURCE $DESTINATION
    EXITCODE=$?
}
tryRcp()
{
    echo "Trying rcp..."
    SRCP $SOURCE $DESTINATION
    EXITCODE=$?
}
usage()
{
    echo "Usage: $SHELL_NAME source destination"
}
if [ $# -ne 2 ]; then
    usage
    exit 2
fi
case $LSF_LSRCP_ERRNO in
    $ENOENT)
        noFallback
        ;;
    $EACCES)
        noFallback
        ;;
    $ENOSPC)
        noFallback
        ;;
    *)
        tryRcpScpInOrder
        ;;
esac
exit $EXITCODE

```


Error and Event Logging

System directories and log files

LSF uses directories for temporary work files, log files and transaction files and spooling.

LSF keeps track of all jobs in the system by maintaining a transaction log in the work subtree. The LSF log files are found in the directory `LSB_SHAREDIR/cluster_name/logdir`.

The following files maintain the state of the LSF system:

lsb.events

LSF uses the `lsb.events` file to keep track of the state of all jobs. Each job is a transaction from job submission to job completion. LSF system keeps track of everything associated with the job in the `lsb.events` file.

lsb.events.n

The events file is automatically trimmed and old job events are stored in `lsb.event.n` files. When `mbatchd` starts, it refers only to the `lsb.events` file, not the `lsb.event.n` files. The `bhist` command can refer to these files.

Job script files in the info directory

When a user issues a `bsub` command from a shell prompt, LSF collects all of the commands issued on the `bsub` line and spools the data to `mbatchd`, which saves the `bsub` command script in the info directory (or in one of its subdirectories if `MAX_INFO_DIRS` is defined in `lsb.params`) for use at dispatch time or if the job is rerun. The info directory is managed by LSF and should not be modified by anyone.

Log directory permissions and ownership

Ensure that the permissions on the `LSF_LOGDIR` directory to be writable by `root`. The LSF administrator must own `LSF_LOGDIR`.

UNICOS accounting

In Cray UNICOS environments, LSF writes to the Network Queuing System (NQS) accounting data file, `nqacct`, on the execution host. This lets you track LSF jobs and other jobs together, through NQS.

IRIX Comprehensive System Accounting (CSA)

The IRIX 6.5.9 Comprehensive System Accounting facility (CSA) writes an accounting record for each process in the `pacct` file, which is usually located in the `/var/adm/acct/day` directory. IRIX system administrators then use the `csabuild` command to organize and present the records on a job by job basis.

The `LSF_ENABLE_CSA` parameter in `lsf.conf` enables LSF to write job events to the `pacct` file for processing through CSA. For LSF job accounting, records are written to `pacct` at the start and end of each LSF job.

See the *LSF Configuration Reference* for more information about the `LSF_ENABLE_CSA` parameter.

See the IRIX 6.5.9 resource administration documentation for information about CSA.

Log levels and descriptions

Number	Level	Description
0	LOG_EMERG	Log only those messages in which the system is unusable.
1	LOG_ALERT	Log only those messages for which action must be taken immediately.
2	LOG_CRIT	Log only those messages that are critical.
3	LOG_ERR	Log only those messages that indicate error conditions.
4	LOG_WARNING	Log only those messages that are warnings or more serious messages. This is the default level of debug information.
5	LOG_NOTICE	Log those messages that indicate normal but significant conditions or warnings and more serious messages.
6	LOG_INFO	Log all informational messages and more serious messages.
7	LOG_DEBUG	Log all debug-level messages.
8	LOG_TRACE	Log all available messages.

Manage error logs

Error logs maintain important information about LSF operations. When you see any abnormal behavior in LSF, you should first check the appropriate error logs to find out the cause of the problem.

LSF log files grow over time. These files should occasionally be cleared, either by hand or using automatic scripts.

Daemon error logs

LSF log files are reopened each time a message is logged, so if you rename or remove a daemon log file, the daemons will automatically create a new log file.

The LSF daemons log messages when they detect problems or unusual situations.

The daemons can be configured to put these messages into files.

The error log file names for the LSF system daemons are:

- `res.log`, *host_name*
- `sbatchd.log`, *host_name*
- `mbatchd.log`, *host_name*
- `mbschd.log`, *host_name*

LSF daemons log error messages in different levels so that you can choose to log all messages, or only log messages that are deemed critical. Message logging for LSF daemons (except LIM) is controlled by the parameter `LSF_LOG_MASK` in `lsf.conf`. Possible values for this parameter can be any log priority symbol that is defined in `/usr/include/sys/syslog.h`. The default value for `LSF_LOG_MASK` is `LOG_WARNING`.

Important:

`LSF_LOG_MASK` in `lsf.conf` no longer specifies LIM logging level in LSF Version 8. For LIM, you must use `EGO_LOG_MASK` in `ego.conf` to control message logging for LIM. The default value for `EGO_LOG_MASK` is `LOG_WARNING`.

Set the log files owner

You must be the cluster administrator. The performance monitoring (perfmon) metrics must be enabled or you must set `LC_PERFM` to debug.

You can set the log files owner for the LSF daemons (not including the `mbschd`). The default owner is the LSF Administrator.

Restriction:

Applies to UNIX hosts only.

Restriction:

This change only takes effect for daemons that are running as root.

1. Edit `lsf.conf` and add the parameter `LSF_LOGFILE_OWNER`.
2. Specify a user account name to set the owner of the log files.
3. Shut down the LSF daemon or daemons you want to set the log file owner for.

Run `lsfshut` down on the host.

4. Delete or move any existing log files.

Important:

If you do not clear out the existing log files, the file ownership does not change.

5. Restart the LSF daemons you shut down.

Run `lsfstart` up on the host.

View the number of file descriptors remaining

The performance monitoring (perfmom) metrics must be enabled or you must set `LC_PERFM` to debug.

The `mbatchd` daemon can log a large number of files in a short period when you submit a large number of jobs to LSF. You can view the remaining file descriptors at any time.

Restriction:

Applies to UNIX hosts only.

1. Run `badmi n perfmom view`.

The free, used, and total amount of file descriptors display.

On AIX5, 64-bit hosts, if the file descriptor limit has never been changed, the maximum value displays: 9223372036854775797.

Locate Error logs

- Optionally, set the `LSF_LOGDIR` parameter in `lsf.conf`.
Error messages from LSF servers are logged to files in this directory.
- If `LSF_LOGDIR` is defined, but the daemons cannot write to files there, the error log files are created in `/tmp`.
- If `LSF_LOGDIR` is not defined, errors are logged to the system error logs (`syslog`) using the `LOG_DAEMON` facility.

`syslog` messages are highly configurable, and the default configuration varies from system to system. Start by looking for the file `/etc/syslog.conf`, and read the man pages for `syslog(3)` and `syslogd(1)`. If the error log is managed by `syslog`, it is probably being automatically cleared.

- If LSF daemons cannot find `lsf.conf` when they start, they will not find the definition of `LSF_LOGDIR`. In this case, error messages go to `syslog`. If you cannot find any error messages in the log files, they are likely in the `syslog`.

System event log

The LSF daemons keep an event log in the `l sb. event s` file. The `mbat chd` daemon uses this information to recover from server failures, host reboots, and `mbat chd` restarts. The `l sb. event s` file is also used by the `bhi st` command to display detailed information about the execution history of batch jobs, and by the `badmi n` command to display the operational history of hosts, queues, and daemons.

By default, `mbat chd` automatically backs up and rewrites the `l sb. event s` file after every 1000 batch job completions. This value is controlled by the `MAX_JOB_NUM` parameter in the `l sb. params` file. The old `l sb. event s` file is moved to `l sb. event s. 1`, and each old `l sb. event s. n` file is moved to `l sb. event s. n+1`. LSF never deletes these files. If disk storage is a concern, the LSF administrator should arrange to archive or remove old `l sb. event s. n` files periodically.

Caution:

Do not remove or modify the current `l sb. event s` file. Removing or modifying the `l sb. event s` file could cause batch jobs to be lost.

Duplicate logging of event logs

To recover from server failures, host reboots, or `mbat chd` restarts, LSF uses information stored in `lsb.events`. To improve the reliability of LSF, you can configure LSF to maintain copies of these logs, to use as a backup.

If the host that contains the primary copy of the logs fails, LSF will continue to operate using the duplicate logs. When the host recovers, LSF uses the duplicate logs to update the primary copies.

How duplicate logging works

By default, the event log is located in `LSB_SHAREDIR`. Typically, `LSB_SHAREDIR` resides on a reliable file server that also contains other critical applications necessary for running jobs, so if that host becomes unavailable, the subsequent failure of LSF is a secondary issue. `LSB_SHAREDIR` must be accessible from all potential LSF master hosts.

When you configure duplicate logging, the duplicates are kept on the file server, and the primary event logs are stored on the first master host. In other words, `LSB_LOCALDIR` is used to store the primary copy of the batch state information, and the contents of `LSB_LOCALDIR` are copied to a replica in `LSB_SHAREDIR`, which resides on a central file server. This has the following effects:

- Creates backup copies of `lsb.events`
- Reduces the load on the central file server
- Increases the load on the LSF master host

Failure of file server

If the file server containing `LSB_SHAREDIR` goes down, LSF continues to process jobs. Client commands such as `bhist`, which directly read `LSB_SHAREDIR` will not work.

When the file server recovers, the current log files are replicated to `LSB_SHAREDIR`.

Failure of first master host

If the first master host fails, the primary copies of the files (in `LSB_LOCALDIR`) become unavailable. Then, a new master host is selected. The new master host uses the duplicate files (in `LSB_SHAREDIR`) to restore its state and to log future events. There is no duplication by the second or any subsequent LSF master hosts.

When the first master host becomes available after a failure, it will update the primary copies of the files (in `LSB_LOCALDIR`) from the duplicates (in `LSB_SHAREDIR`) and continue operations as before.

If the first master host does not recover, LSF will continue to use the files in `LSB_SHAREDIR`, but there is no more duplication of the log files.

Simultaneous failure of both hosts

If the master host containing `LSB_LOCALDIR` and the file server containing `LSB_SHAREDIR` both fail simultaneously, LSF will be unavailable.

Network partitioning

We assume that Network partitioning does not cause a cluster to split into two independent clusters, each simultaneously running `mbat chd`.

This may happen given certain network topologies and failure modes. For example, connectivity is lost between the first master, M1, and both the file server and the secondary master, M2. Both M1 and M2 will run `mbat chd` service with M1 logging events to `LSB_LOCALDIR` and M2 logging to

LSB_SHAREDIR. When connectivity is restored, the changes made by M2 to LSB_SHAREDIR will be lost when M1 updates LSB_SHAREDIR from its copy in LSB_LOCALDIR.

The archived event files are only available on LSB_LOCALDIR, so in the case of network partitioning, commands such as `bhist` cannot access these files. As a precaution, you should periodically copy the archived files from LSB_LOCALDIR to LSB_SHAREDIR.

Automatic archives

Archived event logs, `lsb.events.n`, are not replicated to LSB_SHAREDIR. If LSF starts a new event log while the file server containing LSB_SHAREDIR is down, you might notice a gap in the historical data in LSB_SHAREDIR.

Configure duplicate logging

1. Edit `lsf.conf` and set `LSB_LOCALDIR` to a local directory that exists only on the first master host.
This directory is used to store the primary copies of `lsb.events`.
2. Use the commands `lsadmin reconfig` and `lsadmin mbdrestart` to make the changes take effect.

Set an event update interval

If NFS traffic is high you can reduce network traffic by changing the update interval.

- Use `EVENT_UPDATE_INTERVAL` in `lsb.params` to specify how often to back up the data and synchronize the `LSB_SHAREDIR` and `LSB_LOCALDIR` directories.

The directories are always synchronized when data is logged to the files, or when `mbatchd` is started on the first LSF master host.

LSF job termination reason logging

When a job finishes, LSF reports the last job termination action it took against the job and logs it into `lsb. acct`.

If a running job exits because of node failure, LSF sets the correct exit information in `lsb. acct`, `lsb. events`, and the job output file. Jobs terminated by a signal from LSF, the operating system, or an application have the signal logged as the LSF exit code. Exit codes are not the same as the termination actions.

View logged job exit information (bacct -l)

1. Use `bacct -l` to view job exit information logged to `lsb. acct`:

bacct -l 7265

Accounting information about jobs that are:

- submitted by all users.
- accounted on all projects.
- completed normally or exited
- executed on all hosts.
- submitted to all queues.
- accounted on all service classes.

Job <7265>, User <lsfadmin>, Project <default>, Status <EXIT>, Queue <normal>,
Command <sleep 100000>

Thu Sep 16 15:22:09 2009: Submitted from host <hostA>, CWD <\$HOME>;

Thu Sep 16 15:22:20 2009: Dispatched to 4 Hosts/Processors <4*hostA>;

Thu Sep 16 15:22:20 2009: slurm_id=21793;ncpus=4;slurm_alloc=n[13-14];

Thu Sep 16 15:23:21 2009: Completed <exit>; TERM_RUNLIMIT: job killed after reaching LSF run time limit.

Accounting information about this job:

Share group charged </lsfadmin>

CPU_T	WAIT	TURNAROUND	STATUS	HOG_FACTOR	MEM	SWAP
0.04	11	72	exit	0.0006	OK	OK

SUMMARY: (time unit: second)

Total number of done jobs:	0	Total number of exited jobs:	1
Total CPU time consumed:	0.0	Average CPU time consumed:	0.0
Maximum CPU time of a job:	0.0	Minimum CPU time of a job:	0.0
Total wait time in queues:	11.0		
Average wait time in queue:	11.0		
Maximum wait time in queue:	11.0	Minimum wait time in queue:	11.0
Average turnaround time:	72 (seconds/job)		
Maximum turnaround time:	72	Minimum turnaround time:	72
Average hog factor of a job:	0.00 (cpu time / turnaround time)		
Maximum hog factor of a job:	0.00	Minimum hog factor of a job:	0.00

View recent job exit information (bjobs -l)

1. Use `bjobs -l` to view job exit information for recent jobs:

bjobs -l 7265

Job <642>, User <user12>, Project <default>, Status <EXIT>, Queue <normal>, Command <perl -e "while (1){}">

Fri Feb 27 15:06:35 2010: Submitted from host <hostabc>, CWD <\$HOME/home/lsf/lsf8.0.slt/8.0/linux2.4-glibc2.3-x86/bin>;

CPULIMIT

1.0 min of hostabc

Fri Feb 27 15:07:59 2010: Started on <hostabc>, Execution Home </home/user12>, Execution CWD </home/user12/home/lsf/lsf8.0.slt/8.0/linux2.4-glibc2.3-x86/bin>;

Fri Feb 27 15:09:30 2010: Exited by signal 24. The CPU time used is 84.0 seconds.

```
Fri Feb 27 15:09:30 2010: Completed <exit>; TERM_CPULIMIT: job killed after reaching
LSF CPU usage limit.
```

SCHEDULING PARAMETERS:

```
      r15s  r1m  r15m  ut      pg      io      ls      it      tmp      swp
memloadSched - - - - - - - - - - - - - - - -
loadStop      - - - - - - - - - - - - - - - -
```

Termination reasons displayed by bacct and bjobs

When LSF detects that a job is terminated, `bacct -l` and `bjobs -l` display one of the following termination reasons:

Keyword displayed by bacct	Termination reason	Integer value logged to JOB_FINISH in lsb.acct
TERM_ADMIN	Job killed by root or LSF administrator	15
TERM_BUCKET_KILL	Job killed with <code>bkill -b</code>	23
TERM_CHKPNT	Job killed after checkpointing	13
TERM_CPULIMIT	Job killed after reaching LSF CPU usage limit	12
TERM_CWD_NOTEXIST	Current working directory is not accessible or does not exist on the execution host	25
TERM_DEADLINE	Job killed after deadline expires	6
TERM_EXTERNAL_SIGNAL	Job killed by a signal external to LSF	17
TERM_FORCE_ADMIN	Job killed by root or LSF administrator without time for cleanup	9
TERM_FORCE_OWNER	Job killed by owner without time for cleanup	8
TERM_LOAD	Job killed after load exceeds threshold	3
TERM_MEMLIMIT	Job killed after reaching LSF memory usage limit	16
TERM_OTHER	Member of a chunk job in WAIT state killed and requeued after being switched to another queue.	4
TERM_OWNER	Job killed by owner	14
TERM_PREEMPT	Job killed after preemption	1
TERM_PROCESSLIMIT	Job killed after reaching LSF process limit	7
TERM_REQUEUE_ADMIN	Job killed and requeued by root or LSF administrator	11
TERM_REQUEUE_OWNER	Job killed and requeued by owner	10
TERM_RMS	Job exited from an RMS system error	18
TERM_RUNLIMIT	Job killed after reaching LSF run time limit	5
TERM_SWAP	Job killed after reaching LSF swap usage limit	20

Keyword displayed by bacct	Termination reason	Integer value logged to JOB_FINISH in lsb.acct
TERM_THREADLIMIT	Job killed after reaching LSF thread limit	21
TERM_UNKNOWN	LSF cannot determine a termination reason—0 is logged but TERM_UNKNOWN is not displayed	0
TERM_WINDOW	Job killed after queue run window closed	2
TERM_ZOMBIE	Job exited while LSF is not available	19

Tip:

The integer values logged to the JOB_FINISH event in l sb. acct and termination reason keywords are mapped in l sbat ch. h.

Restrictions

- If a queue-level JOB_CONTROL is configured, LSF cannot determine the result of the action. The termination reason only reflects what the termination reason could be in LSF.
- LSF cannot be guaranteed to catch any external signals sent directly to the job.
- In MultiCluster, a brequeue request sent from the submission cluster is translated to TERM_OWNER or TERM_ADMIN in the remote execution cluster. The termination reason in the email notification sent from the execution cluster as well as that in the l sb. acct is set to TERM_OWNER or TERM_ADMIN.

Example output of bacct and bhist

Example termination cause	Termination reason in bacct -l	Example bhist output
bkill -s KILL bkill <i>job_ID</i>	Completed <exit>; TERM_OWNER or TERM_ADMIN	Thu Mar 13 17:32:05: Signal <KILL> requested by user or administrator <user2>; Thu Mar 13 17:32:06: Exited by signal 2. The CPU time used is 0.1 seconds;
bkill -r	Completed <exit>; TERM_FORCE_ADMIN or TERM_FORCE_OWNER when sbatchd is not reachable. Otherwise, TERM_USER or TERM_ADMIN	Thu Mar 13 17:32:05: Signal <KILL> requested by user or administrator <user2>; Thu Mar 13 17:32:06: Exited by signal 2. The CPU time used is 0.1 seconds;
TERMINATE_WHEN	Completed <exit>; TERM_LOAD/ TERM_WINDOWS/ TERM_PREEMPT	Thu Mar 13 17:33:16: Signal <KILL> requested by user or administrator <user2>; Thu Mar 13 17:33:18: Exited by signal 2. The CPU time used is 0.1 seconds;
Memory limit reached	Completed <exit>; TERM_MEMLIMIT	Thu Mar 13 19:31:13: Exited by signal 2. The CPU time used is 0.1 seconds;

Example termination cause	Termination reason in bacct -l	Example bhist output
Run limit reached	Completed <exit>; TERM_RUNLIMIT	Thu Mar 13 20:18:32: Exited by signal 2. The CPU time used is 0.1 seconds.
CPU limit	Completed <exit>; TERM_CPULIMIT	Thu Mar 13 18:47:13: Exited by signal 24. The CPU time used is 62.0 seconds;
Swap limit	Completed <exit>; TERM_SWAPLIMIT	Thu Mar 13 18:47:13: Exited by signal 24. The CPU time used is 62.0 seconds;
Regular job exits when host crashes	Rusage 0, Completed <exit>; TERM_ZOMBIE	Thu Jun 12 15:49:02: Unknown; unable to reach the execution host; Thu Jun 12 16:10:32: Running; Thu Jun 12 16:10:38: Exited with exit code 143. The CPU time used is 0.0 seconds;
bqueue -r	For each requeue, Completed <exit>; TERM_REQUEUE_ADMIN or TERM_REQUEUE_OWNER	Thu Mar 13 17:46:39: Signal <REQUEUE_PEND> requested by user or administrator <user2>; Thu Mar 13 17:46:56: Exited by signal 2. The CPU time used is 0.1 seconds;
bchkpnt -k	On the first run: Completed <exit>; TERM_CHKPNT	Wed Apr 16 16:00:48: Checkpoint succeeded (actpid 931249); Wed Apr 16 16:01:03: Exited with exit code 137. The CPU time used is 0.0 seconds;
Kill -9 <RES> and job	Completed <exit>; TERM_EXTERNAL_SIGNAL	Thu Mar 13 17:30:43: Exited by signal 15. The CPU time used is 0.1 seconds;
Others	Completed <exit>;	Thu Mar 13 17:30:43: Exited with 3; The CPU time used is 0.1 seconds;

Platform LSF job exit codes

Exit codes are generated by LSF when jobs end due to signals received instead of exiting normally. LSF collects exit codes via the `wait3()` system call on UNIX platforms. The LSF exit code is a result of the system exit values. Exit codes less than 128 relate to application exit values, while exit codes greater than 128 relate to system signal exit values (LSF adds 128 to system values). Use `bhist` to see the exit code for your job.

How or why the job may have been signaled, or exited with a certain exit code, can be application and/or system specific. The application or system logs might be able to give a better description of the problem.

Note:

Termination signals are operating system dependent, so signal 5 may not be SIGTRAP and 11 may not be SIGSEGV on all UNIX and Linux systems. You need to pay attention to the execution host type in order to correct translate the exit value if the job has been signaled.

Application exit values

The most common cause of abnormal LSF job termination is due to application system exit values. If your application had an explicit exit value less than 128, `bjobs` and `bhist` display the actual exit code of the application; for example,

```
Exited with exit code 3
```

. You would have to refer to the application code for the meaning of exit code 3.

It is possible for a job to explicitly exit with an exit code greater than 128, which can be confused with the corresponding system signal. Make sure that applications you write do not use exit codes greater than 128.

System signal exit values

Jobs terminated with a system signal are returned by LSF as exit codes greater than 128 such that $\text{exit_code} - 128 = \text{signal_value}$. For example, exit code 133 means that the job was terminated with signal 5 (SIGTRAP on most systems, $133 - 128 = 5$). A job with exit code 130 was terminated with signal 2 (SIGINT on most systems, $130 - 128 = 2$).

Some operating systems define exit values as 0-255. As a result, negative exit values or values > 255 may have a wrap-around effect on that range. The most common example of this is a program that exits -1 will be seen with "exit code 255" in LSF.

bhist and bjobs output

In most cases, `bjobs` and `bhist` show the application exit value ($128 + \text{signal}$). In some cases, `bjobs` and `bhist` show the actual signal value.

If LSF sends catchable signals to the job, it displays the exit value. For example, if you run `bkill jobID` to kill the job, LSF passes SIGINT, which causes the job to exit with exit code 130 (SIGINT is 2 on most systems, $128 + 2 = 130$).

If LSF sends uncatchable signals to the job, then the entire process group for the job exits with the corresponding signal. For example, if you run `bkill -s SEGV jobID` to kill the job, `bjobs` and `bhist` show:

```
Exited by signal 7
```

In addition `bj obs` displays the termination reason immediately following the exit code or signal value. For example:

```
Exited by signal 24. The CPU time used is 84.0 seconds.
Completed <exit>; TERM_CPULIMIT: job killed after reaching LSF CPU usage limit.
```

Unknown termination reasons appear without a detailed description in the `bj obs` output as follows:

```
Completed <exit>;
```

Example

The following example shows a job that exited with exit code 130, which means that the job was terminated by the owner.

```
bkill 248
Job <248> is being terminated
bjobs -l 248
Job <248>, User <user1>, Project <default>, Status <EXIT>, Queue <normal>, Command
Sun May 31 13:10:51 2009: Submitted from host <host1>, CWD <$HOME>;
Sun May 31 13:10:54 2009: Started on <host5>, Execution Home </home/user1>, Execution
CWD <$HOME>;
Sun May 31 13:11:03 2009: Exited with exit code 130. The CPU time used is 0.9 seconds.
Sun May 31 13:11:03 2009: Completed <exit>; TERM_OWNER: job killed by owner.

SCHEDULING PARAMETERS:
      r15s  r1m  r15m  ut      pg    io    ls    it    tmp    swp    mem
loadSched -    -    -    -      -    -    -    -    -    -    -
loadStop  -    -    -    -      -    -    -    -    -    -    -
```


Troubleshooting and Error Messages

Shared file access

A frequent problem with LSF is non-accessible files due to a non-uniform file space. If a task is run on a remote host where a file it requires cannot be accessed using the same name, an error results. Almost all interactive LSF commands fail if the user's current working directory cannot be found on the remote host.

Shared files on UNIX

If you are running NFS, rearranging the NFS mount table may solve the problem. If your system is running the `automount` server, LSF tries to map the filenames, and in most cases it succeeds. If shared mounts are used, the mapping may break for those files. In such cases, specific measures need to be taken to get around it.

The automount maps must be managed through NIS. When LSF tries to map filenames, it assumes that automounted file systems are mounted under the `/tmp_mnt` directory.

Shared files across UNIX and Windows

For file sharing across UNIX and Windows, you require a third party NFS product on Windows to export directories from Windows to UNIX.

Shared files on Windows

1. To share files among Windows machines, set up a share on the server and access it from the client. You can access files on the share either by specifying a UNC path (`\\server\share\path`) or connecting the share to a local drive name and using a `drive: \path` syntax. Using UNC is recommended because drive mappings may be different across machines, while UNC allows you to unambiguously refer to a file on the network.

Common Platform LSF problems

Most problems are due to incorrect installation or configuration.

- Check the error log files first.

Often the log message points directly to the problem.

LIM dies quietly

1. Run the following command to check for errors in the LIM configuration files.

lsadmin ckconfig -v

This displays most configuration errors. If this does not report any errors, check in the LIM error log.

LIM unavailable

Sometimes the LIM is up, but executing the `lsl load` command prints the following error message:

```
Communication time out.
```

If the LIM has just been started, this is normal, because the LIM needs time to get initialized by reading configuration files and contacting other LIMs. If the LIM does not become available within one or two minutes, check the LIM error log for the host you are working on.

To prevent communication timeouts when starting or restarting the local LIM, define the parameter `LSF_SERVER_HOSTS` in the `lsf.conf` file. The client will contact the LIM on one of the `LSF_SERVER_HOSTS` and execute the command, provided that at least one of the hosts defined in the list has a LIM that is up and running.

When the local LIM is running but there is no master LIM in the cluster, LSF applications display the following message:

```
Cannot locate master LIM now, try later.
```

1. Check the LIM error logs on the first few hosts listed in the Host section of the `lsf.cluster.cluster_name` file. If `LSF_MASTER_LIST` is defined in `lsf.conf`, check the LIM error logs on the hosts listed in this parameter instead.

Master LIM is down

Sometimes the master LIM is up, but executing the `lsl load` or `lshosts` command prints the following error message:

```
Master LIM is down; try later
```

If the `/etc/hosts` file on the host where the master LIM is running is configured with the host name assigned to the loopback IP address (127.0.0.1), LSF client LIMs cannot contact the master LIM. When the master LIM starts up, it sets its official host name and IP address to the loopback address. Any client requests will get the master LIM address as 127.0.0.1, and try to connect to it, and in fact will try to access itself.

1. Check the IP configuration of your master LIM in `/etc/hosts`. The following example incorrectly sets the master LIM IP address to the loopback address:

```
127. 0. 0. 1      local host      myhostname
```

The following example correctly sets the master LIM IP address:

```
127. 0. 0. 1      local host
192. 168. 123. 123 myhostname
```

For a master LIM running on a host that uses an IPv6 address, the loopback address is

```
::1
```

The following example correctly sets the master LIM IP address using an IPv6 address:

```
::1          local host  i pv6- local host  i pv6- loopback
fe00::0      i pv6- local net
ff00::0      i pv6- mcast prefix
ff02::1      i pv6- all nodes
ff02::2      i pv6- all routers
ff02::3      i pv6- all hosts
```

RES does not start

1. Check the RES error log.

User permission denied

If remote execution fails with the following error message, the remote host could not securely determine the user ID of the user requesting remote execution.

```
User permission denied.
```

1. Check the RES error log on the remote host; this usually contains a more detailed error message.
2. If you are not using an identification daemon (LSF_AUTH is not defined in the `lsf.conf` file), then all applications that do remote executions must be owned by root with the `setuid` bit set. This can be done as follows.

chmod 4755 filename

3. If the binaries are on an NFS-mounted file system, make sure that the file system is not mounted with the `nosuid` flag.
4. If you are using an identification daemon (defined in the `lsf.conf` file by LSF_AUTH), `inetd` must be configured to run the daemon. The identification daemon must not be run directly.
5. If LSF_USE_HOSTEQUIV is defined in the `lsf.conf` file, check if `/etc/hosts.equiv` or `HOME/.rhosts` on the destination host has the client host name in it. Inconsistent host names in a name server with `/etc/hosts` and `/etc/hosts.equiv` can also cause this problem.
6. On SGI hosts running a name server, you can try the following command to tell the host name lookup code to search the `/etc/hosts` file before calling the name server.

setenv HOSTRESORDER "local,nis,bind"

7. For Windows hosts, users must register and update their Windows passwords using the `lspasswd` command. Passwords must be 3 characters or longer, and 31 characters or less.

For Windows password authentication in a non-shared file system environment, you must define the parameter LSF_MASTER_LIST in `lsf.conf` so that jobs will run with correct permissions. If you do not define this parameter, LSF assumes that the cluster uses a shared file system environment.

Non-uniform file name space

A command may fail with the following error message due to a non-uniform file name space.

```
chdir(...) failed: no such file or directory
```

You are trying to execute a command remotely, where either your current working directory does not exist on the remote host, or your current working directory is mapped to a different name on the remote host.

- If your current working directory does not exist on a remote host, you should not execute commands remotely on that host.

On UNIX

- If the directory exists, but is mapped to a different name on the remote host, you have to create symbolic links to make them consistent.
- LSF can resolve most, but not all, problems using `automount`. The automount maps must be managed through NIS.

Follow the instructions in your *Release Notes* for obtaining technical support if you are running automount and LSF is not able to locate directories on remote hosts.

Batch daemons die quietly

1. First, check the `sbatchd` and `mbatchd` error logs. Try running the following command to check the configuration.

badadmin ckconfig

This reports most errors. You should also check if there is any email in the LSF administrator's mailbox. If the `mbatchd` is running but the `sbatchd` dies on some hosts, it may be because `mbatchd` has not been configured to use those hosts.

sbatchd starts but mbatchd does not

1. Check whether LIM is running. You can test this by running the `lslsd` command. If LIM is not running properly, follow the suggestions in this chapter to fix the LIM first. It is possible that `mbatchd` is temporarily unavailable because the master LIM is temporarily unknown, causing the following error message.

`sbatchd: unknown service`

2. Check whether services are registered properly.

Detached processes

LSF uses process groups to keep track of all the processes of a job.

1. When a job is launched, the application runs under the job-RES (or root) process group.
2. If an application creates a new process group, and its PPID still belongs to the job, the PIM can track this new process group as part of the job.

However, if the application forks a child, the child becomes a new process group, and the parent dies immediately, the child process group is now orphaned and cannot be tracked

Any process that daemonizes itself is almost certainly lost (orphans child processes) because it changes its process group right after being detached. The only reliable way to not lose track of a process is to prevent it from using a new process group.

Host not used by LSF

`mbatchd` allows `sbatchd` to run only on the hosts listed in the `Host` section of the `lsb.hosts` file. If you try to configure an unknown host in the `HostGroup` or `HostPartition` sections of the `lsb.hosts` file, or as a `HOSTS` definition for a queue in the `lsb.queues` file, `mbatchd` logs the following message.

mbatchd on host: LSB_CONFDIR/cluster1/configdir/file(line #): Host hostname is not used by lsbatch; ignored

If you start sbatchd on a host that is not known by mbatchd, mbatchd rejects the sbatchd. The sbatchd logs the following message and exits.

This host is not used by lsbatch system.

- Run the following commands, in order, after adding a host to the configuration and before starting the daemons on the new host:

```
lsadmin reconfig
```

```
badmin reconfig
```

View UNKNOWN host type or model

1. Run `lshosts`. A model or type UNKNOWN indicates the host is down or the LIM on the host is down. You need to take immediate action. For example:

```
lshosts
HOST_NAME  type      model  cpuf  ncpus  maxmem  maxswp  server  RESOURCES
hostA      UNKNOWN    Ultra2  20.2  2      256M    710M    Yes    ()
```

Fix UNKNOWN matched host type or matched model

1. Start the host.
2. Run `lsadmin limstart` to start LIM on the host.

For example:

```
lsadmin limstart hostAStarting up LIM on <hostA> .... done
```

or, if EGO is enabled in the LSF cluster, you can also run:

```
egosh ego start lim hostAStarting up LIM on <hostA> .... done
```

You can specify more than one host name to start up LIM on multiple hosts. If you do not specify a host name, LIM is started up on the host from which the command is submitted.

On UNIX, in order to start up LIM remotely, you must be root or listed in `lsf.sudoers` (or `ego.sudoers` if EGO is enabled in the LSF cluster) and be able to run the `rsh` command across all hosts without entering a password.

3. Wait a few seconds, then run `lshosts` again. You should now be able to see a specific model or type for the host or DEFAULT. If you see DEFAULT, it means that automatic detection of host type or model has failed, and the host type configured in `lsf.shared` cannot be found. LSF will work on the host, but a DEFAULT model may be inefficient because of incorrect CPU factors. A DEFAULT type may also cause binary incompatibility because a job from a DEFAULT host type can be migrated to another DEFAULT host type.

View DEFAULT host type or model

If you see DEFAULT in `lim -t`, it means that automatic detection of host type or model has failed, and the host type configured in `lsf.shared` cannot be found. LSF will work on the host, but a DEFAULT model may be inefficient because of incorrect CPU factors. A DEFAULT type may also cause binary incompatibility because a job from a DEFAULT host type can be migrated to another DEFAULT host type.

1. Run `lshosts`. If Model or Type are displayed as DEFAULT when you use `lshosts` and automatic host model and type detection is enabled, you can leave it as is or change it. For example:

lshosts									
HOST_NAME	type	model	cpuf	ncpus	maxmem	maxswp	server	RESOURCES	
hostA	DEFAULT	DEFAULT		1	2	256M	710M	Yes	()

If model is DEFAULT, LSF will work correctly but the host will have a CPU factor of 1, which may not make efficient use of the host model.

If type is DEFAULT, there may be binary incompatibility. For example, there are 2 hosts, one is Solaris, the other is HP. If both hosts are set to type DEFAULT, it means jobs running on the Solaris host can be migrated to the HP host and vice-versa.

Fix DEFAULT matched host type or matched model

1. Run `lim -t` on the host whose type is DEFAULT:

```
lim -t
Host Type      : LINUX86
Host Architecture : SUNWUltra2_200_sparcv9
Physical Processors : 2
Cores per Processor : 4
Threads per Core: : 2
Matched Type: DEFAULT
Matched Architecture: DEFAULT
Matched Model: DEFAULT
CPU Factor      : 60.0
```

Note the value of Host Type and Host Architecture.

2. Edit `lsf.shared`.

- a) In the Host Type section, enter a new host type. Use the host type name detected with `lim -t`. For example:

```
Begin HostType
TYPENAME
DEFAULT
CRAYJ
LINUX86
...
End HostType
```

- b) In the Host Model section, enter the new host model with architecture and CPU factor. Use the architecture detected with `lim -t`. Add the host model to the end of the host model list. The limit for host model entries is 127. Lines commented out with # are not counted in the 127-line limit. For example:

```
Begin HostModel
MODELNAME CPUFACTOR ARCHITECTURE # keyword
Ultra2 20 SUNWUltra2_200_sparcv9
End HostModel
```

3. Save changes to `lsf.shared`.
4. Run `lsadm in reconfi g` to reconfigure LIM.
5. Wait a few seconds, and run `lim -t` again to check the type and model of the host.

Error messages

The following error messages are logged by the LSF daemons, or displayed by the following commands.

```
lsadmin ckconfig
badmin ckconfig
```

General errors

The messages listed in this section may be generated by any LSF daemon.

can't open file: error

The daemon could not open the named file for the reason given by *error*. This error is usually caused by incorrect file permissions or missing files. All directories in the path to the configuration files must have execute (x) permission for the LSF administrator, and the actual files must have read (r) permission. Missing files could be caused by incorrect path names in the `lsf.conf` file, running LSF daemons on a host where the configuration files have not been installed, or having a symbolic link pointing to a nonexistent file or directory.

file(line): malloc failed

Memory allocation failed. Either the host does not have enough available memory or swap space, or there is an internal error in the daemon. Check the program load and available swap space on the host; if the swap space is full, you must add more swap space or run fewer (or smaller) programs on that host.

auth_user: getservbyname(ident/tcp) failed: error; ident must be registered in services

LSF_AUTH=ident is defined in the `lsf.conf` file, but the `ident/tcp` service is not defined in the services database. Add `ident/tcp` to the services database, or remove LSF_AUTH from the `lsf.conf` file and set `uid root` those LSF binaries that require authentication.

auth_user: operation(<host>/<port>) failed: error

LSF_AUTH=ident is defined in the `lsf.conf` file, but the LSF daemon failed to contact the `identd` daemon on host. Check that `identd` is defined in `inetd.conf` and the `identd` daemon is running on host.

auth_user: Authentication data format error (rbuf=<data>) from <host>/<port>

auth_user: Authentication port mismatch (...) from <host>/<port>

LSF_AUTH=ident is defined in the `lsf.conf` file, but there is a protocol error between LSF and the `ident` daemon on *host*. Make sure the `ident` daemon on the host is configured correctly.

userok: Request from bad port (<port_number>), denied

LSF_AUTH is not defined, and the LSF daemon received a request that originates from a non-privileged port. The request is not serviced.

Set the LSF binaries to be owned by root with the `setuid` bit set, or define LSF_AUTH=ident and set up an `ident` server on all hosts in the cluster. If the binaries are on an NFS-mounted file system, make sure that the file system is not mounted with the `nosuid` flag.

userok: Forged username suspected from <host>/<port>: <claimed_user>/<actual_user>

The service request claimed to come from user *claimed_user* but `ident` authentication returned that the user was actually *actual_user*. The request was not serviced.


```
userok: ruserok(<host>, <uid>) failed
```

LSF_USE_HOSTEQUIV is defined in the `lsf.conf` file, but *host* has not been set up as an equivalent host (see `/etc/host.equiv`), and user *uid* has not set up a `.rhosts` file.

```
init_AcceptSock: RES service(res) not registered, exiting
```

```
init_AcceptSock: res/tcp: unknown service, exiting
```

```
initSock: LIM service not registered.
```

```
initSock: Service lim/udp is unknown. Read LSF Guide for help
```

```
get_ports: <serv> service not registered
```

The LSF services are not registered.

```
init_AcceptSock: Can't bind daemon socket to port <port>: error, exiting
```

```
init_ServSock: Could not bind socket to port <port>: error
```

These error messages can occur if you try to start a second LSF daemon (for example, RES is already running, and you execute RES again). If this is the case, and you want to start the new daemon, kill the running daemon or use the `lsadmi n` or `badmi n` commands to shut down or restart the daemon.

Configuration errors

The messages listed in this section are caused by problems in the LSF configuration files. General errors are listed first, and then errors from specific files.

```
file(line): Section name expected after Begin; ignoring section
```

```
file(line): Invalid section name name; ignoring section
```

The keyword `begin` at the specified line is not followed by a section name, or is followed by an unrecognized section name.

```
file(line): section section: Premature EOF
```

The end of file was reached before reading the end `section` line for the named section.

```
file(line): keyword line format error for section section; Ignore this section
```

The first line of the section should contain a list of keywords. This error is printed when the keyword line is incorrect or contains an unrecognized keyword.

```
file(line): values do not match keys for section section; Ignoring line
```

The number of fields on a line in a configuration section does not match the number of keywords. This may be caused by not putting `()` in a column to represent the default value.

```
file: HostModel section missing or invalid
```

```
file: Resource section missing or invalid
```

```
file: HostType section missing or invalid
```

The `HostModel`, `Resource`, or `HostType` section in the `lsf.shared` file is either missing or contains an unrecoverable error.

```
file(line): Name name reserved or previously defined. Ignoring index
```

The name assigned to an external load index must not be the same as any built-in or previously defined resource or load index.

file(line): Duplicate clustername name in section cluster. Ignoring current line

A cluster name is defined twice in the same lsf. shared file. The second definition is ignored.

file(line): Bad cpuFactor for host model model. Ignoring line

The CPU factor declared for the named host model in the lsf. shared file is not a valid number.

file(line): Too many host models, ignoring model name

You can declare a maximum of 127 host models in the lsf. shared file.

file(line): Resource name name too long in section resource. Should be less than 40 characters. Ignoring line

The maximum length of a resource name is 39 characters. Choose a shorter name for the resource.

file(line): Resource name name reserved or previously defined. Ignoring line.

You have attempted to define a resource name that is reserved by LSF or already defined in the lsf. shared file. Choose another name for the resource.

file(line): illegal character in resource name: name, section resource. Line ignored.

Resource names must begin with a letter in the set [a-zA-Z], followed by letters, digits or underscores [a-zA-Z0-9_].

LIM messages

The following messages are logged by the LIM:

main: LIM cannot run without licenses, exiting

The LSF software license key is not found or has expired. Check that FlexNet is set up correctly, or contact your LSF technical support.

main: Received request from unlicensed host <host>/<port>

LIM refuses to service requests from hosts that do not have licenses. Either your LSF license has expired, or you have configured LSF on more hosts than your license key allows.

initLicense: Trying to get license for LIM from source <LSF_CONFDIR/license.dat>

getLicense: Can't get software license for LIM from license file <LSF_CONFDIR/license.dat>: feature not yet available.

Your LSF license is not yet valid. Check whether the system clock is correct.

findHostbyAddr/<proc>: Host <host>/<port> is unknown by <myhostname>

function: Gethostbyaddr_(<host>/<port>) failed: error

main: Request from unknown host <host>/<port>: error

function: Received request from non-LSF host <host>/<port>

The daemon does not recognize *host*. The request is not serviced. These messages can occur if *host* was added to the configuration files, but not all the daemons have been reconfigured to read the new information. If the problem still occurs after reconfiguring all the daemons, check whether the host is a multi-addressed host.

rcvLoadVector: Sender (<host>/<port>) may have different config?

MasterRegister: Sender (host) may have different config?

LIM detected inconsistent configuration information with the sending LIM. Run the following command so that all the LIMs have the same configuration information.

Isadmin reconfig

Note any hosts that failed to be contacted.

rcvLoadVector: Got load from client-only host <host>/<port>. Kill LIM on <host>/<port>

A LIM is running on a client host. Run the following command, or go to the client host and kill the LIM daemon.

Isadmin limshutdown host

saveIndx: Unknown index name <name> from ELIM

LIM received an external load index name that is not defined in the lsf.shared file. If name is defined in lsf.shared, reconfigure the LIM. Otherwise, add name to the lsf.shared file and reconfigure all the LIMs.

saveIndx: ELIM over-riding value of index <name>

This is a warning message. The ELIM sent a value for one of the built-in index names. LIM uses the value from ELIM in place of the value obtained from the kernel.

getusr: Protocol error numIndx not read (cc=num): error

getusr: Protocol error on index number (cc=num): error

Protocol error between ELIM and LIM.

RES messages

These messages are logged by the RES.

doacceptconn: getpwnam(<username>@<host>/<port>) failed: error

doacceptconn: User <username> has uid <uid1> on client host <host>/<port>, uid <uid2> on RES host; assume bad user

authRequest: username/uid <userName>/<uid>@<host>/<port> does not exist

authRequest: Submitter's name <clname>@<clhost> is different from name <lname> on this host

RES assumes that a user has the same userID and username on all the LSF hosts. These messages occur if this assumption is violated. If the user is allowed to use LSF for interactive remote execution, make sure the user's account has the same userID and username on all LSF hosts.

doacceptconn: root remote execution permission denied

authRequest: root job submission rejected

Root tried to execute or submit a job but LSF_ROOT_REX is not defined in the lsf.conf file.

resControl: operation permission denied, uid = <uid>

The user with user ID *uid* is not allowed to make RES control requests. Only the LSF manager, or root if LSF_ROOT_REX is defined in lsf.conf, can make RES control requests.

```
resControl: access(respath, X_OK): error
```

The RES received a reboot request, but failed to find the file `respath` to re-execute itself. Make sure `respath` contains the RES binary, and it has execution permission.

mbatchd and sbatchd messages

The following messages are logged by the `mbatchd` and `sbatchd` daemons:

```
renewJob: Job <jobId>: rename(<from>, <to>) failed: error
```

`mbatchd` failed in trying to re-submit a rerunnable job. Check that the file *from* exists and that the LSF administrator can rename the file. If *from* is in an AFS directory, check that the LSF administrator's token processing is properly setup.

See the document "Installing LSF on AFS" on the Platform Web site for more information about installing on AFS.

```
logJobInfo_: fopen(<logdir/info/jobfile>) failed: error
```

```
logJobInfo_: write <logdir/info/jobfile> <data> failed: error
```

```
logJobInfo_: seek <logdir/info/jobfile> failed: error
```

```
logJobInfo_: write <logdir/info/jobfile> xdrpos <pos> failed: error
```

```
logJobInfo_: write <logdir/info/jobfile> xdr buf len <len> failed: error
```

```
logJobInfo_: close(<logdir/info/jobfile>) failed: error
```

```
rmLogJobInfo: Job <jobId>: can't unlink(<logdir/info/jobfile>): error
```

```
rmLogJobInfo_: Job <jobId>: can't stat(<logdir/info/jobfile>): error
```

```
readLogJobInfo: Job <jobId>: can't open(<logdir/info/jobfile>): error
```

```
start_job: Job <jobId>: readLogJobInfo failed: error
```

```
readLogJobInfo: Job <jobId>: can't read(<logdir/info/jobfile>) size size:
error
```

```
initLog: mkdir(<logdir/info>) failed: error
```

```
<fname>: fopen(<logdir/file>) failed: error
```

```
getEl ogLock: Can't open existing lock file <logdir/file>: error
```

```
getEl ogLock: Error in opening lock file <logdir/file>: error
```

```
releaseEl ogLock: unlink(<logdir/lockfile>) failed: error
```

```
touchEl ogLock: Failed to open lock file <logdir/file>: error
```

```
touchEl ogLock: close <logdir/file> failed: error
```

`mbatchd` failed to create, remove, read, or write the log directory or a file in the log directory, for the reason given in *error*. Check that LSF administrator has read, write, and execute permissions on the `logdir` directory.

If `logdir` is on AFS, check that the instructions in the document "Installing LSF on AFS" on the Platform Web site have been followed. Use the `fs ls` command to verify that the LSF administrator owns `logdir` and that the directory has the correct acl.

replay_newjob: File <logfile> at line <line>: Queue <queue> not found, saving to queue <lost_and_found>

replay_switchjob: File <logfile> at line <line>: Destination queue <queue> not found, switching to queue <lost_and_found>

When mbatchd was reconfigured, jobs were found in *queue* but that queue is no longer in the configuration.

replay_startjob: JobId <jobId>: exec host <host> not found, saving to host <lost_and_found>

When mbatchd was reconfigured, the event log contained jobs dispatched to host, but that host is no longer configured to be used by LSF.

do_restartReq: Failed to get hData of host <host_name>/<host_addr>

mbatchd received a request from sbatchd on host *host_name*, but that host is not known to mbatchd. Either the configuration file has been changed but mbatchd has not been reconfigured to pick up the new configuration, or *host_name* is a client host but the sbatchd daemon is running on that host. Run the following command to reconfigure the mbatchd or kill the sbatchd daemon on *host_name*.

badadmin reconfig

Platform LSF command messages

LSF daemon (LIM) not responding ... still trying

During LIM restart, LSF commands will fail and display this error message. User programs linked to the LIM API will also fail for the same reason. This message is displayed when LIM running on the master host list or server host list is restarted after configuration changes, such as adding new resources, binary upgrade, and so on.

Use LSF_LIM_API_NTRIES in *lsf.conf* or as an environment variable to define how many times LSF commands will retry to communicate with the LIM API while LIM is not available.

LSF_LIM_API_NTRIES is ignored by LSF and EGO daemons and all EGO commands.

When LSB_API_VERBOSE=Y in *lsf.conf*, LSF batch commands will display the not responding retry error message to *stderr* when LIM is not available.

When LSB_API_VERBOSE=N in *lsf.conf*, LSF batch commands will not display the retry error message when LIM is not available.

Batch command client messages

LSF displays error messages when a batch command cannot communicate with mbatchd. The following table provides a list of possible error reasons and the associated error message output.

Point of failure	Possible reason	Error message output
Establishing a connection with <code>mbatchd</code>	<code>mbatchd</code> is too busy to accept new connections. The <code>connect()</code> system call times out.	LSF is processing your request. Please wait...
	<code>mbatchd</code> is down or there is no process listening at either the <code>LSB_MBD_PORT</code> or the <code>LSB_QUERY_PORT</code>	LSF is down. Please wait...
	<code>mbatchd</code> is down and the <code>LSB_QUERY_PORT</code> is busy	<code>bhost.s</code> displays "LSF is down. Please wait. . ." <code>bj obs</code> displays "Cannot connect to LSF. Please wait..."
	Socket error on the client side	Cannot connect to LSF. Please wait...
	<code>connect()</code> system call fails	Cannot connect to LSF. Please wait...
	Internal library error	Cannot connect to LSF. Please wait...
Send/receive handshake message to/from <code>mbatchd</code>	<code>mbatchd</code> is busy. Client times out when waiting to receive a message from <code>mbatchd</code> .	LSF is processing your request. Please wait...
	Socket <code>read()/write()</code> fails	Cannot connect to LSF. Please wait...
	Internal library error	Cannot connect to LSF. Please wait...

EGO command messages

You cannot run the `egosh` command because the administrator has chosen not to enable EGO in `lsf.conf`: `LSF_ENABLE_EGO=N`.

If EGO is disabled, the `egosh` command cannot find `ego.conf` or cannot contact `vemkd` (not started).

Set daemon message log to debug level

The message log level for LSF daemons is set in `lsf.conf` with the parameter `LSF_LOG_MASK`. To include debugging messages, set `LSF_LOG_MASK` to one of:

- `LOG_DEBUG`
- `LOG_DEBUG1`
- `LOG_DEBUG2`
- `LOG_DEBUG3`

By default, `LSF_LOG_MASK=LOG_WARNING` and these debugging messages are not displayed.

The debugging log classes for LSF daemons is set in `lsf.conf` with the parameters `LSB_DEBUG_CMD`, `LSB_DEBUG_MBD`, `LSB_DEBUG_SBD`, `LSB_DEBUG_SCH`, `LSF_DEBUG_LIM`, `LSF_DEBUG_RES`.

The location of log files is specified with the parameter `LSF_LOGDIR` in `lsf.conf`.

You can use the `lsadmi n` and `badmi n` commands to temporarily change the class, log file, or message log level for specific daemons such as `LIM`, `RES`, `mbatchd`, `sbatchd`, and `mbschd` without changing `lsf.conf`.

How the message log level takes effect

The message log level you set will only be in effect from the time you set it until you turn it off or the daemon stops running, whichever is sooner. If the daemon is restarted, its message log level is reset back to the value of `LSF_LOG_MASK` and the log file is stored in the directory specified by `LSF_LOGDIR`.

Limitations

When debug or timing level is set for `RES` with `lsadmi n resdebug`, or `lsadmi n res time`, the debug level only affects root `RES`. The root `RES` is the `RES` that runs under the root user ID.

Application `RES`s always use `lsf.conf` to set the debug environment. Application `RES`s are the `RES`s that have been created by `sbatchd` to service jobs and run under the ID of the user who submitted the job.

This means that any `RES` that has been launched automatically by the LSF system will not be affected by temporary debug or timing settings. The application `RES` will retain settings specified in `lsf.conf`.

Debug commands for daemons

The following commands set temporary message log level options for `LIM`, `RES`, `mbatchd`, `sbatchd`, and `mbschd`.

```
lsadmin limdebug [-c class_name] [-l debug_level] [-f logfile_name] [-o] [host_name]
lsadmin resdebug [-c class_name] [-l debug_level] [-f logfile_name] [-o] [host_name]
badmin mbddebug [-c class_name] [-l debug_level] [-f logfile_name] [-o]
badmin sbddebug [-c class_name] [-l debug_level] [-f logfile_name] [-o] [host_name]
badmin schddebug [-c class_name] [-l debug_level] [-f logfile_name] [-o]
```

For a detailed description of `lsadmi n` and `badmi n`, see the *Platform LSF Command Reference*.

Examples

lsadmin limdebug -c "LC_MULTI LC_PIM" -f myfile hostA hostB

Log additional messages for the `LIM` daemon running on `hostA` and `hostB`, related to MultiCluster and PIM. Create log files in the `LSF_LOGDIR` directory with the name `myfile.lim.log.hostA`, and `myfile.lim.log.hostB`. The debug level is the default value, `LOG_DEBUG` level in parameter `LSF_LOG_MASK`.

lsadmin limdebug -o hostA hostB

Turn off temporary debug settings for LIM on hostA and hostB and reset them to the daemon starting state. The message log level is reset back to the value of LSF_LOG_MASK and classes are reset to the value of LSF_DEBUG_RES, LSF_DEBUG_LIM, LSB_DEBUG_MBD, LSB_DEBUG_SBD, and LSB_DEBUG_SCH. The log file is reset to the LSF system log file in the directory specified by LSF_LOGDIR in the format *daemon_name.log.host_name*.

badadmin sbddebug -o

Turn off temporary debug settings for sbat chd on the local host (host from which the command was submitted) and reset them to the daemon starting state. The message log level is reset back to the value of LSF_LOG_MASK and classes are reset to the value of LSF_DEBUG_RES, LSF_DEBUG_LIM, LSB_DEBUG_MBD, LSB_DEBUG_SBD, and LSB_DEBUG_SCH. The log file is reset to the LSF system log file in the directory specified by LSF_LOGDIR in the format *daemon_name.log.host_name*.

badadmin mbddebug -l 1

Log messages for mbat chd running on the local host and set the log message level to LOG_DEBUG1. This command must be submitted from the host on which mbat chd is running because *host_name* cannot be specified with mbddebug.

badadmin sbddebug -f hostB/myfolder/myfile hostA

Log messages for sbat chd running on hostA, to the directory myfi le on the server hostB, with the file name myfi le. sbat chd. log. hostA. The debug level is the default value, LOG_DEBUG level in parameter LSF_LOG_MASK.

badadmin schddebug -l 2

Log messages for mbat chd running on the local host and set the log message level to LOG_DEBUG2. This command must be submitted from the host on which mbat chd is running because *host_name* cannot be specified with schddebug.

badadmin schddebug -l 1 -c "LC_PERFM"

badadmin schdtime -l 2

Activate the LSF scheduling debug feature.

Log performance messages for mbat chd running on the local host and set the log message level to LOG_DEBUG. Set the timing level for mbschd to include two levels of timing information.

lsadmin resdebug -o hostA

Turn off temporary debug settings for RES on hostA and reset them to the daemon starting state. The message log level is reset back to the value of LSF_LOG_MASK and classes are reset to the value of LSF_DEBUG_RES, LSF_DEBUG_LIM, LSB_DEBUG_MBD, LSB_DEBUG_SBD, and LSB_DEBUG_SCH. The log file is reset to the LSF system log file in the directory specified by LSF_LOGDIR in the format *daemon_name.log.host_name*.

Set daemon timing levels

The timing log level for LSF daemons is set in `lsf.conf` with the parameters `LSB_TIME_CMD`, `LSB_TIME_MBD`, `LSB_TIME_SBD`, `LSB_TIME_SCH`, `LSF_TIME_LIM`, `LSF_TIME_RES`.

The location of log files is specified with the parameter `LSF_LOGDIR` in `lsf.conf`. Timing is included in the same log files as messages.

To change the timing log level, you need to stop any running daemons, change `lsf.conf`, and then restart the daemons.

It is useful to track timing to evaluate the performance of the LSF system. You can use the `lsadmin` and `badmi` commands to temporarily change the timing log level for specific daemons such as `LIM`, `RES`, `mbatchd`, `sbatchd`, and `mbschd` without changing `lsf.conf`.

`LSF_TIME_RES` is not supported on Windows.

How the timing log level takes effect

The timing log level you set will only be in effect from the time you set it until you turn the timing log level off or the daemon stops running, whichever is sooner. If the daemon is restarted, its timing log level is reset back to the value of the corresponding parameter for the daemon (`LSB_TIME_MBD`, `LSB_TIME_SBD`, `LSF_TIME_LIM`, `LSF_TIME_RES`). Timing log messages are stored in the same file as other log messages in the directory specified with the parameter `LSF_LOGDIR` in `lsf.conf`.

Limitations

When debug or timing level is set for `RES` with `lsadmin resdebug`, or `lsadmin restime`, the debug level only affects root `RES`. The root `RES` is the `RES` that runs under the root user ID.

An application `RES` always uses `lsf.conf` to set the debug environment. An application `RES` is the `RES` that has been created by `sbatchd` to service jobs and run under the ID of the user who submitted the job.

This means that any `RES` that has been launched automatically by the LSF system will not be affected by temporary debug or timing settings. The application `RES` will retain settings specified in `lsf.conf`.

Timing level commands for daemons

The total execution time of a function in the LSF system is recorded to evaluate response time of jobs submitted locally or remotely.

The following commands set temporary timing options for `LIM`, `RES`, `mbatchd`, `sbatchd`, and `mbschd`.

```
lsadmin limtime [-l timing_level] [-f logfile_name] [-o] [host_name]
lsadmin restime [-l timing_level] [-f logfile_name] [-o] [host_name]
badmin mbdtime [-l timing_level] [-f logfile_name] [-o]
badmin sbdtime [-l timing_level] [-f logfile_name] [-o] [host_name]
badmin schdtime [-l timing_level] [-f logfile_name] [-o]
```

For a detailed description of `lsadmin` and `badmi`, see the *Platform LSF Command Reference*.

Time-Based Configuration

26

Time Configuration

Specify time values

To specify a time value, a specific point in time, specify at least the hour. Day and minutes are optional.

Time value syntax

```
time = hour | hour: minute | day: hour: minute
```

hour

Integer from 0 to 23, representing the hour of the day.

minute

Integer from 0 to 59, representing the minute of the hour.

If you do not specify the minute, LSF assumes the first minute of the hour (:00).

day

Integer from 0 to 6, representing the day of the week, 0 represents Monday and 6 represents Sunday.

If you do not specify the day, LSF assumes every day. If you do specify the day, you must also specify the minute.

Time windows

To specify a time window, specify two time values separated by a hyphen (-), with no space in between.

```
time_window = begin_time-end_time
```

Time format

Times are specified in the format:

```
[ day: ] hour[: minute]
```

where all fields are numbers with the following ranges:

- *day of the week*: 0-6 (0 is Sunday)
- *hour*: 0-23
- *minute*: 0-59

Specify a time window one of the following ways:

- *hour-hour*
- *hour.minute-hour.minute*
- *day.hour.minute-day.hour.minute*

The default value for minute is 0 (on the hour); the default value for day is every day of the week.

You must specify at least the hour. Day of the week and minute are optional. Both the start time and end time values must use the same syntax. If you do not specify a minute, LSF assumes the first minute of the hour (: 00). If you do not specify a day, LSF assumes every day of the week. If you do specify the day, you must also specify the minute.

You can specify multiple time windows, but they cannot overlap. For example:

```
timeWindow(8: 00- 14: 00 18: 00- 22: 00)
```

is correct, but

```
timeWindow(8: 00- 14: 00 11: 00- 15: 00)
```

is not valid.

Examples of time windows

Daily window

To specify a daily window omit the day field from the time window. Use either the hour - hour or hour: minute - hour: minute format. For example, to specify a daily 8:30 a.m. to 6:30 p.m window:

```
8: 30- 18: 30
```

Overnight window

To specify an overnight window make *time1* greater than *time2*. For example, to specify 6:30 p.m. to 8:30 a.m. the following day:

```
18: 30- 8: 30
```

Weekend window

To specify a weekend window use the day field. For example, to specify Friday at 6:30 p.m to Monday at 8:30 a.m.:

```
5: 18: 30- 1: 8: 30
```

Time expressions

Time expressions use time windows to specify when to change configurations.

Time expression syntax

A time expression is made up of the `time` keyword followed by one or more space-separated time windows enclosed in parenthesis. Time expressions can be combined using the `&&`, `||`, and `!` logical operators.

The syntax for a time expression is:

```
expression = time( time_window [ time_window ... ] )
              | expression && expression
              | expression || expression
              | !expression
```

Example

Both of the following expressions specify weekends (Friday evening at 6:30 p.m. until Monday morning at 8:30 a.m.) and nights (8:00 p.m. to 8:30 a.m. daily).

```
time(5: 18: 30- 1: 8: 30 20: 00- 8: 30)
time(5: 18: 30- 1: 8: 30) || time(20: 00- 8: 30)
```


Automatic time-based configuration

Variable configuration is used to automatically change LSF configuration based on time windows. It is supported in the following files:

- `lsb.hosts`
- `lsb.params`
- `lsb.queues`
- `lsb.resources`
- `lsb.users`
- `lsf.licensescheduler`

You define automatic configuration changes in configuration files by using if-else constructs and time expressions. After you change the files, reconfigure the cluster with the `badm n reconfi g` command.

The expressions are evaluated by LSF every 10 minutes based on `mbat chd` start time. When an expression evaluates true, LSF dynamically changes the configuration based on the associated configuration statements. Reconfiguration is done in real time without restarting `mbat chd`, providing continuous system availability.

In the following examples, the `#i f`, `#el se`, `#endi f` are not interpreted as comments by LSF but as if-else constructs.

lsb.hosts example

```
Begin Host
HOST_NAME    r15s    r1m    pg
host1        3/5     3/5     12/20
#if time(5:16:30-1:8:30 20:00-8:30)
host2        3/5     3/5     12/20
#else
host2        2/3     2/3     10/12
#endif
host3        3/5     3/5     12/20
End Host
```

lsb.params example

```
# if 18:30-19:30 is your short job express period, but
# you want all jobs going to the short queue by default
# and be subject to the thresholds of that queue
# for all other hours, normal is the default queue
#if time(18:30-19:30)
DEFAULT_QUEUE=short
#else
DEFAULT_QUEUE=normal
#endif
```

lsb.queues example

```
Begin Queue
...
#if time(8:30-18:30)
    INTERACTIVE = ONLY # interactive only during day shift
#endif
...
End Queue
```

lsb.resources example

```
# Example: limit usage of hosts in 'license1' group and time based configuration
# - 10 jobs can run from normal queue
# - any number can run from short queue between 18:30 and 19:30
# all other hours you are limited to 100 slots in the short queue
```

```
# - each other queue can run 30 jobs
Begin Limit
PER_QUEUE           HOSTS      SLOTS    # Example
normal              license1    10
# if time(18:30-19:30)
short               license1    -
#else
short               license1    100
#endif
(all ~normal ~short) license1    30
End Limit
```

lsb.users example

From 12 - 1 p.m. daily, user smith has 10 job slots, but during other hours, user has only 5 job slots.

```
Begin User
USER_NAME           MAX_JOBS    JL/P
#if time (12-13)
smith               10         -
#else
smith               5          -
default             1          -
#endif
End User
```

lsf.licensescheduler example

```
Begin Feature
NAME = f1
#if time(5:16:30-1:8:30 20:00-8:30)
DISTRIBUTION=Lan(P1 2/5 P2 1)
#elif time(3:8:30-3:18:30)
DISTRIBUTION=Lan(P3 1)
#else
DISTRIBUTION=Lan(P1 1 P2 2/5)
#endif
End Feature
```

Create if-else constructs

The if-else construct can express single decisions and multi-way decisions by including elif statements in the construct.

If-else

The syntax for constructing if-else expressions is:

```
#if time(expression) statement #else statement #endif
```

The #endif part is mandatory and the #else part is optional.

elif

The #elif expressions are evaluated in order. If any expression is true, the associated statement is used, and this terminates the whole chain.

The #else part handles the default case where none of the other conditions are satisfied.

When you use #elif, the #else and #endif parts are mandatory.

```
#if time(expression)
statement
#elif time(expression)
statement
#elif time(expression)
statement
#else
```

```
statement  
#endif
```

Verify configuration

1. Depending on what you have configured, use the following LSF commands to verify time configuration:
 1.
 - `bhosts`
 - `bladmin ckconfig`
 - `blimits -c`
 - `blinfo`
 - `blstat`
 - `bparams`
 - `bqueues`
 - `bresources`
 - `busers`

Dispatch and run windows

Both dispatch and run windows are time windows that control when LSF jobs start and run.

- Dispatch windows can be defined in `l sb. hosts`. Dispatch and run windows can be defined in `l sb. queues`.
- Hosts can only have dispatch windows. Queues can have dispatch windows and run windows.
- Both windows affect job starting; only run windows affect the stopping of jobs.
- Dispatch windows define when hosts and queues are active and inactive. It does not control job submission.
- Run windows define when jobs can and cannot run. While a run window is closed, LSF cannot start any of the jobs placed in the queue, or finish any of the jobs already running.
- When a dispatch window closes, running jobs continue and finish, and no new jobs can be dispatched to the host or from the queue. When a run window closes, LSF suspends running jobs, but new jobs can still be submitted to the queue.

Run windows

Queues can be configured with a run window, which specifies one or more time periods during which jobs in the queue are allowed to run. Once a run window is configured, jobs in the queue cannot run outside of the run window.

Jobs can be submitted to a queue at any time; if the run window is closed, the jobs remain pending until it opens again. If the run window is open, jobs are placed and dispatched as usual. When an open run window closes, running jobs are suspended, and pending jobs remain pending. The suspended jobs are resumed when the window opens again.

Configure run windows

1. To configure a run window, set `RUN_WINDOW` in `l sb. queues`.

For example, to specify that the run window will be open from 4:30 a.m. to noon, type:

```
RUN_WINDOW = 4:30-12:00
```

You can specify multiple time windows.

View information about run windows

1. Use `bqueues -l` to display information about queue run windows.

Dispatch windows

Queues can be configured with a dispatch window, which specifies one or more time periods during which jobs are accepted. Hosts can be configured with a dispatch window, which specifies one or more time periods during which jobs are allowed to start.

Once a dispatch window is configured, LSF cannot dispatch jobs outside of the window. By default, no dispatch windows are configured (the windows are always open).

Dispatch windows have no effect on jobs that have already been dispatched to the execution host; jobs are allowed to run outside the dispatch windows, as long as the queue run window is open.

Queue-level

Each queue can have a dispatch window. A queue can only dispatch jobs when the window is open.

You can submit jobs to a queue at any time; if the queue dispatch window is closed, the jobs remain pending in the queue until the dispatch window opens again.

Host-level

Each host can have dispatch windows. A host is not eligible to accept jobs when its dispatch windows are closed.

Configure host dispatch windows

1. To configure dispatch windows for a host, set `DISPATCH_WINDOW` in `l sb. host s` and specify one or more time windows. If no host dispatch window is configured, the window is always open.

Configure queue dispatch windows

1. To configure dispatch windows for queues, set `DISPATCH_WINDOW` in `l sb. queues` and specify one or more time windows. If no queue dispatch window is configured, the window is always open.

Display host dispatch windows

1. Use `bhost s -l` to display host dispatch windows.

Display queue dispatch windows

1. Use `bqueues -l` to display queue dispatch windows.

Deadline constraint scheduling

Deadline constraints suspend or terminate running jobs at a certain time. There are two kinds of deadline constraints:

- A run window, specified at the queue level, suspends a running job
- A termination time, specified at the job level (`bsub -t`), terminates a running job

Time-based resource usage limits

- A CPU limit, specified at job or queue level, terminates a running job when it has used up a certain amount of CPU time.
- A run limit, specified at the job or queue level, terminates a running job after it has spent a certain amount of time in the RUN state.

How deadline constraint scheduling works

If deadline constraint scheduling is enabled, LSF does not place a job that will be interrupted by a deadline constraint before its run limit expires, or before its CPU limit expires, if the job has no run limit. In this case, deadline constraint scheduling could prevent a job from ever starting. If a job has neither a run limit nor a CPU limit, deadline constraint scheduling has no effect.

A job that cannot start because of a deadline constraint causes an email to be sent to the job owner.

Deadline constraint scheduling only affects the placement of jobs. Once a job starts, if it is still running at the time of the deadline, it will be suspended or terminated because of the deadline constraint or resource usage limit.

Resizable jobs

LSF considers both job termination time and queue run windows as part of deadline constraints. Since the job has already started, LSF does not apply deadline constraint scheduling to job resize allocation requests.

Disable deadline constraint scheduling

Deadline constraint scheduling is enabled by default.

1. To disable deadline constraint scheduling for a queue, set `IGNORE_DEADLINE=y` in `lsb.queues`.

Example

LSF schedules jobs in the `liberal` queue without observing the deadline constraints.

```
Begin Queue
QUEUE_NAME = liberal
IGNORE_DEADLINE=y
End Queue
```

Advance Reservation

About advance reservations

Advance reservations ensure access to specific hosts during specified times. During the time that an advance reservation is active only users or groups associated with the reservation have access to start new jobs on the reserved hosts.

Only LSF administrators or root can create or delete advance reservations. Any LSF user can view existing advance reservations.

Each reservation consists of the number of job slots to reserve, a list of hosts for the reservation, a start time, an end time, and an owner. You can also specify a resource requirement string instead of or in addition to a list of hosts.

Active reservations

When a reservation becomes active, LSF attempts to run all jobs associated with the reservation. By default jobs running before the reservation became active continue to run when the reservation becomes active. When a job associated with the reservation is pending because not enough job slots are available, LSF suspends *all* jobs not associated with the reservation that are running on the required hosts.

During the time the reservation is active, only users or groups associated with the reservation have access to start new jobs on the reserved hosts. The reservation is active only within the time frame specified, and any given host may have several reservations in place, some of which may be active at the same time.

Jobs are suspended only if advance reservation jobs require the slots. Jobs using a reservation are subject to all job resource usage limits, but any resources freed by suspending non-advance reservation jobs are available for advance reservation jobs to use.

Closed and open reservations

Reservations are typically *closed*. When a closed reservation expires, LSF kills jobs running in the reservation and allows any jobs suspended when the reservation became active to run.

Open advance reservations allow jobs to run even after the associated reservation expires. A job in the open advance reservation is only be treated as an advance reservation job during the reservation window, after which it becomes a normal job. This prevents the job from being killed and makes sure that LSF does not prevent any previously suspended jobs from running or interfere with any existing scheduling policies.

Jobs running in a one-time open reservation are detached from the reservation and suspended when the reservation expires, allowing them to be scheduled as regular jobs. Jobs submitted before the reservation became active are still suspended when the reservation becomes active. These are only resumed after the open reservation jobs finish.

Jobs running in a closed *recurring* reservation are killed when the reservation expires.

Jobs running in an open *recurring* reservation are suspended when the reservation expires, and remain pending until the reservation becomes active again to resume.

If a non-advance reservation job is submitted while the open reservation is active, it remains pending until the reservation expires. Any advance reservation jobs that were suspended and became normal jobs when the reservation expired are resumed first before dispatching the non-advance reservation job submitted while the reservation was active.

Job scheduling in advance reservations

LSF treats advance reservation like other deadlines, such as dispatch windows or run windows; LSF does not schedule jobs that are likely to be suspended when a reservation becomes active. Jobs referencing the reservation are killed when the reservation expires.

Note:

If IGNORE_DEADLINE=Y, there is no effect on advance reservations. Jobs are always prevented from starting if there is a chance that they could encounter an advance reservation.

System reservations

Reservations can also be created for system maintenance. If a system reservation is active, no other jobs can use the reserved hosts, and LSF does not dispatch jobs to the specified hosts while the reservation is active.

Enable advance reservation

1. To enable advance reservation in your cluster, make sure the advance reservation scheduling plugin `schmod_advrsv` is configured in `lsb. modules`.

```
Begin PluginModule
SCH_PLUGIN          RB_PLUGIN          SCH_DISABLE_PHASES
schmod_default      ()                  ()
schmod_advrsv       ()                  ()
End PluginModule
```

Allow users to create advance reservations

By default, only LSF administrators or root can add or delete advance reservations. To allow other users to use `brsvadd` to create advance reservations and `brsvdel` to delete advance reservations, you need to configure advance reservation user policies.

Note:

`USER_ADVANCE_RESERVATION` in `lsb.params` is obsolete from LSF Version 8 on. Use the `ResourceReservation` section configuration in `lsb.resources` to configure advance reservation policies for your cluster.

1. Use the `ResourceReservation` section of `lsb.resources` to configure advance reservation policies for users.

A `ResourceReservation` section specifies:

- Users or user groups that can create reservations
- Hosts that can be used for the reservation
- Time window when reservations can be created.

Each advance reservation policy is defined in a separate `ResourceReservation` section, so it is normal to have multiple `ResourceReservation` sections in `lsb.resources`.

Only `user1` and `user2` can make advance reservations on `hostA` and `hostB`. The reservation time window is between 8:00 a.m. and 6:00 p.m. every day:

```
Begin ResourceReservation
```

```

NAME      = dayPol i cy
USERS     = user1 user2      # optional
HOSTS     = hostA hostB     # optional
TIME_WI N DOW = 8: 00- 18: 00      # weekly recurring reservation
End ResourceReservation

```

user1 can add the following reservation for user user2 to use on hostA every Friday between 9:00 a.m. and 11:00 a.m.:

```

brsvadd -m "hostA" -n 1 -u "user2" -t "5:9:0-5:11:0"
Reservation "user2#2" is created

```

Users can only delete reservations they created themselves. In the example, only user user1 can delete the reservation; user2 cannot. Administrators can delete any reservations created by users.

All users in user group ugroup1 except user1 can make advance reservations on any host in hgroup1, except hostB, between 10:00 p.m. and 6:00 a.m. every day

```

Begin ResourceReservation
NAME      = ni ghtPol i cy
USERS     = ugroup1 ~user1
HOSTS     = hgroup1 ~hostB
TIME_WI N DOW = 20: 00- 8: 00
End ResourceReservation

```

Important:

The not operator (~) does not exclude LSF administrators from the policy.

For example:

1. Define a policy for user: user1:

```

Policy Name: dayPol i cy
Users: user1
Hosts: hostA
Time Wi ndow: 8: 00- 18: 00

```

2. User user1 creates a reservation matching the policy (the creator is user1, the user is user2):

```

brsvadd -n 1 -m hostA -u user2 -b 10: 00 -e 12: 00
user2#0 is created.

```

3. User user1 modifies the policy to remove user1 from the users list:

```

Policy Name: dayPol i cy
Users: user3
Hosts: hostA
Time Wi ndow: 8: 00- 18: 00

```

4. As the creator, user1 can modify the reservation with the brsvmod options rmhost, -u, -o, -on, and -d, but user1 cannot add hosts or modify the time window of the reservation.

Use advance reservation

Use the following commands with advance reservations:

brsvadd

Add a reservation

brsvdel

Delete a reservation

brsvmod

Modify a reservation

brsvs

View reservations

Reservation policy checking

The following table summarizes how advance reservation commands interpret reservation policy configurations in `lsb. resources`:

The command ...		Checks policies for ...		
		Creator	Host	TimeWindow
brsvadd		Yes	Yes	Yes
brsvdel		No	No	No
brsvmod	-u or -g (changing user)	No	No	No
	addhost	Yes	Yes	Yes
	rmhost	No	No	No
	-b, -e, -t (change timeWindow)	Yes	Yes	Yes
	-d (description)	No	No	No
	-o or -on	No	No	No

Reservation policies are checked when:

- Modifying the reservation time window
- Adding hosts to the reservation

Reservation policies are *not* checked when

- Running `brsvmod` to remove hosts
- Changing the reservation type (open or closed)
- Changing users or user groups for the reservation
- Modifying the reservation description

Add reservations

Note:

By default, only LSF administrators or root can add or delete advance reservations.

1. Run `brsvadd` to create new advance reservations.

You must specify the following for the reservation:

- Number of job slots to reserve—This number should be less than or equal to the actual number of slots for the hosts defined in the reservation.
- Hosts for the reservation
- Owners of the reservation
- Time period for the reservation—either:
 - Begin time and end time for a one-time reservation, OR
 - Time window for a recurring reservation

Note:

Advance reservations should be 10 minutes or more in length. Advance reservations of less than 10 minutes may be rejected if they overlap other advance reservations in 10-minute time slots of the weekly planner.

The `brsvadd` command returns a reservation ID that you use when you submit a job that uses the reserved hosts. Any single user or user group can have a maximum of 100 reservation IDs.

Specify hosts for the reservation

1. Use one or both of the following `brsvadd` options to specify hosts for which job slots are reserved:
 - The `-m` option lists the hosts needed for the reservation. The hosts listed by the `-m` option can be local to the cluster or hosts leased from remote clusters. At job submission, LSF considers the hosts in the specified order. If you also specify a resource requirement string with the `-R` option, `-m` is optional.
 - The `-R` option selects hosts for the reservation according to a resource requirements string. Only hosts that satisfy the resource requirement expression are reserved. `-R` accepts any valid resource requirement string, but only the select string takes effect. If you also specify a host list with the `-m` option, `-R` is optional.

If `LSF_STRICT_RESREQ=Y` in `lsf.conf`, the selection string must conform to the stricter resource requirement string syntax. The strict resource requirement syntax only applies to the `select` section. It does not apply to the other resource requirement sections (`order`, `usage`, `same`, `span`, or `cu`).

Add a one-time reservation

1. Use the `-b` and `-e` options of `brsvadd` to specify the begin time and end time of a one-time advance reservation. One-time reservations are useful for dedicating hosts to a specific user or group for critical projects.

The day and time are in the form:

```
[ [year:] month:] day:] hour:minute
```

with the following ranges:

- *year*: any year after 1900 (YYYY)
- *month*: 1-12 (MM)
- *day of the month*: 1-31 (dd)
- *hour*: 0-23 (hh)
- *minute*: 0-59 (mm)

You must specify at least hour:minute. Year, month, and day are optional. Three fields are assumed to be day:hour:minute, four fields are assumed to be month:day:hour:minute, and five fields are year:month:day:hour:minute.

If you do not specify a day, LSF assumes the current day. If you do not specify a month, LSF assumes the current month. If you specify a year, you must specify a month.

You must specify a begin and an end time. The time value for -b must use the same syntax as the time value for -e. The begin time must be earlier than the time value for -e. The begin time cannot be earlier than the current time.

The following command creates a one-time advance reservation for 1024 job slots on host hostA for user user1 between 6:00 a.m. and 8:00 a.m. today:

```
brsvadd -n 1024 -m hostA -u user1 -b 6:0 -e 8:0Reservation "user1#0" is created
```

The hosts specified by -m can be local to the cluster or hosts leased from remote clusters.

The following command creates a one-time advance reservation for 1024 job slots on a host of any type for user user1 between 6:00 a.m. and 8:00 a.m. today:

```
brsvadd -n 1024 -R "type==any" -u user1 -b 6:0 -e 8:0Reservation "user1#1" is created
```

The following command creates a one-time advance reservation that reserves 12 slots on host A between 6:00 p.m. on 01 December 2003 and 6:00 a.m. on 31 January 2004:

```
brsvadd -n 12 -m hostA -u user1 -b 2003:12:01:18:00 -e 2004:01:31:06:00Reservation user1#2 is created
```

Add a recurring reservation

1. Use the -t option of brsvadd to specify a recurring advance reservation. The -t option specifies a time window for the reservation. Recurring reservations are useful for scheduling regular system maintenance jobs.

The day and time are in the form:

```
[ day: ] hour [ :minute ]
```

with the following ranges:

- *day of the week*: 0-6
- *hour*: 0-23
- *minute*: 0-59

Specify a time window one of the following ways:

- *hour-hour*
- *hour:minute-hour:minute*
- *day:hour:minute-day:hour:minute*

You must specify at least the hour. Day of the week and minute are optional. Both the start time and end time values must use the same syntax. If you do not specify a minute, LSF assumes the first minute of the hour (: 00). If you do not specify a day, LSF assumes every day of the week. If you do specify the day, you must also specify the minute.

If the current time when the reservation is created is within the time window of the reservation, the reservation becomes active immediately.

When the job starts running, the termination time of the advance reservation job is determined by the minimum of the job run limit (if specified), the queue run limit (if specified), or the duration of the reservation time window.

The following command creates an advance reservation for 1024 job slots on two hosts hostA and hostB for user group groupA every Wednesday from 12:00 midnight to 3:00 a.m.:

```
brsvadd -n 1024 -m "hostA hostB" -g groupA -t "3:0:0-3:3:0"
Reservation "groupA#0" is created
```

The following command creates an advance reservation for 1024 job slots on hostA for user user2 every weekday from 12:00 noon to 2:00 p.m.:

```
brsvadd -n 1024 -m "hostA" -u user2 -t "12:0-14:0"
Reservation "user2#0" is created
```

The following command creates a system reservation on hostA every Friday from 6:00 p.m. to 8:00 p.m.:

```
brsvadd -n 1024 -m hostA -s -t "5:18:0-5:20:0"
Reservation "system#0" is created
```

While the system reservation is active, no other jobs can use the reserved hosts, and LSF does not dispatch jobs to the specified hosts.

The following command creates an advance reservation for 1024 job slots on hosts hostA and hostB with more than 50 MB of swap space for user user2 every weekday from 12:00 noon to 2:00 p.m.:

```
brsvadd -n 1024 -R "swp > 50" -m "hostA hostB" -u user2 -t "12:0-14:0"
Reservation "user2#1" is created
```

Add an open reservation

1. Use the -o option of brsvadd to create an open advance reservation. You must specify the same information as for normal advance reservations.

The following command creates a one-time open advance reservation for 1024 job slots on a host of any type for user user1 between 6:00 a.m. and 8:00 a.m. today:

```
brsvadd -o -n 1024 -R "type==any" -u user1 -b 6:0 -e 8:0
Reservation "user1#1" is created
```

The following command creates an open advance reservation for 1024 job slots on hostB for user user3 every weekday from 12:00 noon to 2:00 p.m.:

```
brsvadd -o -n 1024 -m "hostB" -u user3 -t "12:0-14:0"
Reservation "user3#0" is created
```

Specify a reservation name

1. Use the -N option of brsvadd to specify a user-defined advance reservation name unique in an LSF cluster.

The reservation name is a string of letters, numeric characters, underscores, and dashes beginning with a letter. The maximum length of the name is 39 characters.

If no user-defined advance reservation name is specified, LSF creates the reservation with a system assigned name with the form

```
user_name#sequence
```

For example:

```
brsvadd -n 3 -M "hostA hostB" -u user2 -b 16:0 -e 17:0 -d "Production AR test"
Reservation user2#0 (Production AR test) is created
brsvadd -n 2 -N Production_AR -M hostA -u user2 -b 16:0 -e 17:0 -d "Production AR
test"
Reservation Production_AR (Production AR test) is created
```

If a job already exists that references a reservation with the specified name, an error message is returned:
The specified reservation name is referenced by a job.

Modify an advance reservation

1. Use `brsvmod` to modify reservations. Specify the reservation ID for the reservation you want to modify. For example, run the following command to extend the duration from 6:00 a.m. to 9:00 a.m.:

```
brsvmod -e "+60" user1#0
Reservation "user1#0" is modified
```

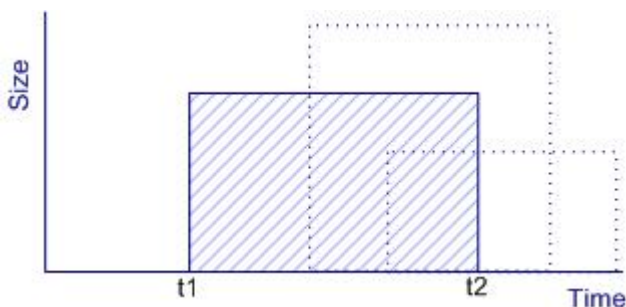
Administrators and root can modify any reservations. Users listed in the ResourceReservation section of `lsb.resouces`, can only modify reservations they created themselves.

Use brsvmod to modify advance reservations

Use `brsvmod` to make the following changes to an existing advance reservation:

- Modify start time (postpone or move closer)
- Modify the duration of the reservation window (and thus the end time)
- Modify the slot numbers required by the reservation (add or remove slots with hosts)
- Modify the host or host group list (add or remove hosts or host groups)
- Modify the user or user group
- Add hosts by resource requirement (-R)
- Modify the reservation type (open or closed)
- Disable the specified occurrences of a recurring reservation

For example, assume an advance reservation is the box between the time `t1` and `t2`, as shown in the following figure:



In this figure:

- The shadowed box shows the original reservation
- Time means the time window of the reservation
- `t1` is the begin time of the reservation
- `t2` is the end time of the reservation
- The reservation size means the resources that are reserved, such as hosts (slots) or host groups

Use `brsvmod` to shift, extend or reduce the time window horizontally; grow or shrink the size vertically.

Extend the duration

The following command creates a one-time advance reservation for 1024 job slots on host hostA for user user1 between 6:00 a.m. and 8:00 a.m. today:

```
brsvadd -n 1024 -m hostA -u user1 -b "6:0" -e "8:0"  
Reservation "user1#0" is created
```

Run the following command to extend the duration from 6:00 a.m. to 9:00 a.m.:

```
brsvmod -e "+60" user1#0  
Reservation "user1#0" is modified
```

Add hosts to a reservation allocation

Use `brsvmod` to add hosts and slots on hosts into the original advance reservation allocation. The hosts can be local to the cluster or hosts leased from remote clusters.

Adding a host without `-n` reserves all available slots on the host; that is, slots that are not already reserved by other reservations. You must specify `-n` along with `-m` or `-R`. The `-m` option can be used alone if there is no host group specified in the list. You cannot specify `-R` without `-n`.

The specified slot number must be less than or equal to the available number of job slots for the host.

You can only add hosts (`-m`) to a system reservation. You cannot add slots (`-n`) to a system reservation.

For example:

- Reserve 2 more slots from hostA:

```
brsvmod addhost -n2 -m "hostA"
```

- Reserve 4 slots in total from hostA and hostB:

```
brsvmod addhost -n4 -m "hostA hostB"
```

- Reserve 4 more slots from any Linux hosts:

```
brsvmod addhost -n4 -R "type==linux"
```

- Reserve 4 more slots from any Linux hosts in the host group hostgroup1:

```
brsvmod addhost -n4 -m "hostgroup1" -R "type==linux"
```

- Reserve all available slots from hostA and hostB:

```
brsvmod addhost -m "hostA hostB"
```

The following command creates an advance reservation for 1024 slots on two hosts hostA and hostB for user group groupA every Wednesday from 12:00 midnight to 3:00 a.m.:

```
brsvadd -n 1024 -m "hostA hostB" -g groupA -t "3:0:0-3:3:0"  
Reservation "groupA#0" is created
```

```
brsvs  
RSVID      TYPE      USER      NCPUS      RSV_HOSTS      TIME_WINDOW  
groupA#0    user      groupA     0/1024      hostA: 0/256    3: 3: 0- 3: 3: 0 *  
                                     hostB: 0/768
```

The following commands reserve 512 slots from each host for the reservation:

```
brsvmod addhost -n 256 -m "hostA" groupA#0  
Reservation "groupA#0" is modified  
brsvmod rmhost -n 256 -m "hostB" groupA#0  
Reservation "groupA#0" is modified
```

Remove hosts from a reservation allocation

Use `brsvmod rmhost` to remove hosts or slots on hosts from the original reservation allocation. You must specify either `-n` or `-m`. Use `-n` to specify the number of slots to be released from the host. Removing a host without `-n` releases all reserved slots on the host. The slot specification must be less than or equal to the actual reserved slot number of the host.

For example:

- Remove 4 reserved slots from hostA

```
brsvmod rmhost -n 4 -m "hostA"
```
- Remove 4 slots in total from hostA and hostB.

```
brsvmod rmhost -n 4 -m "hostA hostB"
```
- Release reserved hostA and hostB.

```
brsvmod rmhost -m "hostA hostB"
```
- Remove 4 slots from current reservation allocation.

```
brsvmod rmhost -n 4
```

You cannot remove slots from a system reservation. The following modification to the system reservation System#1 is rejected:

```
brsvmod rmhost -n 2 -m "hostA" system#1
```

How many slots or hosts can be removed also depends on the number of slots free while the reservation is active. `brsvmod rmhost` cannot remove more slots than free amount on a host. For example:

brsvs	RSVID	TYPE	USER	NCPUS	RSV_HOSTS	TIME_WINDOW
	user1_1	user	user1	3/4	hostA: 2/2 hostB: 1/2	1/24/12/2- 1/24/13/0

The following modifications are accepted, and one slot is removed from hostB:

```
brsvmod rmhost -m hostB user1_1
brsvmod rmhost -n 1 -m hostB user1_1
```

The following modifications are rejected:

```
brsvmod rmhost -n 2 user1_1
brsvmod rmhost -m hostA user1_1
brsvmod rmhost -n 1 -m hostA user1_1
brsvmod rmhost -n 2 -m hostB user1_1
```

Modify closed reservations

The following command creates an open advance reservation for 1024 job slots on host hostA for user user1 between 6:00 a.m. and 8:00 a.m. today.

```
brsvadd -o -n 1024 -m hostA -u user1 -b 6:0 -e 8:0
Reservation "user1#0" is created
```

Run the following command to close the reservation when it expires.

```
brsvmod -on user1#0
Reservation "user1#0" is modified
```

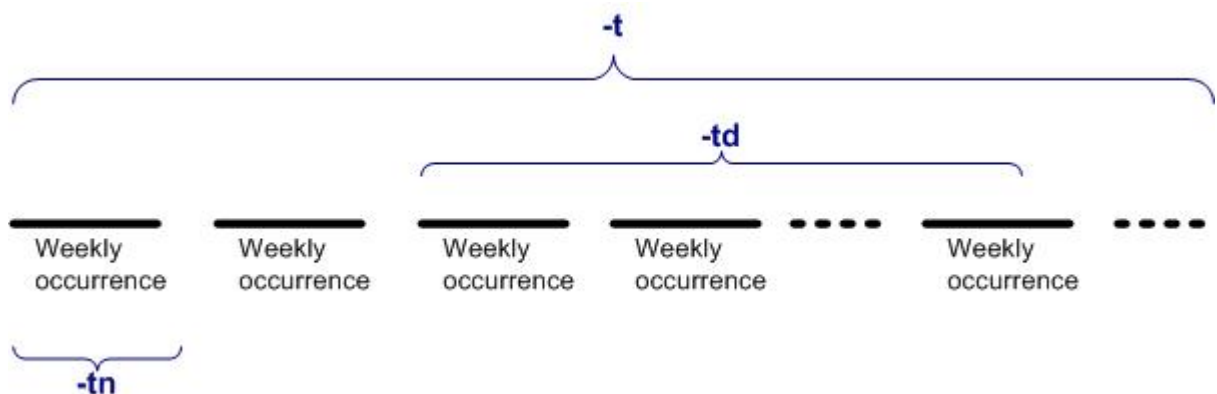
Disable specified occurrences for recurring reservations

Use `brsvmod disable` to disable specified periods, or *instances*, of a recurring advance reservation.

Recurring reservations may repeat either on a daily cycle or a weekly cycle. For daily reservations, the instances of the reservation that occur on disabled days will be inactive. Jobs using the reservation are not dispatched during on those disabled days. Other reservations are permitted to use slots of the reservation on those days. For overnight reservations (active from 11 p.m. to 9 a.m. daily), if the reservation is disabled on the starting day of an instance, the reservation is disabled for the whole of that instance.

For a weekly reservation, if the reservation is disabled on the start date of an instance of the reservation then the reservation is disabled for the entire instance. For example, for a weekly reservation with time window from 9 a.m. Wednesday to 10 p.m. Friday, in one particular week, the reservation is disabled on Thursday, then the instance of the reservation remains active for that week. However, if the same reservation is disabled for the Wednesday of the week, then the reservation is disabled for the week.

The following figure illustrates how the disable options apply to the weekly occurrences of a recurring advance reservation.



Once a reservation is disabled for a period, it cannot be enabled again; that is, the disabled periods remain fixed. Before a reservation is disabled, you are prompted to confirm whether to continue disabling the reservation. Use the `-f` option to silently force the command to run without prompting for confirmation, for example, to allow for automating disabling reservations from a script.

For example, the following command creates a recurring advance reservation for 4 slots on host `hostA` for user `user1` between 6:00 a.m. and 8:00 a.m. every day.

```
Reservation "user1#0" is created
brsvadd -n 4 -m hostA -u user1 -t "6:0-8:0"
```

Run the following command to disable the reservation instance that is active between Dec 1 to Dec 10, 2007.

```
brsvmod -disable -td "2007:12:1-2007:12:10" user1#0
Reservation "user1#0" is modified
```

Then the administrator can use host `hostA` for other reservations during the duration

```
brsvadd -n 4 -m hostA -u user1 -b "2007:12:1:6:0" -e "2007:12:1:8:0"
Reservation "user1#2" is created
```

Change users and user groups

Use `brsvmod -u` to change the user or `brsvmod -g` to change the user group that is able to submit jobs with the advance reservation.

Jobs submitted by the original user or user group to the reservation still belong to the reservation and scheduled as advance reservation jobs, but new submitted jobs from the removed user or user group cannot use the reservation any longer.

brun

An advance reservation job dispatched with `brun` is still subject to run windows and suspending conditions of the advance reservation for the job. The job must finish running before the time window of a closed reservation expires. Extending or shrinking a closed advance reservation duration prolongs or shortens lifetime of a `brun` job.

bslots

`bslots` displays a snapshot of the slots currently not in use by parallel jobs or advance reservations. If the hosts or duration of an advance reservation is modified, `bslots` recalculates and displays the available slots and available run time accordingly.

How advance reservation modifications interact

The following table summarizes how advance reservation modification applies to various advance reservation instances.

Modification...			Disable	Begin time	End Time	Add Hosts	Rm Hosts	User/ Usergroup	open / closed	Pre cmd	Post cmd
One-time	Active		No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Inactive		No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Recurring	Occurrences	All	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
		Specified	Yes	No	No	No	No	No	No	No	No
	Active instance		No	No	No	No	No	No	No	No	No

Where: "Yes" means the modification is supported; otherwise, "No" is marked. For example, all modifications are acceptable in the case that the advance reservation is inactive (and not disabled).

Remove an advance reservation

1. Use `brsvdel` to delete reservations. Specify the reservation ID for the reservation you want to delete.

For example:

```
brsvdel user1#0Reservation user1#0 is being deleted
```

You can delete more than one reservation at a time. Administrators can delete any reservation, but users may only delete their own reservations.

If the recurring reservation is deleted with `brsvdel`, jobs running in the reservation are detached from the reservation and scheduled as normal jobs.

View reservations

1. Use `brsvs` to show current reservations:

brsvs					
RSVID	TYPE	USER	NCPUS	RSV_HOSTS	TIME_WINDOW
user1#0	user	user1	0/1024	hostA: 0/1024	11/12/6/0- 11/12/8/0
user2#0	user	user2	0/1024	hostA: 0/1024	12: 0- 14: 0 *
groupA#0	group	groupA	- /2048	hostA: - /1024 hostB: 0/1024	3: 0- 0- 3: 3: 0 *
system#0	sys	system	1024	hostA: 0/1024	5: 18: 0- 5: 20: 0 *

In the TIME_WINDOW column:

- A one-time reservation displays fields separated by slashes (month/day/hour/minute). For example:
11/12/14/0- 11/12/18/0
- A recurring reservation displays fields separated by colons (day: hour: minute). An asterisk (*) indicates a recurring reservation. For example:
5: 18: 0- 5: 20: 0 *

In the NCPUS and RSV_HOSTS columns:

- Remote reservations do not display details. For example:

```
- /2048      hostA: - /1024
```

Show a weekly planner

- Use `brsvs -p` to show a weekly planner for specified hosts using advance reservation. The `all` keyword shows the planner for all hosts with reservations.

The output of `brsvs -p` is displayed in terms of weeks. The week starts on Sunday. The timeframe of a recurring reservation is not displayed, since it is unlimited. The timeframe of one-time reservation is displayed in terms of a week. If the reservation spans multiple weeks, these weeks are displayed separately. If a week contains a one-time reservation and a recurring reservation, the timeframe is displayed, since that is relevant for one-time reservation.

Tip:

MAX indicates the configured maximum number of job slots for the host (MXJ defined in `lsb. host.s`).

```
brsvs -p all
RSVID      TYPE    USER    NCPUS    RSV_HOSTS    TIME_WINDOW
user1#0     user    user1    0/1024    hostA: 0/1024  11/12/6/0- 11/12/8/0
user2#0     user    user2    0/1024    hostA: 0/1024  12: 0- 14: 0 *
groupA#0    group   groupA    0/2048    hostA: 0/1024  3: 0- 3: 3: 0 *
                                hostB: 0/1024
system#0     sys     system    1024     hostA: 0/1024  5: 18: 0- 5: 20: 0 *
HOST: hostA (MAX = 1024)
Week: 11/11/2009 - 11/17/2009
Hour: Mi n   Sun    Mon    Tue    Wed    Thu    Fri    Sat
-----
0: 0         0      0      0      1024   0      0      0
0: 10        0      0      0      1024   0      0      0
0: 20        0      0      0      1024   0      0      0
...
2: 30        0      0      0      1024   0      0      0
2: 40        0      0      0      1024   0      0      0
2: 50        0      0      0      1024   0      0      0
3: 0         0      0      0      0      0      0      0
3: 10        0      0      0      0      0      0      0
3: 20        0      0      0      0      0      0      0
...
5: 30        0      0      0      0      0      0      0
5: 40        0      0      0      0      0      0      0
5: 50        0      0      0      0      0      0      0
6: 0         0      1024   0      0      0      0      0
6: 10        0      1024   0      0      0      0      0
6: 20        0      1024   0      0      0      0      0
...
7: 30        0      1024   0      0      0      0      0
7: 40        0      1024   0      0      0      0      0
7: 50        0      1024   0      0      0      0      0
8: 0         0      0      0      0      0      0      0
8: 10        0      0      0      0      0      0      0
8: 20        0      0      0      0      0      0      0
...
11: 30       0      0      0      0      0      0      0
11: 40       0      0      0      0      0      0      0
11: 50       0      0      0      0      0      0      0
12: 0        1024   1024   1024   1024   1024   1024   1024
12: 10       1024   1024   1024   1024   1024   1024   1024
12: 20       1024   1024   1024   1024   1024   1024   1024
...
13: 30       1024   1024   1024   1024   1024   1024   1024
13: 40       1024   1024   1024   1024   1024   1024   1024
13: 50       1024   1024   1024   1024   1024   1024   1024
14: 0        0      0      0      0      0      0      0
14: 10       0      0      0      0      0      0      0
```

14: 20	0	0	0	0	0	0	0
...							
17: 30	0	0	0	0	0	0	0
17: 40	0	0	0	0	0	0	0
17: 50	0	0	0	0	0	0	0
18: 0	0	0	0	0	0	1024	0
18: 10	0	0	0	0	0	1024	0
18: 20	0	0	0	0	0	1024	0
...							
19: 30	0	0	0	0	0	1024	0
19: 40	0	0	0	0	0	1024	0
19: 50	0	0	0	0	0	1024	0
20: 0	0	0	0	0	0	0	0
20: 10	0	0	0	0	0	0	0
20: 20	0	0	0	0	0	0	0
...							
23: 30	0	0	0	0	0	0	0
23: 40	0	0	0	0	0	0	0
23: 50	0	0	0	0	0	0	0
HOST: hostB (MAX = 1024)							
Week: 11/11/2009 - 11/17/2009							
Hour: Mi n	Sun	Mon	Tue	Wed	Thu	Fri	Sat

0: 0	0	0	0	1024	0	0	0
0: 10	0	0	0	1024	0	0	0
0: 20	0	0	0	1024	0	0	0
...							
2: 30	0	0	0	1024	0	0	0
2: 40	0	0	0	1024	0	0	0
2: 50	0	0	0	1024	0	0	0
3: 0	0	0	0	0	0	0	0
3: 10	0	0	0	0	0	0	0
3: 20	0	0	0	0	0	0	0
...							
23: 30	0	0	0	0	0	0	0
23: 40	0	0	0	0	0	0	0
23: 50	0	0	0	0	0	0	0

- Use `brsvs -z` instead of `brsvs -p` to show only the weekly items that have reservation configurations. Lines that show all zero (0) are omitted.

For example:

brsvs -z all							
RSVID	TYPE	USER	NCPUS	RSV_HOSTS	TIME_WINDOW		
user1_1	user	user1	0/3	hostA: 0/2 hostB: 0/1	12/28/14/30- 12/28/15/30		
HOST: hostA (MAX = 2)							
Week: 12/23/2007 - 12/29/2007							
Hour: Mi n	Sun	Mon	Tue	Wed	Thu	Fri	Sat

14: 30	0	0	0	0	0	1	0
14: 40	0	0	0	0	0	1	0
14: 50	0	0	0	0	0	1	0
15: 0	0	0	0	0	0	1	0
15: 10	0	0	0	0	0	1	0
15: 20	0	0	0	0	0	1	0
HOST: hostB (MAX = 2)							
Week: 12/23/2007 - 12/29/2007							
Hour: Mi n	Sun	Mon	Tue	Wed	Thu	Fri	Sat

14: 30	0	0	0	0	0	2	0
14: 40	0	0	0	0	0	2	0
14: 50	0	0	0	0	0	2	0
15: 0	0	0	0	0	0	2	0
15: 10	0	0	0	0	0	2	0
15: 20	0	0	0	0	0	2	0

Show reservation types and associated jobs

- Use the `-l` option of `brsvs` to show each advance reservation in long format.

The rows that follow the reservation information show the

- The status of the reservation
- Time when the next instance of recurring reservation is active
- Type of reservation (open or closed)
- The status by job ID of any job associated with the specified reservation (FINISHED, PEND, RUN, or SUSP)

```
brsvs -l
RSVID      TYPE      USER      NCPUS      RSV_HOSTS      TIME_WINDOW
user1_1#0   user      user1_1    10/10       host1: 4/4      8: 00- 22: 00 *
             host2: 4/4
             host3: 2/2

Reservation Status: Active
Next Active Period:
    Sat Aug 22 08:00:00 2009 - Sat Aug 22 22:00:00 2009
Creator: user1_1
Reservation Type: CLOSED
FINISHED Jobs: 203 204 205 206 207 208 209 210 211 212
PEND Jobs: 323 324
RUN Jobs: 313 314 316 318 319 320 321 322
SUSP Jobs: 315 317
```

Show reservation ID

1. Use `bjobs -l` to show the reservation ID used by a job:

```
bjobs -l
Job <1152>, User <user1>, Project <default>, Status <PEND>, Queue <normal>,
Reservation <user1#0>, Command <myjob>
Mon Nov 12 5:13:21 2009: Submitted from host <hostB>, CWD </home/user1/jobs>;
```

View historical accounting information for advance reservations

1. Use the `-U` option of the `bacct` command to display accounting information about advance reservations.

`bacct -U` summarizes all historical modification of the reservation and displays information similar to the `brsvs` command:

- The reservation ID specified on the `-U` option.
- The type of reservation: `user` or `system`
- The user names of users who used the `brsvadd` command to create the advance reservations
- The user names of the users who can use the advance reservations (with `bsub -U`)
- Number of slots reserved
- List of hosts for which job slots are reserved
- Time window for the reservation.
 - A one-time reservation displays fields separated by slashes (`month/day/hour/minute`). For example:


```
11/12/14/0-11/12/18/0
```
 - A recurring reservation displays fields separated by colons (`day: hour: minute`). For example:


```
5: 18: 0 5: 20: 0
```

For example, the following advance reservation has four time modifications during its life time. The original reservation has the scope of one user (`user1`) and one host (`hostA`) with 1 slot. The various modifications change the user to `user2`, then back to `user1`, adds, then removes 1 slot from the reservation.

```
bacct -U user1#1
```

Accounting about advance reservations that are:

- accounted on advance reservation IDs user1#1,
- accounted on advance reservations created by user1,

```
----- SUMMARY -----
RSVID:                user1#1
TYPE:                 user
CREATOR:              user1
Total number of jobs: 0
Total CPU time consumed: 0.0 second
Maximum memory of a job: 0.0 MB
Maximum swap of a job: 0.0 MB
Total active time:    0 hour 6 minute 42 second
----- Configuration 0 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#1   user    user1    user1    1        hostA: 1
Active time with this configuration: 0 hour 0 minute 16 second
----- Configuration 1 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#1   user    user1    user2    1        hostA: 1
Active time with this configuration: 0 hour 0 minute 24 second
----- Configuration 2 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#1   user    user1    user2    1        hostA: 1
Active time with this configuration: 0 hour 1 minute 58 second
----- Configuration 3 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#1   user    user1    user1    2        hostA: 2
Active time with this configuration: 0 hour 1 minute 34 second
----- Configuration 4 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#1   user    user1    user1    1        hostA: 2
Active time with this configuration: 0 hour 2 minute 30 second
```

The following reservation (user2#0) has one time modification during its life time. The original one has the scope of one user (user2) and one host (hostA) with 1 slot; the modification changes the user to user3.

bacct -U user2#0

Accounting about advance reservations that are:

- accounted on all advance reservation IDs;
- accounted on advance reservations created by all users;

```
----- SUMMARY -----
RSVID:                user2#0
TYPE:                 user
CREATOR:              user2
Total number of jobs: 1
Total CPU time consumed: 5.0 second
Maximum memory of a job: 1.7 MB
Maximum swap of a job: 7.5 MB
Total active time:    2 hour 0 minute 0 second
----- Configuration 0 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#0   user    user2    user2    1        hostA: 1
Active time with this configuration: 1 hour 0 minute 0 second
----- Configuration 1 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#0   user    user2    user3    1        hostA: 1
Active time with this configuration: 1 hour 0 minute 0 second
```

Submit and modify jobs using advance reservations

1. Use the -U option of bsub to submit jobs with a reservation ID. For example:

bsub -U user1#0 myjob

The job can only use hosts reserved by the reservation user1#0. By default, LSF selects only hosts in the reservation. Use the -m option to specify particular hosts within the list of hosts reserved by the reservation; you can only select from hosts that were included in the original reservation.

If you do not specify hosts (`bsub -m`) or resource requirements (`bsub -R`), the default resource requirement is to select hosts that are of any host type (LSF assumes "`type==any`" instead of "`type==local`" as the default select string).

If you later delete the advance reservation while it is still active, any pending jobs still keep the "`type==any`" attribute.

A job can only use one reservation. There is no restriction on the number of jobs that can be submitted to a reservation; however, the number of slots available on the hosts in the reservation may run out. For example, reservation `user2#0` reserves 1024 slots on host A. When all 1024 slots on host A are used by jobs referencing `user2#0`, host A is no longer available to other jobs using reservation `user2#0`. Any single user or user group can have a maximum of 100 reservation IDs.

Jobs referencing the reservation are killed when the reservation expires.

Modify job reservation ID

You must be an administrator to perform this task.

1. Use the `-U` option of `bmod` to change a job to another reservation ID.

For example:

```
bmod -U user1#0 1234
```

2. To cancel the reservation, use the `-Un` option of `bmod`.

For example:

```
bmod -Un 1234
```

Use `bmod -Un` to detach a running job from an *inactive* open reservation. Once detached, the job is scheduled like a normal job.

Advance reservation behavior

Job resource usage limits and job chunking

A job using a reservation is subject to all job resource usage limits. If a limit is reached on a particular host in a reservation, jobs using that reservation cannot start on that host.

An advance reservation job is dispatched to its reservation even if the run limit or estimated run time of the job exceeds the remaining active time of the reservation. For example, if a job has a `runlimit` of 1 hour, and a reservation has a remaining active time of 1 minute, the job is still dispatched to the reservation. If the reservation is closed, the job is terminated when the reservation expires.

Similarly, when using chunk job scheduling, advance reservation jobs are chunked together as usual when dispatched to a host of the reservation without regard to the expiry time of the reservation. This is true even when the jobs are given a run limit or estimated run time. If the reservation is closed, the jobs in `WAIT` state are terminated when the reservation expires.

Advance reservation preemption

Advance reservation preemption allows advance reservation jobs to use the slots reserved by the reservation. Slots occupied by non-advance jobs may be preempted when the reservation becomes active.

Without modification with `brsvmod`, advance reservation preemption is triggered at most once per reservation period (in the case of a non-recurring reservation, there is only one period) whenever *both* of the following conditions are met:

- The reservation is active

- At least one job associated with the advance reservation is pending or suspended

If an advance reservation is modified, preemption is done for an active advance reservation after every modification of the reservation when there is at least one pending or suspended job associated with the reservation.

When slots are added to an advance reservation with `brsvmod`, LSF preempts running non-reservation jobs if necessary to provide slots for jobs belonging to the reservation. Preemption is triggered if there are pending or suspended jobs belonging to the reservation in the system.

When preemption is triggered, non-advance reservation jobs are suspended and their slots given to the advance reservation on the hosts belonging to the reservation. On each host, enough non-advance reservation jobs are suspended so that all of slots required by the advance reservation are obtained. The number of slots obtained does not depend on the number of jobs submitted to the advance reservation. Non-advance reservation jobs on a host can only to use slots not assigned to the advance reservation.

When a job is preempted for an advance reservation, it can only resume on the host when either the advance reservation finishes, or some other non-advance reservation job finishes on the host.

For example, a single-host cluster has 10 slots, with 9 non-advance reservation jobs dispatched to the host (each requiring one slot). An advance reservation that uses 5 slots on the host is created, and a single job is submitted to the reservation. When the reservation becomes active, 4 of the non-advance reservation jobs are suspended, and the advance reservation job will start.

Force a job to run before a reservation is active

LSF administrators can use `brun` to force jobs to run before the reservation is active, but the job must finish running before the time window of the reservation expires.

For example, if the administrator forces a job with a reservation to run one hour before the reservation is active, and the reservation period is 3 hours, a 4 hour run limit takes effect.

Host intersection and advance reservation

When `ENABLE_HOST_INTERSECTION=y` in `lsb.params`, LSF finds any existing intersection with hosts specified in the queue and those specified at job submission by `bsub -m` and/or hosts with advance reservation. When specifying keywords such as `all`, `allremote`, and others, LSF finds an existing intersection of hosts available and the job runs rather than being rejected.

Advance reservations across clusters

You can create and use advance reservation for the MultiCluster job forwarding model. To enable this feature, you must upgrade all clusters to LSF Version 8 or later.

See the *Using Platform LSF MultiCluster* for more information.

Resizable jobs and advance reservations

Like regular jobs, resizable jobs associated with an advance reservation can be dispatched only after the reservation becomes active, and the minimum processor request can be satisfied. The allocation request is treated like a regular advance reservation job, which relies on slots available to the reservation. If an advance reservation gets more resources by modification (`brsvmod addhost`), those resources can be used by pending allocation requests immediately.

The following table summarizes the relationship of the AR lifecycle and resizable job requests:

Advance Reservation	Resizable job		Allocation request
One-time expired/ deleted	Open	RUN->SSUSP->RUN	Postponed until the job runs
	Closed	Removed	Removed
Recurrent expired/ deleted	Open	SSUSP till next instance	Postponed until the job runs again in next instance
	Closed	Removed	Removed

By the time a reservation has expired or deleted, the status change of the resizable job to SSUSP blocks a resizable job allocation request from being scheduled.

Released slots from a resizable job can be reused by other jobs in the reservation.

Resizable advance reservation jobs can preempt non-advance reservation jobs that are consuming the slots that belong to the reservation. Higher priority advance reservation jobs can preempt low priority advance reservation jobs, regardless of whether both are resizable jobs.

Allocation requests of resizable AR jobs honor limits configuration. They cannot preempt any limit tokens from other jobs.

Compute units and advance reservations

Like regular jobs, jobs with compute unit resource requirements and an advance reservation can be dispatched only after the reservation becomes active, and the minimum processor request can be satisfied.

In the case of exclusive compute unit jobs (with the resource requirement `cu[excl]`), the advance reservation can affect hosts outside the advance reservation but in the same compute unit as follows:

- An exclusive compute unit job dispatched to a host inside the advance reservation will lock the entire compute unit, including any hosts outside the advance reservation.
- An exclusive compute unit job dispatched to a host outside the advance reservation will lock the entire compute unit, including any hosts inside the advance reservation.

Ideally all hosts belonging to a compute unit should be inside or outside of an advance reservation.



Job Scheduling Policies

Preemptive Scheduling

The preemptive scheduling feature allows a pending high-priority job to preempt a running job of lower priority. The lower-priority job is suspended and is resumed as soon as possible. Use preemptive scheduling if you have long-running, low-priority jobs causing high-priority jobs to wait an unacceptably long time.

About preemptive scheduling

Preemptive scheduling takes effect when two jobs compete for the same job slots. If a high-priority job is pending, LSF can suspend a lower-priority job that is running, and then start the high-priority job instead. For this to happen, the high-priority job must be pending in a *preemptive queue* (a queue that can preempt other queues), or the low-priority job must belong to a *preemptable queue* (a queue that can be preempted by other queues).

If multiple slots are required, LSF can preempt multiple jobs until sufficient slots are available. For example, one or more jobs can be preempted for a job that needs multiple job slots.

A preempted job is resumed as soon as more job slots become available; it does not necessarily have to wait for the preempting job to finish.

Preemptive queue	Jobs in a preemptive queue can preempt jobs in any queue of lower priority, even if the lower-priority queues are not specified as preemptable. Preemptive queues are more aggressive at scheduling jobs because a slot that is not available to a low-priority queue may be available by preemption to a high-priority queue.
Preemptable queue	Jobs in a preemptable queue can be preempted by jobs from any queue of a higher priority, even if the higher-priority queues are not specified as preemptive. When multiple preemptable jobs exist (low-priority jobs holding the required slots), and preemption occurs, LSF preempts a job from the least-loaded host.

Resizable jobs

Resize allocation requests are not able take advantage of the queue-based preemption mechanism to preempt other jobs. However, regular pending jobs are still able to preempt running resizable jobs, even while they have a resize request pending. When a resizable job is preempted and goes to the SSUSP state, its resize request remains pending and LSF stops scheduling it until it returns back to RUN state.

- New pending allocation requests cannot make use of preemption policy to get slots from other running or suspended jobs.
- Once a resize decision has been made, LSF updates its job counters to be reflected in future preemption calculations. For instance, resizing a running preemptable job from 2 slots to 4 slots, makes 4 preemptable slots for high priority pending jobs.
- If a job is suspended, LSF stops allocating resources to a pending resize request.
- When a preemption decision is made, if job has pending resize request and scheduler already has made an allocation decision for this request, LSF cancels the allocation decision.
- If a preemption decision is made while a job resize notification command is running, LSF prevents the suspend signal from reaching the job.

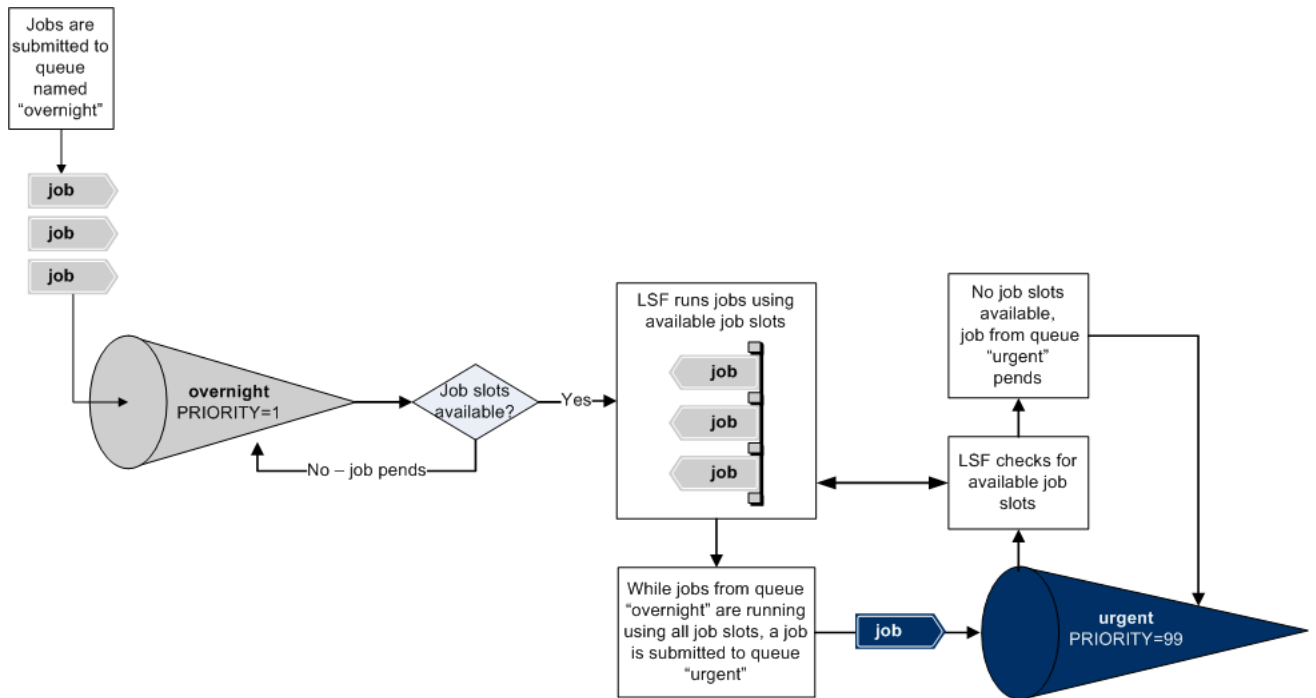
Scope

By default, preemptive scheduling does not apply to jobs that have been forced to run (using `brun`) or backfill and exclusive jobs.

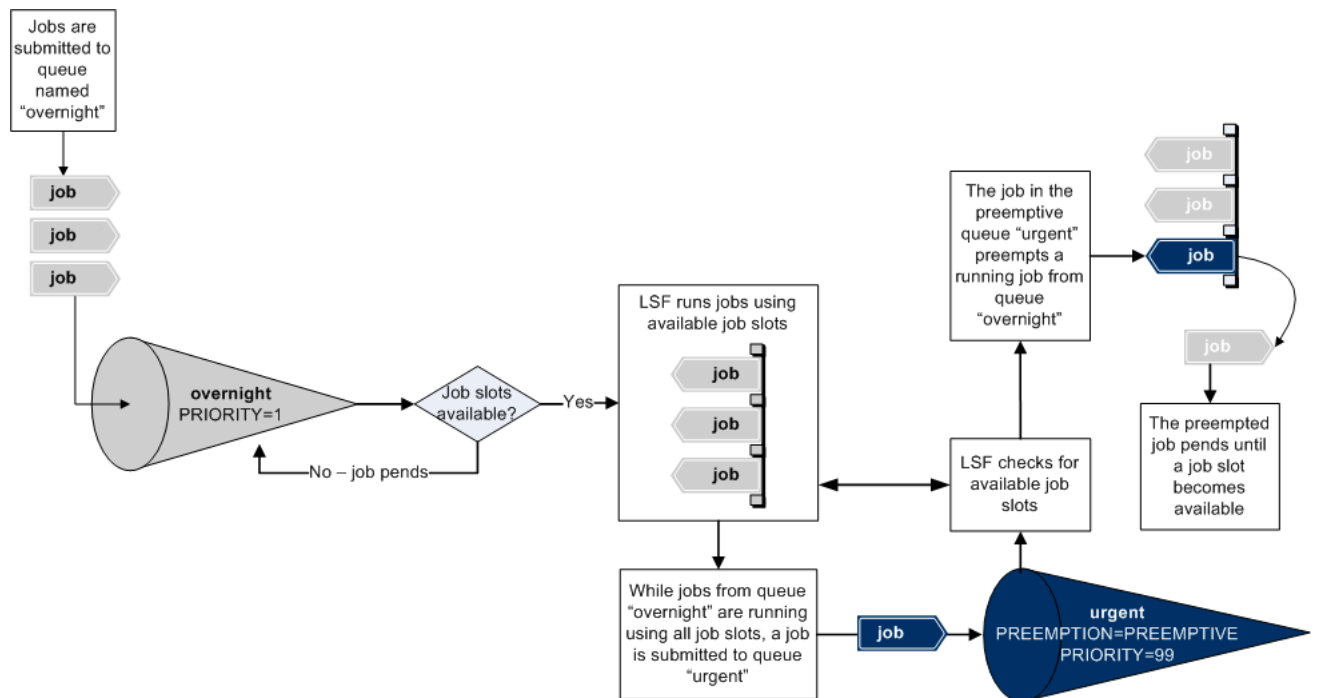
Limitations	Description
Exclusive jobs	Jobs requesting exclusive use of resources cannot preempt other jobs. Jobs using resources exclusively cannot be preempted.

Limitations	Description
Backfill jobs	Jobs backfilling future advance reservations cannot be preempted.
brun	Jobs forced to run with the command brun cannot be preempted.

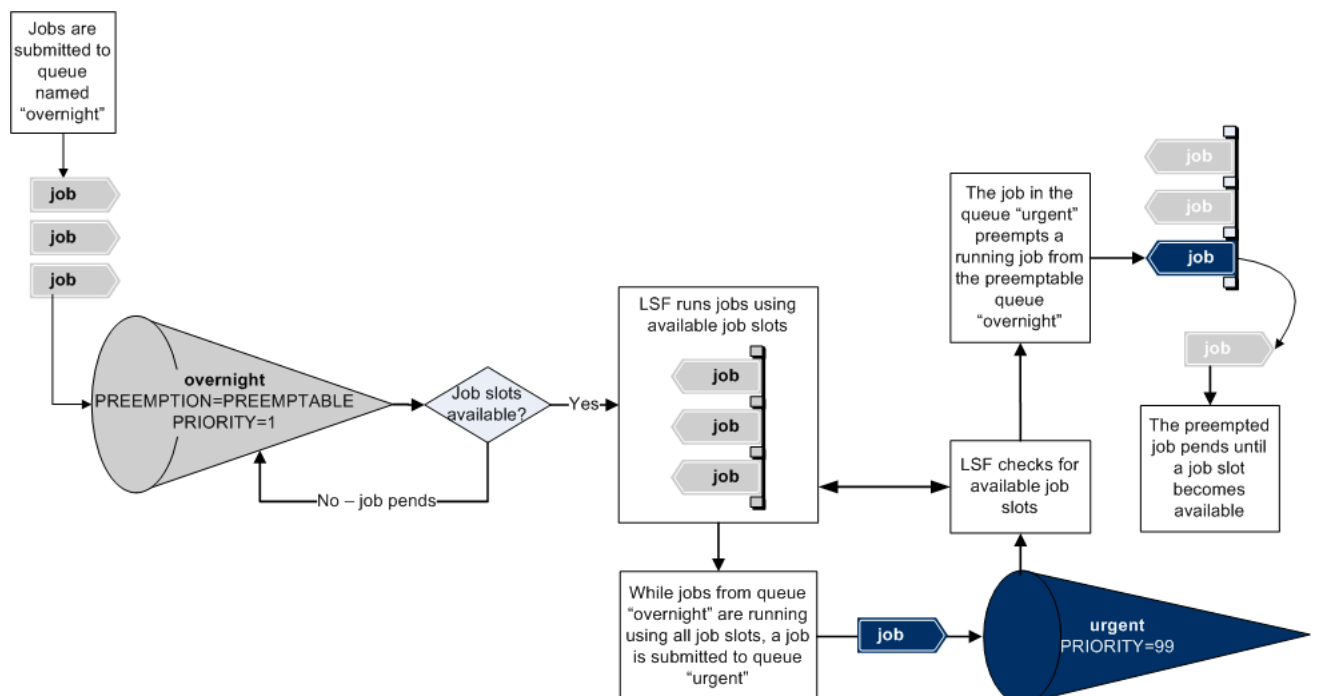
Default behavior (preemptive scheduling not enabled)



With preemptive scheduling enabled (preemptive queue)



With preemptive scheduling enabled (preemptable queue)



Configuration to enable preemptive scheduling

The preemptive scheduling feature is enabled by defining at least one queue as preemptive or preemptable, using the `PREEMPTION` parameter in the `lsb.queues` file. Preemption does not actually occur until at least one queue is assigned a higher relative priority than another queue, using the `PRIORITY` parameter, which is also set in the `lsb.queues` file.

Both `PREEMPTION` and `PRIORITY` are used to determine which queues can preempt other queues, either by establishing relative priority of queues or by specifically defining preemptive properties for a queue.

Configuration file	Parameter and syntax	Default behavior
lsb.queues	<code>PREEMPTION=PREEMPTIVE</code>	<ul style="list-style-type: none"> Enables preemptive scheduling Jobs in this queue can preempt jobs in any queue of lower priority, even if the lower-priority queue is not specified as preemptable
	<code>PREEMPTION=PREEMPTABLE</code>	<ul style="list-style-type: none"> Enables preemptive scheduling Jobs in this queue can be preempted by jobs from any queue of higher priority, even if the higher-priority queue is not specified as preemptive
	<code>PRIORITY=<i>integer</i></code>	<ul style="list-style-type: none"> Sets the priority for this queue relative to all other queues The larger the number, the higher the priority—a queue with <code>PRIORITY=99</code> has a higher priority than a queue with <code>PRIORITY=1</code>

Preemptive scheduling behavior

Preemptive scheduling is based primarily on parameters specified at the queue level: some queues are eligible for preemption, others are not. Once a hierarchy of queues has been established, other factors determine which jobs from a queue should be preempted.

There are three ways to establish which queues should be preempted:

- Based on queue priority—the `PREEMPTION` parameter defines a queue as preemptive or preemptable and preemption is based on queue priority, where jobs from higher-priority queues can preempt jobs from lower-priority queues
- Based on a preferred order—the `PREEMPTION` parameter defines queues that can preempt other queues, in a preferred order
- Explicitly, by specific queues—the `PREEMPTION` parameter defines queues that can be preempted, and by which queues

When...	The behavior is ...
Preemption is not enabled—no queue is defined as preemptable, and no queue is defined as preemptive	<ul style="list-style-type: none"> • High-priority jobs do not preempt jobs that are already running
A queue is defined as preemptable, but no specific queues are listed that can preempt it	<ul style="list-style-type: none"> • Jobs from this queue can be preempted by jobs from any queue with a higher value for priority
A queue is defined as preemptable, and one or more queues are specified that can preempt it	<ul style="list-style-type: none"> • Jobs from this queue can be preempted only by jobs from the specified queues
A queue is defined as preemptive, but no specific queues are listed that it can preempt	<ul style="list-style-type: none"> • Jobs from this queue preempt jobs from all queues with a lower value for priority • Jobs are preempted from the least-loaded host
A queue is defined as preemptive, and one or more specific queues are listed that it can preempt, but no queue preference is specified	<ul style="list-style-type: none"> • Jobs from this queue preempt jobs from any queue in the specified list • Jobs are preempted on the least-loaded host first
A queue is defined as preemptive, and one or more queues have a preference number specified, indicating a preferred order of preemption	<ul style="list-style-type: none"> • Queues with a preference number are preferred for preemption over queues without a preference number • Queues with a higher preference number are preferred for preemption over queues with a lower preference number • For queues that have the same preference number, the queue with lowest priority is preferred for preemption over queues with higher priority • For queues without a preference number, the queue with lower priority is preferred for preemption over the queue with higher priority
A queue is defined as preemptive, or a queue is defined as preemptable, and preemption of jobs with the shortest run time is configured	<ul style="list-style-type: none"> • A queue from which to preempt a job is determined based on other parameters as shown above • The job that has been running for the shortest period of time is preempted

When...	The behavior is ...
A queue is defined as preemptive, or a queue is defined as preemptable, and preemption of jobs that will finish within a certain time period is prevented	<ul style="list-style-type: none"> A queue from which to preempt a job is determined based on other parameters as shown above A job that has a run limit or a run time specified and that will not finish within the specified time period is preempted
A queue is defined as preemptive, or a queue is defined as preemptable, and preemption of jobs with the specified run time is prevented	<ul style="list-style-type: none"> A queue from which to preempt a job is determined based on other parameters as shown above The job that has been running for less than the specified period of time is preempted

Case study: Three queues with varying priority

Consider the case where three queues are defined as follows:

Queue A has the highest relative priority, with a value of 99

Queue B is both preemptive and preemptable, and has a relative priority of 10

Queue C has the lowest relative priority, with the default value of 1

The queues can preempt as follows:

- A can preempt B because B is preemptable and B has a lower priority than A
- B can preempt C because B is preemptive and C has a lower priority than B
- A cannot preempt C, even though A has a higher priority than C, because A is not preemptive, nor is C preemptable

Calculation of job slots in use

The number of job slots in use determines whether preemptive jobs can start. The method in which the number of job slots in use is calculated can be configured to ensure that a preemptive job can start. When a job is preempted, it is suspended. If the suspended job still counts towards the total number of jobs allowed in the system, based on the limits imposed in the `lsb. resources` file, suspending the job may not be enough to allow the preemptive job to run.

The `PREEMPT_FOR` parameter is used to change the calculation of job slot usage, ignoring suspended jobs in the calculation. This ensures that if a limit is met, the preempting job can actually run.

When...	The effect on the calculation of job slots used is ...
Preemption is not enabled	<ul style="list-style-type: none"> Job slot limits are enforced based on the number of job slots taken by both running and suspended jobs. Job slot limits specified at the queue level are enforced for both running and suspended jobs.

When...	The effect on the calculation of job slots used is ...
Preemption is enabled	<ul style="list-style-type: none"> The total number of jobs at both the host and individual user level is not limited by the number of suspended jobs—only running jobs are considered. The number of running jobs never exceeds the job slot limits. If starting a preemptive job violates a job slot limit, a lower-priority job is suspended to run the preemptive job. If, however, a job slot limit is still violated (i.e. the suspended job still counts in the calculation of job slots in use), the preemptive job still cannot start. Job slot limits specified at the queue level are always enforced for both running and suspended jobs. When preemptive scheduling is enabled, suspended jobs never count against the total job slot limit for individual users.
Preemption is enabled, and PREEMPT_FOR=GROUP_JLP	<ul style="list-style-type: none"> Only running jobs are counted when calculating the per-processor job slots in use for a user group, and comparing the result with the limit specified at the user level.
Preemption is enabled, and PREEMPT_FOR=GROUP_MAX	<ul style="list-style-type: none"> Only running jobs are counted when calculating the job slots in use for this user group, and comparing the result with the limit specified at the user level.
Preemption is enabled, and PREEMPT_FOR=HOST_JLU	<ul style="list-style-type: none"> Only running jobs are counted when calculating the total job slots in use for a user group, and comparing the result with the limit specified at the host level. Suspended jobs do not count against the limit for individual users.
Preemption is enabled, and PREEMPT_FOR=USER_JLP	<ul style="list-style-type: none"> Only running jobs are counted when calculating the per-processor job slots in use for an individual user, and comparing the result with the limit specified at the user level.

Preemption of backfill jobs

With preemption of backfill jobs enabled (`PREEMPT_JOBTYPE=BACKFILL` in `l sb. params`), LSF maintains the priority of jobs with resource or slot reservations by preventing lower-priority jobs that preempt backfill jobs from "stealing" resources from jobs with reservations. Only jobs from queues with a higher priority than queues that define resource or slot reservations can preempt backfill jobs. For example,

If ...	Is configured ...	And a priority of ...	The behavior is ...
queueR	With a resource or slot reservation	80	Jobs in this queue reserve resources. If backfill scheduling is enabled, backfill jobs with a defined run limit can use the resources.
queueB	As a preemptable backfill queue	50	Jobs in queueB with a defined run limit use job slots reserved by jobs in queueR.
queueP	As a preemptive queue	75	Jobs in this queue do not necessarily have a run limit. LSF prevents jobs from this queue from preempting backfill jobs because queueP has a lower priority than queue R.

To guarantee a minimum run time for interruptible backfill jobs, LSF suspends them upon preemption. To change this behavior so that LSF terminates interruptible backfill jobs upon preemption, you must define the parameter `TERMINATE_WHEN=PREEMPT` in `l sb. queues`.

Configuration to modify preemptive scheduling behavior

There are configuration parameters that modify various aspects of preemptive scheduling behavior, by

- Modifying the selection of the queue to preempt jobs from
- Modifying the selection of the job to preempt
- Modifying preemption of backfill and exclusive jobs
- Modifying the way job slot limits are calculated
- Modifying the number of jobs to preempt for a parallel job
- Modifying the control action applied to preempted jobs
- Control how many times a job can be preempted

Configuration to modify selection of queue to preempt

File	Parameter	Syntax and description
lsb. queues	PREEMPTION	<div>PREEMPTION=PREEMPTIVE[low_queue+pref ...]</div> <ul style="list-style-type: none">• Jobs in this queue can preempt running jobs from the specified queues, starting with jobs in the queue with the highest value set for preference
		<div>PREEMPTION=PREEMPTABLE[hi_queue ...]</div> <ul style="list-style-type: none">• Jobs in this queue can be preempted by jobs from the specified queues
	PRIORITY= <i>integer</i>	<ul style="list-style-type: none">• Sets the priority for this queue relative to all other queues• The higher the priority value, the more likely it is that jobs from this queue may preempt jobs from other queues, and the less likely it is for jobs from this queue to be preempted by jobs from other queues

Configuration to modify selection of job to preempt

Files	Parameter	Syntax and description
l sb. params	PREEMPT_FOR	PREEMPT_FOR=LEAST_RUN_TIME
l sb. appl i cat i on s		<ul style="list-style-type: none"> Preempts the job that has been running for the shortest time
	NO_PREEMPT_RUN_TIME	NO_PREEMPT_RUN_TIME=%
		<ul style="list-style-type: none"> Prevents preemption of jobs that have been running for the specified percentage of minutes, or longer If NO_PREEMPT_RUN_TIME is specified as a percentage, the job cannot be preempted after running the percentage of the job duration. For example, if the job run limit is 60 minutes and NO_PREEMPT_RUN_TIME=50%, the job cannot be preempted after it running 30 minutes or longer. If you specify percentage for NO_PREEMPT_RUN_TIME, requires a run time (bsub - We or RUNTIME in l sb. appl i cat i ons), or run limit to be specified for the job (bsub - W, or RUNLIMIT in l sb. queues, or RUNLIMIT in l sb. appl i cat i ons)
	NO_PREEMPT_FINISH_TIME	NO_PREEMPT_FINISH_TIME=%
	E	<ul style="list-style-type: none"> Prevents preemption of jobs that will finish within the specified percentage of minutes. If NO_PREEMPT_FINISH_TIME is specified as a percentage, the job cannot be preempted if the job finishes within the percentage of the job duration. For example, if the job run limit is 60 minutes and NO_PREEMPT_FINISH_TIME=10%, the job cannot be preempted after it running 54 minutes or longer. If you specify percentage for NO_PREEMPT_RUN_TIME, requires a run time (bsub - We or RUNTIME in l sb. appl i cat i ons), or run limit to be specified for the job (bsub - W, or RUNLIMIT in l sb. queues, or RUNLIMIT in l sb. appl i cat i ons)

Files	Parameter	Syntax and description
lsb.params lsb.queues lsb.applications	MAX_TOTAL_TIME_PREEMPT	<p>MAX_TOTAL_TIME_PREEMPT=<i>minutes</i></p> <ul style="list-style-type: none"> Prevents preemption of jobs that already have an accumulated preemption time of <i>minutes</i> or greater. The accumulated preemption time is reset in the following cases: <ul style="list-style-type: none"> Job status becomes EXIT or DONE Job is re-queued Job is re-run Job is migrated and restarted MAX_TOTAL_TIME_PREEMPT does not affect preemption triggered by advance reservation or Platform License Scheduler. Accumulated preemption time does not include preemption by advance reservation or Platform License Scheduler.
	NO_PREEMPT_INTERVAL	<p>NO_PREEMPT_INTERVAL=<i>minutes</i></p> <ul style="list-style-type: none"> Prevents preemption of jobs until after an uninterrupted run time interval of <i>minutes</i> since the job was dispatched or last resumed. NO_PREEMPT_INTERVAL does not affect preemption triggered by advance reservation or Platform License Scheduler.

Configuration to modify preemption of backfill and exclusive jobs

File	Parameter	Syntax and description
lsb.params	PREEMPT_JOBTYPE	<p>PREEMPT_JOBTYPE=BACKFILL</p> <ul style="list-style-type: none"> Enables preemption of backfill jobs. Requires the line PREEMPTION=PREEMPTABLE in the queue definition. Only jobs from queues with a higher priority than queues that define resource or slot reservations can preempt jobs from backfill queues. <p>PREEMPT_JOBTYPE=EXCLUSIVE</p> <ul style="list-style-type: none"> Enables preemption of and preemption by exclusive jobs. Requires the line PREEMPTION=PREEMPTABLE or PREEMPTION=PREEMPTIVE in the queue definition. Requires the definition of LSB_DISABLE_LIMLOCK_EXCL in lsf.conf. <p>PREEMPT_JOBTYPE=EXCLUSIVE BACKFILL</p> <ul style="list-style-type: none"> Enables preemption of exclusive jobs, backfill jobs, or both.

File	Parameter	Syntax and description
lsf.conf	LSB_DISABLE_LIMLOCK_EXCL	<p>LSB_DISABLE_LIMLOCK_EXCL=y</p> <ul style="list-style-type: none"> Enables preemption of exclusive jobs. For a host running an exclusive job: <ul style="list-style-type: none"> lsload displays the host status ok. bhosts displays the host status closed. Users can run tasks on the host using lsrun or lsgun. To prevent users from running tasks during execution of an exclusive job, the parameter LSF_DISABLE_LSRUN=y must be defined in lsf.conf. Changing this parameter requires a restart of all sbatchds in the cluster (badmi n hrestart). Do not change this parameter while exclusive jobs are running.

Configuration to modify how job slot usage is calculated

File	Parameter	Syntax and description
lsb.params	PREEMPT_FOR	<p>PREEMPT_FOR=GROUP_JLP</p> <ul style="list-style-type: none"> Counts only running jobs when evaluating if a user group is approaching its per-processor job slot limit (SLOTS_PER_PROCESSOR, USERS, and PER_HOST=all in the lsb.resources file), ignoring suspended jobs <hr/> <p>PREEMPT_FOR=GROUP_MAX</p> <ul style="list-style-type: none"> Counts only running jobs when evaluating if a user group is approaching its total job slot limit (SLOTS, PER_USER=all, and HOSTS in the lsb.resources file), ignoring suspended jobs <hr/> <p>PREEMPT_FOR=HOST_JLU</p> <ul style="list-style-type: none"> Counts only running jobs when evaluating if a user or user group is approaching its per-host job slot limit (SLOTS, PER_USER=all, and HOSTS in the lsb.resources file), ignoring suspended jobs <hr/> <p>PREEMPT_FOR=USER_JLP</p> <ul style="list-style-type: none"> Counts only running jobs when evaluating if a user is approaching their per-processor job slot limit (SLOTS_PER_PROCESSOR, USERS, and PER_HOST=all in the lsb.resources file) Ignores suspended jobs when calculating the per-processor job slot limit for individual users

Configuration to modify preemption of parallel jobs

File	Parameter	Syntax and description
l sb. params	PREEMPT_FOR	PREEMPT_FOR=MINI_JOB <ul style="list-style-type: none"> Optimizes preemption of parallel jobs by preempting only enough low-priority parallel jobs to start the high-priority parallel job
		PREEMPT_FOR=OPTIMAL_MINI_JOB <ul style="list-style-type: none"> Optimizes preemption of parallel jobs by preempting only low-priority parallel jobs based on the least number of jobs that will be suspended to allow the high-priority parallel job to start

Configuration to modify the control action applied to preempted jobs

File	Parameter	Syntax and description
l sb. queues	TERMI NATE_WHEN	TERMINATE_WHEN=PREEMPT <ul style="list-style-type: none"> Changes the default control action of SUSPEND to TERMINATE so that LSF terminates preempted jobs

Configuration to control how many times a job can be preempted

By default, if preemption is enabled, there is actually no guarantee that a job will ever actually complete. A lower priority job could be preempted again and again, and ultimately end up being killed due to a run limit.

Limiting the number of times a job can be preempted is configured cluster-wide (l sb. params), at the queue level (l sb. queues), and at the application level (l sb. appl i cat i ons). MAX_JOB_PREEMPT in l sb. appl i cat i ons overrides l sb. queues, and l sb. queues overrides l sb. params configuration.

Files	Parameter	Syntax and description
l sb. params	MAX_JOB_PREEMPT	MAX_JOB_PREEMPT= <i>integer</i>
l sb. queues		<ul style="list-style-type: none"> Specifies the maximum number of times a job can be preempted. Specify a value within the following ranges: $0 < \text{MAX_JOB_PREEMPT} < \text{INFINIT_INT}$ INFINIT_INT is defined in l sf . h By default, the number of preemption times is unlimited.
l sb. appl i cat i ons		

When MAX_JOB_ PREEMPT is set, and a job is preempted by higher priority job, the number of job preemption times is set to 1. When the number of preemption times exceeds MAX_JOB_ PREEMPT, the job will run to completion and cannot be preempted again.

The job preemption limit times is recovered when LSF is restarted or reconfigured.

Preemptive scheduling commands

Commands for submission

Command	Description
<code>bsub -q <i>queue_name</i></code>	<ul style="list-style-type: none"> Submits the job to the specified queue, which may have a run limit associated with it
<code>bsub -W <i>minutes</i></code>	<ul style="list-style-type: none"> Submits the job with the specified run limit, in minutes
<code>bsub -app <i>application_profile_name</i></code>	<ul style="list-style-type: none"> Submits the job to the specified application profile, which may have a run limit associated with it

Commands to monitor

Command	Description
<code>bjobs -s</code>	<ul style="list-style-type: none"> Displays suspended jobs, together with the reason the job was suspended

Commands to control

Command	Description
<code>brun</code>	<ul style="list-style-type: none"> Forces a pending job to run immediately on specified hosts. For an exclusive job, when <code>LSB_DISABLE_LIMLOCK_EXCL=y</code>, LSF allows other jobs already running on the host to finish but does not dispatch any additional jobs to that host until the exclusive job finishes.

Commands to display configuration

Command	Description
<code>bqueues</code>	<ul style="list-style-type: none"> Displays the priority (PRI O) and run limit (RUNLI MI T) for the queue, and whether the queue is configured to be preemptive, preemptable, or both
<code>bhosts</code>	<ul style="list-style-type: none"> Displays the number of job slots per user for a host Displays the number of job slots available
<code>bparams</code>	<ul style="list-style-type: none"> Displays the value of parameters defined in <code>lsb.params</code>.
<code>badmin showconf</code>	<ul style="list-style-type: none"> Displays all configured parameters and their values set in <code>lsf.conf</code> or <code>ego.conf</code> that affect <code>mbatchd</code> and <code>sbatchd</code>. Use a text editor to view other parameters in the <code>lsf.conf</code> or <code>ego.conf</code> configuration files. In a MultiCluster environment, displays the parameters of daemons on the local cluster.

Specifying Resource Requirements

About resource requirements

Resource requirements define which hosts a job can run on. Each job has its resource requirements and hosts that match the resource requirements are the candidate hosts. When LSF schedules a job, it uses the load index values of all the candidate hosts. The load values for each host are compared to the scheduling conditions. Jobs are only dispatched to a host if all load values are within the scheduling thresholds.

By default, if a job has no resource requirements, LSF places it on a host of the same type as the submission host (i.e., `type==local`). However, if a job has string or Boolean resource requirements specified and the host type has not been specified, LSF places the job on any host (i.e., `type==any`) that satisfies the resource requirements.

To override the LSF defaults, specify resource requirements explicitly. Resource requirements can be set for queues, for application profiles, or for individual jobs.

To best place a job with optimized performance, resource requirements can be specified for each application. This way, you do not have to specify resource requirements every time you submit a job. The LSF administrator may have already configured the resource requirements for your jobs, or you can put your executable name together with its resource requirements into your personal remote task list.

The `bsub` command automatically uses the resource requirements of the job from the remote task lists.

A resource requirement is an expression that contains resource names and operators.

Compound resource requirements

In some cases different resource requirements may apply to different parts of a parallel job. The first execution host, for example, may require more memory or a faster processor for optimal job scheduling. Compound resource requirements allow you to specify different requirements for some slots within a job in the queue-level, application-level, or job-level resource requirement string.

Compound resource requirement strings can be set by the application-level or queue-level `RES_REQ` parameter, or used with `bsub -R` when a job is submitted. `bmod -R` also accepts compound resource requirement strings for both pending and running jobs.

Special rules take effect when compound resource requirements are merged with resource requirements defined at more than one level. If a compound resource requirement is used at any level (job, application, or queue) the compound multi-level resource requirement combinations described later in this chapter apply.

Restriction:

Compound resource requirements cannot contain `cu` sections, multiple `-R` options, or the `||` operator.

Resizable jobs cannot have compound resource requirements.

Compound resource requirements cannot be specified in the definition of a guaranteed resource pool.

Resource allocation for parallel jobs using compound resources is done for each compound resource term in the order listed instead of considering all possible combinations. A host rejected for not satisfying one resource requirement term will not be reconsidered for subsequent resource requirement terms.

Compound resource requirements were introduced in LSF Version 7 Update 5, and are not compatible with earlier versions of LSF.

Resource requirements in application profiles

See [Resource requirements](#) on page 550 for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

Resizable jobs and resource requirements

In general, resize allocation requests for resizable jobs use the resource requirements of the running job. When the resource requirement string for a job is modified with `bmod -R`, the new string takes effects for a job resize request. The resource requirement of the allocation request is merged from resource requirements specified at the queue, job, and application levels.

Restriction:

Autoresizable jobs cannot have compute unit resource requirements. Any autoresizable jobs switched to queues with compute unit resource requirements will no longer be autoresizable.

Resizable jobs cannot have compound resource requirements.

Queue-level resource requirements

Each queue can define resource requirements that apply to all the jobs in the queue.

When resource requirements are specified for a queue, and no job-level or application profile resource requirement is specified, the queue-level resource requirements become the default resource requirements for the job.

Resource requirements determined by the queue no longer apply to a running job after running `badmi n reconfi g`. For example, if you change the `RES_REQ` parameter in a queue and reconfigure the cluster, the previous queue-level resource requirements for running jobs are lost.

Syntax

The condition for dispatching a job to a host can be specified through the queue-level `RES_REQ` parameter in the queue definition in `lsb. queues`. Queue-level `RES_REQ` usage values must be in the range set by `RESRSV_LIMIT` (set in `lsb. queues`), or the queue-level `RES_REQ` is ignored.

Examples

```
RES_REQ=select[ ((type==LINUX2.4 && rlm < 2.0) || (type==AIX && rlm < 1.0)) ]
```

This allows a queue, which contains LINUX2.4 and AIX hosts, to have different thresholds for different types of hosts.

```
RES_REQ=select[ ((hname==hostA && mem > 50) || (hname==hostB && mem > 100)) ]
```

Using the `hname` resource in the resource requirement string allows you to set up different conditions for different hosts in the same queue.

Load thresholds

Load thresholds can be configured by your LSF administrator to schedule jobs in queues. Load thresholds specify a load index value.

loadSched

The scheduling threshold that determines the load condition for dispatching pending jobs. If a host's load is beyond any defined `loadSched`, a job is not started on the host. This threshold is also used as the condition for resuming suspended jobs.

loadStop

The suspending condition that determines when running jobs should be suspended.

Thresholds can be configured for each queue, for each host, or a combination of both. To schedule a job on a host, the load levels on that host must satisfy both the thresholds configured for that host and the thresholds for the queue from which the job is being dispatched.

The value of a load index may either increase or decrease with load, depending on the meaning of the specific load index. Therefore, when comparing the host load conditions with the threshold values, you need to use either greater than (`>`) or less than (`<`), depending on the load index.

View queue-level resource requirements

1. Use `bqueues -l` to view resource requirements (`RES_REQ`) defined for the queue:

```
bqueues -l normal
QUEUE: normal
-- No description provided. This is the default queue.
```

```
...  
RES_REQ: select [ type==any]  
rusage[mem=10, dynamic_src=10: duration=2: decay=1]  
...
```

Job-level resource requirements

Each job can specify resource requirements. Job-level resource requirements override any resource requirements specified in the remote task list.

In some cases, the queue specification sets an upper or lower bound on a resource. If you attempt to exceed that bound, your job will be rejected.

Syntax

To specify resource requirements for your job, use `bsub -R` and specify the resource requirement string as usual. You can specify multiple `-R` order, same, rusage, and select sections.

Note:

Within `esub`, you can get resource requirements using the `LSB_SUB_RES_REQ` variable, which merges multiple `-R` from the `bsub` command. If you want to modify the `LSB_SUB_RES_REQ` variable, you cannot use multiple `-R` format. Instead, use the `&&` operator to merge them manually.

Merged `RES_REQ` rusage values from the job and application levels must be in the range of `RESRSV_LIMIT` (set in `lsb.queues`), or the job is rejected.

Examples

```
bsub -R "swp > 15 && hpx order[ut]" myjob
```

or

```
bsub -R "select[swp > 15]" -R "select[hpx] order[ut]" myjob
```

This runs `myjob` on an HP-UX host that is lightly loaded (CPU utilization) and has at least 15 MB of swap memory available.

```
bsub -R "select[swp > 15]" -R "select[hpx] order[r15m]" -R "order[r15m]" -R rusage[mem=100]" -R "order[ut]" -R "same[type]" -R "rusage[tmp=50:duration=60]" -R "same[model]" myjob
```

LSF merges the multiple `-R` options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

View job-level resource requirements

1. Use `bjobs -l` to view resource requirements defined for the job:

```
bsub -R "type==any" -q normal myjob
Job <2533> is submitted to queue <normal>.
bjobs -l 2533
Job <2533>, User <user1>, Project <default>, Status <DONE>, Queue <normal>,
  Command <myjob>
Fri May 10 17:21:26 2009: Submitted from host <hostA>, CWD <$HOME>, Requested
Resources <type==any>;
Fri May 10 17:21:31 2009: Started on <hostB>, Execution Home </home/user1>, Execution
CWD </home/user1>;
Fri May 10 17:21:47 2009: Done successfully. The CPU time used is 0.3 seconds.
...
```

2. After a job is finished, use `bhist -l` to view resource requirements defined for the job:

```
bhist -l 2533
Job <2533>, User <user1>, Project <default>, Command <myjob>
```



```
Fri May 10 17:21:26 2009: Submitted from host <hostA>, to Queue <normal>, CWD
<SHOME>, Requested Resources <type==any>;
Fri May 10 17:21:31 2009: Dispatched to <hostB>;
Fri May 10 17:21:32 2009: Starting (Pid 1850232);
Fri May 10 17:21:33 2009: Running with execution home </home/user1>, Execution
CWD </home/user1>, Execution Pid <1850232>;
Fri May 10 17:21:45 2009: Done successfully. The CPU time used is 0.3 seconds;
...
```

Note:

If you submitted a job with multiple select strings using the `bsub -R` option, `bj obs -l` and `bhi st -l` display a single, merged select string.

About resource requirement strings

Most LSF commands accept a `-R res_req` argument to specify resource requirements. The exact behavior depends on the command. For example, specifying a resource requirement for the `lsl load` command displays the load levels for all hosts that have the requested resources.

Specifying resource requirements for the `lssrun` command causes LSF to select the best host out of the set of hosts that have the requested resources.

A resource requirement string describes the resources a job needs. LSF uses resource requirements to select hosts for remote execution and job execution.

Resource requirement strings can be simple (applying to the entire job) or compound (applying to the specified number of slots).

Resource requirement string sections

- A selection section (`select`). The selection section specifies the criteria for selecting hosts from the system.
- An ordering section (`order`). The ordering section indicates how the hosts that meet the selection criteria should be sorted.
- A resource usage section (`rusage`). The resource usage section specifies the expected resource consumption of the task.
- A job spanning section (`span`). The job spanning section indicates if a parallel batch job should span across multiple hosts.
- A same resource section (`same`). The same section indicates that all processes of a parallel job must run on the same type of host.
- A compute unit resource section (`cu`). The `cu` section specifies how a job should be placed with respect to the underlying network architecture.

Which sections apply

Depending on the command, one or more of these sections may apply. For example:

- `bsub` uses all sections
- `lshosts` only selects hosts, but does not order them
- `lsl load` selects and orders hosts
- `lsl place` uses the information in `select`, `order`, and `rusage` sections to select an appropriate host for a task
- `lsl oadadj` uses the `rusage` section to determine how the load information should be adjusted on a host

Simple syntax

```
select[selection_string] order[order_string] rusage[usage_string [, usage_string]  
[| usage_string] ... ] span[span_string] same[same_string] cu[cu_string]
```

With the `bsub` and `bmod` commands, and only with these commands, you can specify multiple `-R order`, `same`, `rusage`, and `select` sections. The `bmod` command does not support the use of the `||` operator.

The section names are `select`, `order`, `rusage`, `span`, `same`, and `cu`. Sections that do not apply for a command are ignored. Each section has a different syntax.

The square brackets must be typed as shown for each section. A blank space must separate each resource requirement section.

You can omit the `select` keyword and the square brackets, but the selection string must be the *first* string in the resource requirement string. If you do not give a section name, the first resource requirement string is treated as a selection string (`select [selection_string]`).

Each section has a different syntax.

By default, memory (`mem`) and swap (`swp`) limits in `select []` and `rusage []` sections are specified in MB. Use `LSF_UNIT_FOR_LIMITS` in `lsf.conf` to specify a larger unit for these limits (MB, GB, TB, PB, or EB).

Compound syntax

```
num1*{simple_string1} + num2*{simple_string2} + ...
```

where *numx* is the number of slots affected and *simple_stringx* is a simple resource requirement string with the syntax:

```
select[selection_string] order[order_string] rusage[usage_string [, usage_string]... ] span  
[span_string]
```

Resource requirements applying to the first execution host (if used) should appear in the first compound term *num1*{simple_string1}*.

Place specific (harder to fill) requirements before general (easier to fill) requirements since compound resource requirement terms are considered in the order they appear. Resource allocation for parallel jobs using compound resources is done for each compound resource term independently instead of considering all possible combinations.

Note:

A host rejected for not satisfying one resource requirement term will not be reconsidered for subsequent resource requirement terms.

For jobs without the number of total slots specified using `bsub -n`, the final *numx* can be omitted. The final resource requirement is then applied to the zero or more slots not yet accounted for using the default slot setting of the parameter `PROCLIMIT` as follows:

- (final `res_req` number of slots) = $\text{MAX}(0, (\text{default number of job slots from PROCLIMIT}) - (\text{num1} + \text{num2} + \dots))$

For jobs with the total number of slots specified using `bsub -n num_slots`, the total number of slots must match the number of slots in the resource requirement as follows, and the final *numx* can be omitted:

- $\text{num_slots} = (\text{num1} + \text{num2} + \text{num3} + \dots)$

For jobs with compound resource requirements and first execution host candidates specified using `bsub -m`, the first allocated host must satisfy the simple resource requirement string appearing first in the compound resource requirement. Thus the first execution host must satisfy the requirements in *simple_string1* for the following compound resource requirement:

- $\text{num1}\{ \text{simple_string1} \} + \text{num2}\{ \text{simple_string2} \} + \text{num3}\{ \text{simple_string3} \}$

Compound resource requirements do not support use of the `||` operator within the component `rusage` simple resource requirements, or use of the `cu` section.

How simple multi-level resource requirements are resolved

Simple resource requirements can be specified at the job, application, and queue levels. When none of the resource requirements are compound, requirements defined at different levels are resolved in the following ways:

- In a select string, a host must satisfy *all* queue-level, application-level, and job-level requirements for the job to be dispatched.
- In a same string, all queue-level, application-level, and job-level requirements are combined before the job is dispatched.
- order, span, and cu sections defined at the job level overwrite those defined at the application level or queue level. order, span, and cu sections defined at the application level overwrite those defined at the queue level. The default order string is r15s: pg.
- For usage strings, the rusage section defined for the job overrides the rusage section defined in the application. The two rusage definitions are merged, with the job-level rusage taking precedence. Similarly, rusage strings defined for the job or application are merged with queue-level strings, with the job and then application definitions taking precedence over the queue if there is any overlap.

section	simple resource requirement multi-level behavior
select	all levels satisfied
same	all levels combined
order span cu	job-level section overwrites application-level section, which overwrites queue-level section (if a given level is present)
rusage	all levels merge if conflicts occur the job-level section overwrites the application-level section, which overwrites the queue-level section.

For internal load indices and duration, jobs are rejected if the merged job-level and application-level resource reservation requirements exceed the requirements specified at the queue level.

Note:

If a compound resource requirement is used at one or more levels, (job, application, or queue) the compound rules apply.

How compound multi-level resource requirements are resolved

Compound resource requirements can be specified at the job, application, and queue levels. When one or more of the resource requirements is compound, requirements at different levels are resolved depending on where the compound resource requirement appears.

For internal load indices and duration, jobs are rejected if they specify resource reservation requirements that exceed the requirements specified at the application level or queue level.

Note:

If a compound resource requirement is used at one or more levels, (job, application, or queue) the compound rules apply.

Compound queue level

When a compound resource requirement is set for a queue it will be ignored unless it is the only resource requirement specified (no resource requirements are set at the job level or application level).

Compound application level

When a compound resource requirement is set at the application level, it will be ignored if any job-level resource requirements (simple or compound) are defined.

In the event no job-level resource requirements are set, the compound application-level requirements interact with queue-level resource requirement strings in the following ways:

- If no queue-level resource requirement is defined or a compound queue-level resource requirement is defined, the compound application-level requirement is used.
- If a simple queue-level requirement is defined, the application-level and queue-level requirements combine as follows:

section	compound application and simple queue behavior
select	both levels satisfied; queue requirement applies to all compound terms
same	queue level ignored
order span	application-level section overwrites queue-level section (if a given level is present); queue requirement (if used) applies to all compound terms
rusage	<ul style="list-style-type: none"> • both levels merge • queue requirement if a job-based resource is applied to the first compound term, otherwise applies to all compound terms • if conflicts occur the application-level section overwrites the queue-level section. <p>For example: if the application-level requirement is $\text{num1} * \{\text{rusage}[\text{R1}]\} + \text{num2} * \{\text{rusage}[\text{R2}]\}$ and the queue-level requirement is $\text{rusage}[\text{RQ}]$ where RQ is a job-based resource, the merged requirement is $\text{num1} * \{\text{rusage}[\text{merge}(\text{R1}, \text{RQ})]\} + \text{num2} * \{\text{rusage}[\text{R2}]\}$</p>

Compound job level

When a compound resource requirement is set at the job level, any simple or compound application-level resource requirements are ignored, and any compound queue-level resource requirements are ignored.

In the event a simple queue-level requirement appears along with a compound job-level requirement, the requirements interact as follows:

section	compound job and simple queue behavior
select	both levels satisfied; queue requirement applies to all compound terms
same	queue level ignored
order span	job-level section overwrites queue-level section (if a given level is present); queue requirement (if used) applies to all compound terms
rusage	<ul style="list-style-type: none"> • both levels merge • queue requirement if a job-based resource is applied to the first compound term, otherwise applies to all compound terms • if conflicts occur the job-level section overwrites the queue-level section. <p>For example: if the job-level requirement is $\text{num1} * \{\text{rusage}[\text{R1}]\} + \text{num2} * \{\text{rusage}[\text{R2}]\}$ and the queue-level requirement is $\text{rusage}[\text{RQ}]$ where RQ is a job resource, the merged requirement is $\text{num1} * \{\text{rusage}[\text{merge}(\text{R1}, \text{RQ})]\} + \text{num2} * \{\text{rusage}[\text{R2}]\}$</p>

Example 1

A compound job requirement and simple queue requirement.

job level: **2*{select[type==X86_64] rusage[licA=1] span[hosts=1]} + 8*{select[type==any]}**

application level: not defined

queue level: **rusage[perslot=1]**

The final job scheduling resource requirement merges the simple queue-level rusage section into each term of the compound job-level requirement, resulting in: **2*{select[type==X86_64] rusage[licA=1:perslot=1] span[hosts=1]} + 8*{select[type==any] rusage[perslot=1]}**

Example 2

A compound job requirement and compound queue requirement.

job level: **2*{select[type==X86_64 && tmp>10000] rusage[mem=1000] span[hosts=1]} + 8*{select[type==X86_64]}**

application level: not defined

queue level: **2*{select[type==X86_64] rusage[mem=1000] span[hosts=1]} + 8*{select[type==X86_64]}**

The final job scheduling resource requirement ignores the compound queue-level requirement, resulting in: **2*{select[type==X86_64 && tmp>10000] rusage[mem=1000] span[hosts=1]} + 8*{select[type==X86_64]}**

Example 3

A compound job requirement and simple queue requirement where the queue requirement is a job-based resource.

job level: **2*{select[type==X86_64]} + 2*{select[mem>1000]}**

application level: not defined

queue level: **rusage[licA=1]** where **licA=1** is job-based.

The queue-level requirement is added to the first term of the compound job-level requirement, resulting in: **2*{select[type==X86_64] rusage[licA=1]} + 2*{select[mem>1000]}**

Example 4

Compound multi-phase job requirements and simple multi-phase queue requirements.

job level: **2*{rusage[mem=(400 350):duration=(10 15):decay=(0 1)]} + 2*{rusage[mem=300:duration=10:decay=1]}**

application level: not defined

queue level: **rusage[mem=(500 300):duration=(20 10):decay=(0 1)]**

The queue-level requirement is overridden by the first term of the compound job-level requirement, resulting in: **2*{rusage[mem=(400 350):duration=(10 15):decay=(0 1)]} + 2*{rusage[mem=300:duration=10:decay=1]}**

Selection string

The selection string specifies the characteristics a host must have to match the resource requirement. It is a logical expression built from a set of resource names. The selection string is evaluated for each host; if the result is non-zero, then that host is selected. When used in conjunction with a `cu` string, hosts not belonging to compute unit are not considered.

Syntax

The selection string can combine resource names with logical and arithmetic operators. Non-zero arithmetic values are treated as logical TRUE, and zero (0) as logical FALSE. Boolean resources (for example, `server` to denote LSF server hosts) have a value of one (1) if they are defined for a host, and zero (0) if they are not defined for the host.

The resource names `swap`, `idle`, `login`, and `cpu` are accepted as aliases for `swp`, `it`, `ls`, and `rlm` respectively.

The `ut` index measures CPU utilization, which is the percentage of time spent running system and user code. A host with no processes running has a `ut` value of 0 percent; a host on which the CPU is completely loaded has a `ut` of 100 percent. You must specify `ut` as a floating-point number between 0.0 and 1.0.

For the string resources `type` and `model`, the special value `any` selects any value and `local` selects the same value as that of the local host. For example, `type==local` selects hosts of the same type as the host submitting the job. If a job can run on any type of host, include `type==any` in the resource requirements.

If no `type` is specified, the default depends on the command. For `bsub`, `lsplace`, `lsrun`, and `lsgrun` the default is `type==local` unless a string or Boolean resource is specified, in which case it is `type==any`. For `lshosts`, `lslload`, `lsmom` and `lsllogin` the default is `type==any`.

Tip:

When `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the resource requirement string keyword `ncpus` refers to the number of slots instead of the number of CPUs, however `lshosts` output will continue to show `ncpus` as defined by `EGO_DEFINE_NCPUS` in `lsf.conf`.

Specify multiple -R options

`bsub` accepts multiple `-R` options for the select section in simple resource requirements.

Restriction:

Compound resource requirements do not support multiple `-R` options.

You can specify multiple resource requirement strings instead of using the `&&` operator. For example:

```
bsub -R "select[swp > 15]" -R "select[hpx]"
```

LSF merges the multiple `-R` options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

When `LSF_STRICT_RESREQ=Y` is configured in `lsf.conf`, you cannot specify more than one select section in the same `-R` option. Use the logical and (`&&`) operator to specify multiple selection strings in the same select section. For example, the following command submits a job called `myjob` to run on a host

that has more than 15 MB of swap space available, and maximum RAM larger than 100MB. The job is expected to reserve 100MB memory on the host:

```
% bsub -R "select [swp > 15 && maxmem > 100] rusage[mem = 100] " myjob
```

The number of -R option sections is unlimited.

Select shared string resources

You must use single quote characters (') around string-type shared resources. For example, use `lsl load -s` to see the shared resources defined for the cluster:

RESOURCE	VALUE	LOCATION
os_version	4.2	pc36
os_version	4.0	pc34
os_version	4.1	devlinux4
cpu_type	ia	pc36
cpu_type	ia	pc34
cpu_type	unknown	devlinux4

Use a select string in `lsl load -R` to specify the shared resources you want to view, enclosing the shared resource values in single quotes. For example:

```
lsl load -R "select[os_version=='4.2' || cpu_type=='unknown']"
HOST_NAME      status  r15s  r1m  r15m  ut    pg  ls    it    tmp    swp    mem
pc36            ok      0.0  0.2  0.1   1%   3.4  3     0    895M  517M  123M
devlinux4      ok      0.0  0.1  0.0   0%   2.8  4     0   6348M  504M  205M
```

Note:

When reserving resources based on host status (`bsub -R "status==ok"`), the host status must be the one displayed by running `bhosts not lsl load`.

Operators

These operators can be used in selection strings. The operators are listed in order of decreasing precedence.

Syntax	Meaning
(a)	When LSF_STRICT_RESREQ=Y is configured in lsf.conf, an expression between parentheses has higher priority than other operators.
-a	Negative of a
!a	Logical not: 1 if a==0, 0 otherwise
a * b	Multiply a and b
a / b	Divide a by b
a + b	Add a and b
a - b	Subtract b from a
a > b	1 if a is greater than b, 0 otherwise
a < b	1 if a is less than b, 0 otherwise
a >= b	1 if a is greater than or equal to b, 0 otherwise
a <= b	1 if a is less than or equal to b, 0 otherwise

Syntax	Meaning
<code>a == b</code>	1 if a is equal to b, 0 otherwise
<code>a != b</code>	1 if a is not equal to b, 0 otherwise
<code>a && b</code>	Logical AND: 1 if both a and b are non-zero, 0 otherwise
<code>a b</code>	Logical OR: 1 if either a or b is non-zero, 0 otherwise

Examples

```
select[(swp > 50 && type == MIPS) || (swp > 35 && type == ALPHA)]
select[((2*r15s + 3*r1m + r15m) / 6 < 1.0) && !fs && (cpuf > 4.0)]
```

Specify shared resources with the keyword “defined”

A shared resource may be used in the resource requirement string of any LSF command. For example, when submitting an LSF job that requires a certain amount of shared scratch space, you might submit the job as follows:

```
bsub -R "avail_scratch > 200 && swap > 50" myjob
```

The above assumes that all hosts in the cluster have access to the shared scratch space. The job is only scheduled if the value of the "avail_scratch" resource is more than 200 MB and goes to a host with at least 50 MB of available swap space.

It is possible for a system to be configured so that only some hosts within the LSF cluster have access to the scratch space. To exclude hosts that cannot access a shared resource, the `defined(resource_name)` function must be specified in the resource requirement string.

For example:

```
bsub -R "defined(avail_scratch) && avail_scratch > 100 && swap > 100" myjob
```

would exclude any hosts that cannot access the scratch resource. The LSF administrator configures which hosts do and do not have access to a particular shared resource.

Supported resource names in the defined function

Only the following resource names are accepted as the argument in the `defined(resource_name)` function:

- The following built-in resource names:

```
LSF_Base lsf_base LSF_Manager lsf_manager LSF_JobScheduler
lsf_js LSF_Make LSF_parallel LSF_Analyzer lsf_analyzer
```

- Resource names configured in `lsf.shared`, *except* dynamic NUMERIC resource names with INTERVAL fields defined.

The following resource names are *not* accepted in the `defined(resource_name)` function:

- The following built-in resource names:

```
r15s r1m r15m ut pg io ls it tmp swp mem ncpus ndisks maxmem
maxswp maxtmp cpuf type model status rexpri server and hname
```

- Dynamic NUMERIC resource names configured in `lsf.shared` with INTERVAL fields defined. In the default configuration, these are `mode`, `cntrl`, `it_t`.)
- Other non-built-in resource names not configured in `lsf.shared`.

Specify exclusive resources

An exclusive resource may be used in the resource requirement string of any placement or scheduling command, such as `bsub`, `lplace`, `lrun`, or `lsgun`. An exclusive resource is a special resource that is assignable to a host. This host will not receive a job unless that job explicitly requests the host. For example, use the following command to submit a job requiring the exclusive resource `bigmem`:

```
bsub -R "bigmem" myjob
```

Jobs will not be submitted to the host with the `bigmem` resource unless the command uses the `-R` option to explicitly specify `"bigmem"`.

To configure an exclusive resource, first define a static Boolean resource in `lsf.conf`. For example:

```
Begin Resource
...
bigmem Boolean () ()
End Resource
```

Assign the resource to a host in the Host section of `lsf.cluster.cluster_name`. Prefix the resource name with an exclamation mark (!) to indicate that the resource is exclusive to the host. For example:

```
Begin Host
HOSTNAME  model      type      server  rlm  pg  tmp  RESOURCES          RUNWINDOW
...
hostE     !          !          1       3.5  ()  ()  (linux !bigmem)    ()
...
End Host
```

Strict syntax for resource requirement selection strings

When `LSF_STRICT_RESREQ=Y` is configured in `lsf.conf`, resource requirement strings in select sections must conform to a more strict syntax. The strict resource requirement syntax only applies to the `select` section. It does not apply to the other resource requirement sections (`order`, `usage`, `same`, `span`, or `cu`). When `LSF_STRICT_RESREQ=Y` in `lsf.conf`, LSF rejects resource requirement strings where an `usage` section contains a non-consumable resource.

Strict syntax in EBNF form:

```
<expression> ::= <relation1> { <logical or> <relation1> }
<relation1> ::= <relation2> { <logical and> <relation2> }
<relation2> ::= <simple expression> [ <relation op> <simple expression> ]
<simple expression> ::= <term> { <adding op> <term> }
<term> ::= <factor> { <multiple op> <factor> }
<factor> ::= [ <unary op> ] <primary>
<primary> ::= <numeric> | <string> | ( <expression> ) | <name or function call>
<logical or> ::= |
<logical and> ::= &&
<relation op> ::= <= | >= | == | != | < | > | =
<adding op> ::= + | -
<unary op> ::= - | !
<multiple op> ::= * | /
<name or function call> ::= <name> [ ( <argument list> ) ]
<argument list> ::= <empty> | <argument> { , <argument> }
<argument> ::= <expression>
<name> ::= [ a-zA-Z_ ] [ a-zA-Z_0-9 ] *
<numeric> ::= <int> [ . [ 0-9 ] * ]
<int> ::= [ 1-9 ] [ 0-9 ] * | 0
<string> ::= <single quote> { <string chars> } <single quote> | <double quote> { <string chars> } <double quote>
<string chars> ::= <printable ascii characters except single/double quote>
<single quote> ::= '
<double quote> ::= "
<empty> ::=
```

Strict select string syntax usage notes

The strict syntax is case sensitive.

Operators '=' and '==' are equivalent.

Boolean variables, such as `fs`, `hpx`, `cs`, can only be computed with the following operators:

```
&& || !
```

String variables, such as `type`, can only be computed with the following operators:

```
= == != < > <= >=
```

For function calls, blanks between the parentheses "()" and the resource name are not valid. For example, the following is not correct:

```
defi ned(   mg   )
```

Multiple logical NOT operators (!) are not valid. For example, the following is not correct:

```
!!mg
```

The following resource requirement is valid:

```
!( !mg)
```

At least one blank space must separate each section. For example, the following are correct:

```
type==any rusage[mem=1024]
sel ect [type==any] rusage[mem=1024]
sel ect [type==any] rusage[mem=1024]
```

but the following is not correct:

```
type==anyrusage[mem=1024]
```

Only a single select section is supported by the stricter syntax. The following is not supported in the same resource requirement string:

```
sel ect [mem>0] sel ect [maxmem>0]
```

Escape characters (like '\n') are not supported in string literals.

A colon (:) is not allowed inside the select string. For example, `sel ect [mg: bi gmem]` is not correct.

`inf` and `nan` can be used as resource names or part of a resource name.

Single or double quotes are only supported around the whole resource requirement string, not within the square brackets containing the selection string. For example, in `l sb. queues`, `RES_REQ=' swp>100'` and `RES_REQ=" swp>100"` are correct. Neither `RES_REQ=sel ect [' swp>100']` nor `RES_REQ=sel ect [" swp>100"]` are supported.

The following are correct `bsub` command-level resource requirements:

- `bsub -R "' swp>100' "`
- `bsub -R "' " swp>100"'`

The following are not correct:

- `bsub -R "sel ect [' swp>100'] "`
- `bsub -R ' sel ect [" swp>100"] '`

Some incorrect resource requirements are no longer silently ignored. For example, when `LSF_STRICT_RESREQ=Y` is configured in `l sf. conf`, the following are rejected by the resource requirement parser:

- `mi crocs73` is rejected:

```
l i nux rusage[mem=16000] mi crocs73
```

- `select [AMD64]` is rejected:

```
mem < 16384 && select [AMD64]
```
- `linux` is rejected:

```
rusage[mem=2000] linux
```
- Using a colon (:) to separate select conditions, such as `linux: qscw`.
- The restricted syntax of resource requirement select strings described in the `lsfintro(1)` man page is not supported.

Explicit and implicit select sections

An explicit select section starts from the section keyword and ends at the begin of next section, for example: the select section is `select [selectio_n_string]`. An implicit select section starts from the first letter of the resource requirement string and ends at the end of the string if there are no other resource requirement sections. If the resource requirement has other sections, the implicit select section ends before the first letter of the first section following the selection string.

All explicit sections must begin with a section keywords (`select`, `order`, `span rusage`, or `same`). The resource requirement content is contained by square brackets (`[]` and `()`).

An implicit select section must be the first resource requirement string in the whole resource requirement specification. Explicit select sections can appear after other sections. A resource requirement string can have only one select section (either an explicit select section or an implicit select section). A section with an incorrect keyword name is not a valid section.

An implicit select section must have the same format as the content of an explicit select section. For example, the following commands are correct:

- `bsub -R "select [swp>15] rusage[mem=100]" myjob`
- `bsub -R "swp > 15 rusage[mem=100]" myjob`
- `bsub -R "rusage[mem=100] select [swp >15]" myjob`

Examples

The following examples illustrate some correct resource requirement select string syntax.

- `bsub -R "(r15s * 2 + r15m) < 3.0 && !(type == IBMAIX4) || fs" myjob`
- If swap space is equal to 0, the following means TRUE; if swap space is not equal to 0, it means FALSE:

```
bsub -R "!swp" myjob
```
- Select hosts of the same type as the host submitting the job:

```
bsub -R "type == local" myjob
```
- Select hosts that are not the same type as the host submitting the job:

```
bsub -R "type != local" myjob
```
- `bsub -R "r15s < 1.0 || model ==local && swp <= 10" myjob`

Since `&&` has a higher priority than `||`, this example means:

```
r15s < 1.0 || (model == local && swp <=10)
```

- This example has different meaning from the previous example:

```
bsub -R "(r15s < 1.0 || model == local) && swp <= 10" myjob
```

This example means:

```
(r15s < 1.0 || model == local) && swp <= 10
```

Check resource requirement syntax

Use the `BSUB_CHK_RESREQ` environment variable to check the compatibility of your existing resource requirement select strings against the stricter syntax enabled by `LSF_STRICT_RESREQ=Y` in `lsf.conf`.

Set the `BSUB_CHK_RESREQ` environment variable to any value enable `bsub` to check the syntax of the resource requirement selection string without actually submitting the job for scheduling and dispatch. `LSF_STRICT_RESREQ` does not need to be set to check the resource requirement selection string syntax.

`bsub` only checks the select section of the resource requirement. Other sections in the resource requirement string are not checked.

If resource requirement checking detects syntax errors in the selection string, `bsub` returns an error message. For example:

```
bsub -R "select[type==local] select[hname=abc]" sleep 10
Error near "select": duplicate section. Job not submitted.
echo $?
255
```

If no errors are found, `bsub` returns a successful message and exit code zero (0). For example:

```
env | grep BSUB_CHK_RESREQ
BSUB_CHK_RESREQ=1
bsub -R "select[type==local]" sleep 10
Resource requirement string is valid.
echo $?
0
```

If `BSUB_CHK_RESREQ` is set, but you do not specify `-R`, LSF treats it as empty resource requirement. For example:

```
bsub sleep 120
Resource requirement string is valid.
echo $?
0
```

Resizable jobs

Resize allocation requests are scheduled using hosts as determined by the `select` expression of the merged resource requirement. For example, to run an autoresizable job on 1-100 slots, but only on hosts of type `X86_64`, the following job submission specifies this resource request:

```
bsub -ar -n "1, 100" -R "select[type == X86_64]" myjob
```

Every time the job grows in slots, slots are requested on hosts of the specified type.

Note:

Resizable jobs cannot have compound resource requirements.

Order string

The order string allows the selected hosts to be sorted according to the values of resources. The values of `r15s`, `r1m`, and `r15m` used for sorting are the normalized load indices returned by `lsload -N`.

The order string is used for host sorting and selection. The ordering begins with the rightmost index in the order string and proceeds from right to left. The hosts are sorted into order based on each load index, and if more hosts are available than were requested, the LIM drops the least desirable hosts according to that index. The remaining hosts are then sorted by the next index.

After the hosts are sorted by the leftmost index in the order string, the final phase of sorting orders the hosts according to their status, with hosts that are currently not available for load sharing (that is, not in the `ok` state) listed at the end.

Because the hosts are sorted again for each load index, only the host status and the leftmost index in the order string actually affect the order in which hosts are listed. The other indices are only used to drop undesirable hosts from the list.

When sorting is done on each index, the direction in which the hosts are sorted (increasing vs. decreasing values) is determined by the default order returned by `lsinfo` for that index. This direction is chosen such that after sorting, by default, the hosts are ordered from best to worst on that index.

When used with a `cu` string, the preferred compute unit order takes precedence. Within each compute unit hosts are ordered according to the order string requirements.

Syntax

```
[ - ] resource_name [ : [ - ] resource_name ] . . .
```

You can specify any built-in or external load index or static resource.

When an index name is preceded by a minus sign `'-'`, the sorting order is reversed so that hosts are ordered from worst to best on that index.

Specify multiple -R options

`bsub` accepts multiple `-R` options for the order section.

Restriction:

Compound resource requirements do not support multiple `-R` options.

You can specify multiple resource requirement strings instead of using the `&&` operator. For example:

```
bsub -R "order[r15m]" -R "order[ut]"
```

LSF merges the multiple `-R` options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts. The number of `-R` option sections is unlimited.

Default

The default sorting order is `r15s: pg` (except for `lslogln(1): ls: r1m`).

```
swp: r1m: tmp: r15s
```

Resizable jobs

The order in which hosts are considered for resize allocation requests is determined by the order expression of the job. For example, to run an autoresizable job on 1-100 slots, preferring hosts with larger memory, the following job submission specifies this resource request:

```
bsub -ar -n "1, 100" -R "order[mem]" myjob
```

When slots on multiple hosts become available simultaneously, hosts with larger available memory get preference when the job adds slots.

Note:

Resizable jobs cannot have compound resource requirements.

Usage string

This string defines the expected resource usage of the job. It is used to specify resource reservations for jobs, or for mapping jobs on to hosts and adjusting the load when running interactive jobs.

By default, no resources are reserved.

When `LSF_STRICT_RESREQ=Y` in `lsf.conf`, LSF rejects resource requirement strings where an `rusage` section contains a non-consumable resource.

Multi-phase resources

Multiple phases within the `rusage` string allow different time periods to have different memory requirements (load index `mem`). The duration of all except the last phase must be specified, while decay rates are all optional and are assumed to be 0 if omitted. If the optional final duration is left blank, the final resource requirement applies until the job is finished.

Multi-phase resource reservations cannot include increasing resources, but can specify constant or decreasing resource reservations over multiple periods of time.

Resource reservation limits

Resource requirement reservation limits can be set using the parameter `RESRSV_LIMIT` in `lsb.queues`. Queue-level `RES_REQ` `rusage` values (set in `lsb.queues`) must be in the range set by `RESRSV_LIMIT`, or the queue-level `RES_REQ` is ignored. Merged `RES_REQ` `rusage` values from the job and application levels must be in the range of `RESRSV_LIMIT`, or the job is rejected.

When both the `RES_REQ` and `RESRSV_LIMIT` are set in `lsb.queues` for a consumable resource, the queue-level `RES_REQ` no longer acts as a hard limit for the merged `RES_REQ` `rusage` values from the job and application levels. In this case only the limits set by `RESRSV_LIMIT` must be satisfied, and the queue-level `RES_REQ` acts as a default value.

Batch jobs

The resource usage (`rusage`) section can be specified at the job level, with the queue configuration parameter `RES_REQ`, or with the application profile parameter `RES_REQ`.

Basic syntax

```
rusage[ usage_string [, usage_string] [ | | usage_string] ... ]
```

where *usage_string* is:

```
load_index=value [:load_index=value]... [:duration=minutes[m] | :duration=hoursh  
| :duration=secondss [:decay=0 | :decay=1]]
```

Multi-phase memory syntax

```
rusage[ multi_usage_string [, usage_string] ... ]
```

where *multi_usage_string* is:

```
mem=(v1 [v2 ... vn]):[duration=(t1 [t2 ... tm])][:decay=(d1 [d2... dk])]
```

for $m = n | n-1$. For a single phase ($n=1$), duration is not required.

if $k > m$, $dk+1$ to dk will be ignored; if $k < m$, $dk+1 = .. = dm = 0$.

usage_string is the same as the basic syntax, for any *load_index* other than `mem`.

Multi-phase syntax can be used with a single phase memory resource requirement as well as for multiple phases. For multi-phase slot-based resource reservation, use with **RESOURCE_RESERVE_PER_SLOT=Y** in `lsb.params`.

Multi-phase resource reservations cannot increase over time. A job submission with increasing resource reservations from one phase to the next will be rejected. For example:

```
bsub -R"rusage[mem=(200 300):duration=(2 3)]" myjob
```

specifies an increasing memory reservation from 200 MB to 300 MB. This job will be rejected.

Tip:

When a multi-phase `mem` resource requirement is being used, `duration` can be specified separately for single-phase resources.

Load index

Internal and external load indices are considered in the resource usage string. The resource value represents the initial reserved amount of the resource.

Duration

The duration is the time period within which the specified resources should be reserved. Specify a duration equal to or greater than the ELIM updating interval.

- If the value is followed by the letter `s`, `m`, or `h`, the specified time is measured in seconds, minutes, or hours respectively.
- By default, duration is specified in minutes.

For example, the following specify a duration of 1 hour for multi-phase syntax:

- `duration=(60)`
- `duration=(1h)`
- `duration=(3600s)`

For example, the following specify a duration of 1 hour for single-phase syntax:

- `duration=60`
- `duration=1h`
- `duration=3600s`

Tip:

Duration is not supported for static shared resources. If the shared resource is defined in an `lsb.resources Limit` section, then duration is not applied.

Decay

The decay value indicates how the reserved amount should decrease over the duration.

- A value of 1 indicates that system should linearly decrease the amount reserved over the duration.
- A value of 0 causes the total amount to be reserved for the entire duration.

Values other than 0 or 1 are unsupported, and are taken as the default value of 0. If duration is not specified, decay value is ignored.

Tip:

Decay is not supported for static shared resources. If the shared resource is defined in an `lsb.resources` Limit section, then decay is not applied.

Default

If a resource or its value is not specified, the default is not to reserve that resource. If duration is not specified, the default is to reserve the total amount for the lifetime of the job. (The default decay value is 0.)

Example

```
rusage[mem=50:duration=100:decay=1]
```

This example indicates that 50 MB memory should be reserved for the job. As the job runs, the amount reserved will decrease at approximately 0.5 MB per minute until the 100 minutes is up.

How simple queue-level and job-level rusage sections are resolved

Job-level rusage overrides the queue level specification:

- For internal load indices (`r15s`, `r1m`, `r15m`, `ut`, `pg`, `io`, `l s`, `it`, `tmp`, `swp`, and `mem`), the job-level value cannot be larger than the queue-level value (unless the limit parameter `RESRSV_LIMIT` is being used as a maximum instead of the queue-level value).
- For external load indices (e.g., licenses), the job-level rusage can be larger than the queue-level requirements.
- For duration, the job-level value of internal and external load indices cannot be larger than the queue-level value.
- For multi-phase simple rusage sections:
 - For internal load indices (`r15s`, `r1m`, `r15m`, `ut`, `pg`, `io`, `l s`, `it`, `tmp`, `swp`, and `mem`), the first phase of the job-level value cannot be larger than the first phase of the queue-level value (unless the limit parameter `RESRSV_LIMIT` is being used as a maximum instead of the queue-level value).
 - For duration and decay, if either job-level or queue-level is multi-phase, the job-level value will take precedence.

How simple queue-level and job-level rusage sections are merged

When both job-level and queue-level rusage sections are defined, the rusage section defined for the job overrides the rusage section defined in the queue. The two rusage definitions are merged, with the job-level rusage taking precedence. For example:

Example 1

Given a `RES_REQ` definition in a queue:

```
RES_REQ = rusage[mem=200:lic=1] ...
```

and job submission:

```
bsub -R "rusage[mem=100]" ...
```

The resulting requirement for the job is

```
rusage[mem=100:lic=1]
```

where `mem=100` specified by the job overrides `mem=200` specified by the queue. However, `lic=1` from queue is kept, since job does not specify it.

Example 2

For the following queue-level RES_REQ (decay and duration defined):

```
RES_REQ = rusage[mem=200:duration=20:decay=1] ...
```

and job submission (no decay or duration):

```
bsub -R "rusage[mem=100]" ...
```

The resulting requirement for the job is:

```
rusage[mem=100:duration=20:decay=1]
```

Queue-level duration and decay are merged with the job-level specification, and mem=100 for the job overrides mem=200 specified by the queue. However, duration=20 and decay=1 from queue are kept, since job does not specify them.

rusage in application profiles

See [Resource requirements](#) on page 550 for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

How simple queue-level rusage sections are merged with compound rusage sections

When simple queue-level and compound application-level or job-level rusage sections are defined, the two rusage definitions are merged. If a job-level resource requirement (simple or compound) is defined, the application level is ignored and the job-level and queue-level sections merge. If no job-level resource requirement is defined, the application-level and queue-level merge.

When a compound resource requirement merges with a simple resource requirement from the queue-level, the behavior depends on whether the queue-level requirements are job-based or not.

Example 1

Job-based simple queue-level requirements apply to the first term of the merged compound requirements. For example:

Given a RES_REQ definition for a queue which refers to a job-based resource:

```
RES_REQ = rusage[lic=1] ...
```

and job submission resource requirement:

```
bsub -R "2*{rusage[mem=100] ...} + 4*{mem=200:duration=20:decay=1} ..."
```

The resulting requirement for the job is

```
bsub -R "2*{rusage[mem=100:lic=1] ...} + 4*{rusage[mem=200:duration=20:decay=1] ...}"
```

The job-based resource lic=1 from queue is added to the first term only, since it is job-based and wasn't included the job-level requirement.

Example 2

Host-based or slot-based simple queue-level requirements apply to all terms of the merged compound requirements. For example:

For the following queue-level RES_REQ which does not include job-based resources:

```
RES_REQ = rusage[mem=200:duration=20:decay=1] ...
```

and job submission:

```
bsub -R "2*{rusage[mem=100] ...} + 4*{rusage[lic=1] ...}"
```

The resulting requirement for the job is:

```
2*{rusage[mem=100: duration=20: decay=1] ...} + 4*{rusage
[lic=1: mem=200: duration=20: decay=1] ...}
```

Where `duration=20` and `decay=1` from queue are kept, since job does not specify them in any term. In the first term `mem=100` from the job is kept; in the second term `mem=200` from the queue is used since it wasn't specified by the job resource requirement.

Specify multiple -R options

`bsub` accepts multiple -R options for the `rusage` section.

Restriction:

Compound resource requirements do not support multiple -R options.
Multi-phase `rusage` strings do not support multiple -R options.

You can specify multiple resource requirement strings instead of using the `&&` operator. For example:

```
bsub -R "rusage[mem=100]" -R "rusage[tmp=50:duration=60]"
```

LSF merges the multiple -R options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

The number of -R option sections is unlimited.

Comma-separated multiple resource requirements within one `rusage` string is supported. For example:

```
bsub -R "rusage[mem=20, license=1:duration=2]" -R "rusage[app_lic_v201=1||app_lic_v15=1]"
myjob
```

A given load index cannot appear more than once in the resource usage string.

Examples

- The following job requests 20 MB memory for the duration of the job, and 1 license to be reserved for 2 minutes:

```
bsub -R "rusage[mem=20, license=1:duration=2]" myjob
```

- A queue with the same resource requirements could specify:

```
RES_REQ = rusage[mem=20, license=1: duration=2]
```

- The following job requests 20 MB of memory and 50 MB of swap space for 1 hour, and 1 license to be reserved for 2 minutes:

```
bsub -R "rusage[mem=20:swp=50:duration=1h, license=1:duration=2]" myjob
```

- The following job requests 50 MB of swap space, linearly decreasing the amount reserved over a duration of 2 hours, and requests 1 license to be reserved for 2 minutes:

```
bsub -R "rusage[swp=20:duration=2h:decay=1, license=1:duration=2]" myjob
```

- The following job requests two resources with same duration but different decay:

```
bsub -R "rusage[mem=20:duration=30:decay=1, lic=1:duration=30]" myjob
```

Specify alternative usage strings

If you use more than one version of an application, you can specify the version you prefer to use together with a legacy version you can use if the preferred version is not available. Use the OR (|) expression to separate the different usage strings that define your alternative resources.

Job-level resource requirement specifications that use the || operator are merged with other rusage requirements defined at the application and queue levels.

Note:

Alternative rusage strings cannot be submitted with compound resource requirements.

Job-level examples

The following examples assume that you are running an application version 1.5 as a resource called `app_lic_v15` and the same application version 2.0.1 as a resource called `app_lic_v201`. The license key for version 2.0.1 is backward compatible with version 1.5, but the license key for version 1.5 will not work with 2.0.1

- If you can only run your job using version 2.0.1 of the application, submit the job without specifying an alternate resource. To submit a job that will only use `app_lic_v201`:

```
bsub -R "rusage[app_lic_v201=1]" myjob
```

- If you can run your job using either version of the application, try to reserve version 2.0.1 of the application. If it is not available, you can use version 1.5. To submit a job that will try `app_lic_v201` before trying `app_lic_v15`:

```
bsub -R "rusage[app_lic_v201=1||app_lic_v15=1]" myjob
```

- If different versions of an application require different system resources, you can specify other resources in your rusage strings. To submit a job that will use 20 MB of memory for `app_lic_v201` or 20 MB of memory and 50 MB of swap space for `app_lic_v15`:

```
bsub -R "rusage[mem=20:app_lic_v201=1|mem=20:swp=50:app_lic_v15=1]" myjob
```

- You can also specify alternative multi-phase memory requirements. To submit a job that will use 20 MB of memory for `app_lic_v201` or 50 MB of swap space and 50 MB of memory for 10 minutes followed by 10 MB of memory for the remainder of the job for `app_lic_v15`:

```
bsub -R "rusage[mem=20:app_lic_v201=1  
||mem=(50 10):duration=(10),swp=50:app_lic_v15=1]" myjob
```

How LSF merges rusage strings that contain the || operator

The following examples show how LSF merges job-level and queue-level rusage strings that contain the || operator.

Queue level RES_REQ=rusage...	Job level bsub -R "rusage ..."	Resulting rusage string
[mem=200:duration=180]	[w1=1 w2=1 w3=1]"	[w1=1, mem=200: duration=180 w2=1, mem=200: duration=180 w3=1, mem=200: duration=180]
[w1=1 w2=1 w3=1]	[mem=200:duration=180]"	[mem=200: duration=180, w1=1 mem=200: duration=180, w2=1 mem=200: duration=180, w3=1]

Note:

Alternative `rusage` strings cannot be submitted with compound resource requirements.

Non-batch environments

Resource reservation is only available for batch jobs. If you run jobs using only LSF Base, such as through `l srun`, LIM uses resource usage to determine the placement of jobs. Resource usage requests are used to temporarily increase the load so that a host is not overloaded. When LIM makes a placement advice, external load indices are not considered in the resource usage string. In this case, the syntax of the resource usage string is

```
res[=value]:res[=value]:...:res[=value]
```

`res` is one of the resources whose value is returned by the `lsload` command.

```
rusage[r1m=0.5:mem=20:swp=40]
```

The above example indicates that the task is expected to increase the 1-minute run queue length by 0.5, consume 20 MB of memory and 40 MB of swap space.

If no value is specified, the task is assumed to be intensive in using that resource. In this case no more than one task will be assigned to a host regardless of how many CPUs it has.

The default resource usage for a task is `r15s=1.0:r1m=1.0:r15m=1.0`. This indicates a CPU-intensive task which consumes few other resources.

Resizable jobs

Unlike the other components of a resource requirement string that only pertain to adding additional slots to a running job, `rusage` resource requirement strings affect the resource usage when slots are removed from the job as well.

When adding or removing slots from a running job:

- The amount of *slot-based* resources added to or removed from the job allocation is proportional to the change in the number of slots
- The amount of *job-based* resources is not affected by a change in the number of slots
- The amount of each *host-based* resource is proportional to the change in the number of hosts

When using multi-phase resource reservation, the job allocation is based on the phase of the resource reservation.

Note:

Resizable jobs cannot have compound resource requirements.

Duration and decay of rusage

Duration and decay of resource usage and the `||` operator affect resource allocation.

Duration or decay of a resource in the `rusage` expression is ignored when scheduling the job for the additional slots.

If a job has the following `rusage` string: `rusage[mem=100:duration=300]`, the resize request of one additional slot is scheduled on a host only if there are 100 units of memory available on that host. In this case, `mem` is a slot-based resource (`RESOURCE_RESERVE_PER_SLOT=Y` in `lsb.params`).

Once the resize operation is done, if the job has been running less than 300 seconds then additional memory will be reserved only until the job has run for 300 seconds. If the job has been running for more

than 300 seconds when the job is resized, no additional memory is reserved. The behavior is similar for decay.

The `||` operator lets you specify multiple alternative `rusage` strings, one of which is used when dispatching the job. You cannot use `bmod` to change `rusage` to a new one with a `||` operator after the job has been dispatched

For job resize, when the `||` operator is used, the resize request uses the `rusage` expression that was originally used to dispatch the job. If the `rusage` expression has been modified since the job started, the resize request is scheduled using the new single `rusage` expression.

Example 1

You want to run an autoresizable job such that every slot occupied by the job reserves 100 MB of swap space. In this case, `swp` is a slot-based resource (`RESOURCE_RESERVE_PER_SLOT=Y` in `lsb. params`). The job also needs a separate license for each host on which it runs. Each additional slot allocated to the job should reserve additional swap space, and each new host should reserve an additional license. The following job submission specifies this resource request:

```
bsub -ar -n "1, 100" -R "rusage[swp=100, license=1]" myjob
```

where `license` is a user-defined host-based resource.

Similarly, if you want to release some of the slots from a running job, resources reserved by the job are decreased appropriately. For example, for the following job submission:

```
bsub -ar -n 100 -R "rusage[swp=50: license=1]" myjob
Job <123> is submitted to default queue.
```

you can run `bresize release` to release all the slots from the job on one host:

```
bresize release "hostA" 123
```

The swap space used by the job is reduced by the number of slots used on hostA times 50 MB, and one host-based `license` resource is released from the job.

Example 2

You have a choice between two versions of an application, each version having different memory and swap space requirements on hosts and a different license (`app lic_v15` and `app lic_v201`). If you submit an autoresizable job with the `||` operator, once the job is started using one version of an application, slots added to a job during a resize operation reserve resources depending on which version of the application was originally run. For example, for the following job submission:

```
bsub -n "1, 100" -ar -R "rusage[mem=20: app lic_v201=1 || mem=20: swp=50: app lic_v15=1]" myjob
```

If the job starts with `app lic_v15`, each additional slot added in a resize operation reserves 20 MB of memory and 50 MB of swap space.

Span string

A span string specifies the locality of a parallel job. If span is omitted, LSF allocates the required processors for the job from the available set of processors.

Syntax

The span string supports the following syntax:

span[hosts=1]

Indicates that all the processors allocated to this job must be on the same host.

span[ptile=value]

Indicates the number of processors on each host that should be allocated to the job, where *value* is one of the following:

- Default `ptile` value, specified by *n* processors. In the following example, the job requests 4 processors on each available host, regardless of how many processors the host has:

```
span[ptile=4]
```

- Predefined `ptile` value, specified by '!'. The following example uses the predefined maximum job slot limit `lsb.hosts` (MXJ per host type/model) as its value:

```
span[ptile='!']
```

Tip:

If the host or host type/model does not define MXJ, the default predefined `ptile` value is 1.

Restriction:

Under bash 3.0, the exclamation mark (!) is not interpreted correctly by the shell. To use predefined `ptile` value (`ptile='!'`), use the `+H` option to disable '!' style history substitution in bash (`sh +H`).

- Predefined `ptile` value with optional multiple `ptile` values, per host type or host model:
 - For host type, you must specify `same[type]` in the resource requirement. In the following example, the job requests 8 processors on a host of type `HP` or `SGI`, and 2 processors on a host of type `LINUX`, and the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host types:

```
span[ptile='!', HP: 8, SGI: 8, LINUX: 2] same[type]
```

- For host model, you must specify `same[model]` in the resource requirement. In the following example, the job requests 4 processors on hosts of model `PC1133`, and 2 processors on hosts of model `PC233`, and the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host models:

```
span[ptile='!', PC1133: 4, PC233: 2] same[model]
```

span[hosts=-1]

Disables span setting in the queue. LSF allocates the required processors for the job from the available set of processors.

Resizable jobs

For resource requirements with `span[hosts=1]`, a resize request is limited to slots on the first-execution host of the job. This behavior eliminates the ambiguities that arise when the span expression is modified from the time that the job was originally dispatched.

For `span[ptile=n]`, the job will be allocated exactly n slots on some number of hosts, and a number between 1 and n slots (inclusive) on one host. This is true even if a range of slots is requested. For example, for the following job submission:

```
bsub -n "1, 20" -R "span[ptile=2]" sleep 10000
```

This special span behavior does not only apply to resize requests. It applies to resizable jobs only when the original allocation is made, and in making additional resize allocations.

If every host has only a single slot available, the job is allocated one slot.

Resize requests with partially filled hosts are handled so that LSF does not choose any slots on hosts already occupied by the job. For example, it is common to use the `ptile` feature with `span[ptile=1]` to schedule exclusive jobs. Another typical use is `span[ptile='!']` to make the job occupy all slots on each host it is allocated.

For a resizable job (auto-resizable or otherwise) with a range of slots requested and `span[ptile=n]`, whenever the job is allocated slots, it will receive either of the following:

- The maximum number of slots requested, comprising n slots on each of a number of hosts, and between 0 and $n-1$ (inclusive) slots on one host
- n slots on each of a number of hosts, summing to some value less than the maximum

For example, if a job requests between 1 and 14 additional slots, and `span[ptile=4]` is part of the job resource requirement string, when additional slots are allocated to the job, the job receives either of the following:

- 14 slots, with 2 slots on one host and 4 slots on each of 3 hosts
- 4, 8 or 12 slots, such that 4 slots are allocated per host of the allocation

Note:

Resizable jobs cannot have compound resource requirements.

Example

When running a parallel exclusive job, it is often desirable to specify `span[ptile=1]` so that the job is allocated at most one slot on each host. For an autoresizable job, new slots are allocated on hosts not already used by the job. The following job submission specifies this resource request:

```
bsub -x -ar -n "1, 100" -R "span[ptile=1]" myjob
```

When additional slots are allocated to a running job, the slots will be on new hosts, not already occupied by the job.

Same string

Tip:

You must have the parallel batch job scheduler plugin installed in order to use the same string.

Parallel jobs run on multiple hosts. If your cluster has heterogeneous hosts, some processes from a parallel job may for example, run on Solaris and some on SGI IRIX. However, for performance reasons you may want all processes of a job to run on the same type of host instead of having some processes run on one type of host and others on another type of host.

The *same* string specifies that all processes of a parallel job must run on hosts with the same resource.

You can specify the *same* string:

- At the job level in the resource requirement string of:
 - `bsub`
 - `bmod`
- At the queue level in `lsb.queues` in the `RES_REQ` parameter.

When queue-level, application-level, and job-level *same* sections are defined, LSF combines requirements to allocate processors.

Syntax

```
resource_name[: resource_name]...
```

You can specify any static resource.

For example, if you specify `resource1: resource2`, if hosts always have both resources, the string is interpreted as allocate processors only on hosts that have the same value for `resource1` and the same value for `resource2`.

If hosts do not always have both resources, it is interpreted as allocate processors either on hosts that have the same value for `resource1`, or on hosts that have the same value for `resource2`, or on hosts that have the same value for both `resource1` and `resource2`.

Specify multiple -R options

`bsub` accepts multiple `-R` options for the same section.

Restriction:

Compound resource requirements do not support multiple `-R` options.

You can specify multiple resource requirement strings instead of using the `&&` operator. For example:

```
bsub -R "same[type]" -R "same[model]"
```

LSF merges the multiple `-R` options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

Resizable jobs

The *same* expression ensures that the resize allocation request is dispatched to hosts that have the same resources as the first-execution host. For example, if the first execution host of a job is `SOL7` and the

resource requirement string contains `same[type]`, additional slots are allocated to the job on hosts of type SOL7.

Taking the same resource as the first-execution host avoids ambiguities that arise when the original job does not have a `same` expression defined, or has a different `same` expression when the resize request is scheduled.

For example, a parallel job may be required to have all slots on hosts of the same type or model for performance reasons. For an autoresizable job, any additional slots given to the job will be on hosts of the same type, model, or resource as those slots originally allocated to the job. The following command submits an autoresizable job such that all slots allocated in a resize operation are allocation on hosts with the same model as the original job:

```
bsub -ar -n "1, 100" -R "same[model]" myjob
```

Examples

```
bsub -n 4 -R"select[type==SGI6 || type==SOL7] same[type]" myjob
```

Run all parallel processes on the same host type. Allocate 4 processors on the same host type—either SGI IRIX, or Solaris 7, but not both.

```
bsub -n 6 -R"select[type==any] same[type:model]" myjob
```

Run all parallel processes on the same host type and model. Allocate 6 processors on any host type or model as long as all the processors are on the same host type and model.

Same string in application profiles

See [Resource requirements](#) on page 550 for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

Compute unit string

A `cu` string specifies the network architecture-based requirements of parallel jobs. `cu` sections are accepted by `bsub -R`, and by `bmod -R` for non-running jobs.

Compute unit resource requirements are not supported in compound resource requirements.

Syntax

The `cu` string supports the following syntax:

cu[type=*cu_type*]

Indicates the type of compute units the job can run on. Types are defined by `COMPUTE_UNIT_TYPES` in `lsb.params`. If `type` is not specified, the default set by `COMPUTE_UNIT_TYPES` is assumed.

cu[pref=maxavail | minavail | config]

Indicates the compute unit scheduling preference, grouping hosts by compute unit before applying a first-fit algorithm to the sorted hosts. For resource reservation, the default `pref=config` is always used.

Compute units are ordered as follows:

- `config` lists compute units in the order they appear in the `ComputeUnit` section of `lsb.hosts`. If `pref` is not specified, `pref=config` is assumed.
- `maxavail` lists compute units with more free slots first. Should compute units have equal numbers of free slots, they appear in the order listed in the `ComputeUnit` section of `lsb.hosts`.
- `minavail` lists compute units with fewer free slots first. Should compute units have equal numbers of free slots, they appear in the order listed in the `ComputeUnit` section of `lsb.hosts`.

Free slots include all available slots not occupied by running jobs.

When `pref` is used with the keyword `balance`, `balance` takes precedence.

Hosts accept jobs separated by the time interval set by `JOB_ACCEPT_INTERVAL` in `lsb.params`; jobs submitted closer together than this interval will run on different hosts regardless of the `pref` setting.

cu[maxcus=*number*]

Indicates the maximum number of compute units a job can run over. Jobs may be placed over fewer compute units if possible.

When used with `bsub -n min, max` a job is allocated the first combination satisfying both `min` and `maxcus`, while without `maxcus` a job is allocated as close to `max` as possible.

cu[usablecuslots=*number*]

Specifies the minimum number of slots a job must use on each compute unit it occupies. *number* is a non-negative integer value.

When more than one compute unit is used by a job, the final compute unit allocated can provide less than *number* slots if less are needed.

`usabl ecusl ots` and `bal ance` cannot be used together.

cu[balance]

Indicates that a job should be split evenly between compute units, with a difference in compute unit slot allocation of at most 1. A balanced allocation spans the fewest compute units possible.

When used with `bsub -n min, max` the value of *max* is disregarded.

`bal ance` and `usabl ecusl ots` cannot be used together.

When `bal ance` and `pref` are both used, `bal ance` takes precedence. The keyword `pref` is only considered if there are multiple balanced allocations spanning the same number of compute units. In this case `pref` is considered when choosing the allocation.

When `bal ance` is used with `span[ptile=X]` (for $X > 1$) a balanced allocation is one split evenly between compute units, with a difference in compute unit host allocation of at most 1.

cu[excl]

Indicates that jobs must use compute units exclusively. Exclusivity applies to the compute unit granularity specified by `type`.

Compute unit exclusivity must be enabled by `EXCLUSIVE=CU[cu_type]` in `lsb. queues`.

Resizable jobs

Auto-resizable jobs cannot be submitted with compute unit resource requirements. In the event a `bswitch` call or queue reconfiguration results in an auto-resizable job running in a queue with compute unit resource requirements, the job will no longer be auto-resizable.

Restriction:

Increasing resources allocated to resizable jobs with compute unit resource requirements is not supported.

Examples

`bsub -n 11,60 -R "cu[maxcus=2:type=enclosure]" myjob`

Spans the fewest possible compute units for a total allocation of at least 11 slots using at most 2 compute units of type `enclosure`. In contrast, without `maxcus`:

`bsub -n 11,60 myjob`

In this case the job is allocated as close to 60 slots as possible, with a minimum of 11 slots.

`bsub -n 64 -R "cu[balance:maxcus=4:type=enclosure]" myjob`

Spans the fewest possible compute units for a balanced allocation of 64 slots using 4 or less compute units of type `enclosure`. Possible balanced allocations (in order of preference) are:

- 64 slots on 1 enclosure
- 32 slots on 2 enclosures
- 22 slots on 1 enclosure and 21 slots on 2 enclosures

- 16 slots on 4 enclosures

bsub -n 64 -R "cu[excl:maxcus=8:usablecuslots=10]" myjob

Allocates 64 slots over 8 or less compute units in groups of 10 or more slots per compute unit (with one compute unit possibly using less than 10 slots). The default compute unit type set in `COMPUTE_UNIT_TYPES` is used, and are used exclusively by myjob.

bsub -n 58 -R "cu[balance:type=rack:usablecuslots=20]" myjob

Provides a balanced allocation of 58 slots with at least 20 slots in each compute unit of type rack. Possible allocations are 58 slots in 1 rack or 29 slots in 2 racks.

Jobs submitted with balance requirements choose compute units based on the `pref` keyword secondarily, as shown in the following examples where `cu1` has 5 available slots and `cu2` has 19 available slots.

bsub -n 5 -R "cu[balance:pref=minavail]"

Runs the job on compute unit `cu1` where there are the fewest available slots.

bsub -n 5 -R "cu[balance:pref=maxavail]"

Runs the job on compute unit `cu2` where there are the most available slots. In both cases the job is balanced over the fewest possible compute units.

CU string in application profiles

See [Resource requirements](#) on page 550 for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

Fairshare Scheduling

To configure any kind of fairshare scheduling, you should understand the following concepts:

- User share assignments
- Dynamic share priority
- Job dispatch order

You can configure fairshare at either host level or queue level. If you require more control, you can implement hierarchical fairshare. You can also set some additional restrictions when you submit a job.

Understand fairshare scheduling

By default, LSF considers jobs for dispatch in the same order as they appear in the queue (which is not necessarily the order in which they are submitted to the queue). This is called first-come, first-served (FCFS) scheduling.

Fairshare scheduling divides the processing power of the LSF cluster among users and queues to provide fair access to resources, so that no user or queue can monopolize the resources of the cluster and no queue will be starved.

If your cluster has many users competing for limited resources, the FCFS policy might not be enough. For example, one user could submit many long jobs at once and monopolize the cluster's resources for a long time, while other users submit urgent jobs that must wait in queues until all the first user's jobs are all done. To prevent this, use fairshare scheduling to control how resources should be shared by competing users.

Fairshare is not necessarily equal share: you can assign a higher priority to the most important users. If there are two users competing for resources, you can:

- Give all the resources to the most important user
- Share the resources so the most important user gets the most resources
- Share the resources so that all users have equal importance

Queue-level vs. host partition fairshare

You can configure fairshare at either the queue level or the host level. However, these types of fairshare scheduling are mutually exclusive. You cannot configure queue-level fairshare and host partition fairshare in the same cluster.

If you want a user's priority in one queue to depend on their activity in another queue, you must use cross-queue fairshare or host-level fairshare.

Fairshare policies

A fairshare policy defines the order in which LSF attempts to place jobs that are in a queue or a host partition. You can have multiple fairshare policies in a cluster, one for every different queue or host partition. You can also configure some queues or host partitions with fairshare scheduling, and leave the rest using FCFS scheduling.

How fairshare scheduling works

Each fairshare policy assigns a fixed number of shares to each user or group. These shares represent a fraction of the resources that are available in the cluster. The most important users or groups are the ones with the most shares. Users who have no shares cannot run jobs in the queue or host partition.

A user's dynamic priority depends on their share assignment, the dynamic priority formula, and the resources their jobs have already consumed.

The order of jobs in the queue is secondary. The most important thing is the dynamic priority of the user who submitted the job. When fairshare scheduling is used, LSF tries to place the first job in the queue that belongs to the user with the highest dynamic priority.

User share assignments

Both queue-level and host partition fairshare use the following syntax to define how shares are assigned to users or user groups.

Syntax

[user, number_shares]

Enclose each user share assignment in square brackets, as shown. Separate multiple share assignments with a space between each set of square brackets.

user

Specify users of the queue or host partition. You can assign the shares:

- to a single user (specify *user_name*)
- to users in a group, individually (specify *group_name@*) or collectively (specify *group_name*)
- to users not included in any other share assignment, individually (specify the keyword *default*) or collectively (specify the keyword *others*)

By default, when resources are assigned collectively to a group, the group members compete for the resources according to FCFS scheduling. You can use hierarchical fairshare to further divide the shares among the group members.

When resources are assigned to members of a group individually, the share assignment is recursive. Members of the group and of all subgroups always compete for the resources according to FCFS scheduling, regardless of hierarchical fairshare policies.

number_shares

Specify a positive integer representing the number of shares of cluster resources assigned to the user.

The number of shares assigned to each user is only meaningful when you compare it to the shares assigned to other users, or to the total number of shares. The total number of shares is just the sum of all the shares assigned in each share assignment.

Examples

```
[User1, 1] [GroupB, 1]
```

Assigns 2 shares: 1 to User1, and 1 to be shared by the users in GroupB. Each user in GroupB has equal importance. User1 is as important as all the users in GroupB put together. In this example, it does not matter if the number of shares is 1, 6 or 600. As long as User1 and GroupB are both assigned the same number of shares, the relationship stays the same.

```
[User1, 10] [GroupB@, 1]
```

If GroupB contains 10 users, assigns 20 shares in total: 10 to User1, and 1 to each user in GroupB. Each user in GroupB has equal importance. User1 is ten times as important as any user in GroupB.

```
[User1, 10] [User2, 9] [others, 8]
```

Assigns 27 shares: 10 to User1, 9 to User2, and 8 to the remaining users, as a group. User1 is slightly more important than User2. Each of the remaining users has equal importance.

- If there are 3 users in total, the single remaining user has all 8 shares, and is almost as important as User 1 and User 2.
- If there are 12 users in total, then 10 users compete for those 8 shares, and each of them is significantly less important than User 1 and User 2.

```
[User1, 10] [User2, 6] [default, 4]
```

The relative percentage of shares held by a user will change, depending on the number of users who are granted shares by default.

- If there are 3 users in total, assigns 20 shares: 10 to User 1, 6 to User 2, and 4 to the remaining user. User 1 has half of the available resources (10 shares out of 20).
- If there are 12 users in total, assigns 56 shares: 10 to User 1, 6 to User 2, and 4 to each of the remaining 10 users. User 1 has about a fifth of the available resources (10 shares out of 56).

Dynamic user priority

LSF calculates a *dynamic user priority* for individual users or for a group, depending on how the shares are assigned. The priority is dynamic because it changes as soon as any variable in formula changes. By default, a user's dynamic priority gradually decreases after a job starts, and the dynamic priority immediately increases when the job finishes.

How Platform LSF calculates dynamic priority

By default, LSF calculates the dynamic priority for each user based on:

- The number of shares assigned to the user
- The resources used by jobs belonging to the user:
 - Number of job slots reserved and in use
 - Run time of running jobs
 - Cumulative actual CPU time (not normalized), adjusted so that recently used CPU time is weighted more heavily than CPU time used in the distant past

If you enable additional functionality, the formula can also involve additional resources used by jobs belonging to the user:

- Decayed run time of running jobs
- Historical run time of finished jobs
- Committed run time, specified at job submission with the -W option of bsub, or in the queue with the RUNLIMIT parameter in lsb. queues
- Memory usage adjustment made by the fairshare plugin (libfairshareadjust.*).

How Platform LSF measures fairshare resource usage

LSF measures resource usage differently, depending on the type of fairshare:

- For user-based fairshare:
 - For queue-level fairshare, LSF measures the resource consumption of all the user's jobs in the queue. This means a user's dynamic priority can be different in every queue.
 - For host partition fairshare, LSF measures resource consumption for all the user's jobs that run on hosts in the host partition. This means a user's dynamic priority is the same in every queue that uses hosts in the same partition.
- For queue-based fairshare, LSF measures the resource consumption of all jobs in each queue.

Default dynamic priority formula

By default, LSF calculates dynamic priority according to the following formula:

$$\text{dynamic priority} = \text{number_shares} / (\text{cpu_time} * \text{CPU_TIME_FACTOR} + \text{run_time} * \text{RUN_TIME_FACTOR} + (1 + \text{job_slots}) * \text{RUN_JOB_FACTOR} + \text{fairshare_adjustment} * \text{FAIRSHARE_ADJUSTMENT_FACTOR})$$

Note:

The maximum value of dynamic user priority is 100 times the number of user shares (if the denominator in the calculation is less than 0.01, LSF rounds up to 0.01).

For *cpu_time*, *run_time*, and *job_slots*, LSF uses the total resource consumption of all the jobs in the queue or host partition that belong to the user or group.

number_shares

The number of shares assigned to the user.

cpu_time

The cumulative CPU time used by the user (measured in hours). LSF calculates the cumulative CPU time using the actual (not normalized) CPU time and a decay factor such that 1 hour of recently-used CPU time decays to 0.1 hours after an interval of time specified by HIST_HOURS in *l sb. params* (5 hours by default).

run_time

The total run time of running jobs (measured in hours).

job_slots

The number of job slots reserved and in use.

fairshare_adjustment

The adjustment calculated by the fairshare adjustment plugin (*l i bfai rshareadj ust. **).

Configure the default dynamic priority

You can give additional weight to the various factors in the priority calculation by setting the following parameters for the queue in *l sb. queues* or for the cluster in *l sb. params*. When the queue value is not defined, the cluster-wide value from *l sb. params* is used.

- CPU_TIME_FACTOR
- RUN_TIME_FACTOR
- RUN_JOB_FACTOR
- FAIRSHARE_ADJUSTMENT_FACTOR
- HIST_HOURS

If you modify the parameters used in the dynamic priority formula, it affects every fairshare policy in the cluster.

CPU_TIME_FACTOR

The CPU time weighting factor.

Default: 0.7

RUN_TIME_FACTOR

The run time weighting factor.

Default: 0.7

RUN_JOB_FACTOR

The job slots weighting factor.

*Default: 3***FAIRSHARE_ADJUSTMENT_FACTOR**The fairshare plugin (`libfairshareadjust.*`) weighting factor.*Default: 0***HIST_HOURS**

Interval for collecting resource consumption history

Default: 5

Customize the dynamic priority

In some cases the dynamic priority equation may require adjustments beyond the run time, cpu time, and job slot dependencies provided by default. The fairshare adjustment plugin is open source and can be customized once you identify specific requirements for dynamic priority.

All information used by the default priority equation (except the user shares) is passed to the fairshare plugin. In addition, the fairshare plugin is provided with current memory use over the entire cluster and the average memory allocated to a slot in the cluster.

Note:

If you modify the parameters used in the dynamic priority formula, it affects every fairshare policy in the cluster. The fairshare adjustment plugin (`libfairshareadjust.*`) is not queue-specific. Parameter settings passed to the fairshare adjustment plugin are those defined in `lsb.params`.

Example

Jobs assigned to a single slot on a host can consume host memory to the point that other slots on the hosts are left unusable. The default dynamic priority calculation considers job slots used, but doesn't account for unused job slots effectively blocked by another job.

The fairshare adjustment plugin example code provided by Platform LSF is found in the examples directory of your installation, and implements a memory-based dynamic priority adjustment as follows:

$$\text{fairshare adjustment} = (1 + \text{slots}) * ((\text{used_memory} / \text{used_slots}) / (\text{slot_memory} * \text{THRESHOLD}))$$
used_slots

The number of job slots in use by started jobs.

used_memory

The total memory in use by started jobs.

slot_memory

The average amount of memory that exists per slot in the cluster.

THRESHOLD

The memory threshold set in the fairshare adjustment plugin.

Use time decay and committed run time

By default, as a job is running, the dynamic priority decreases gradually until the job has finished running, then increases immediately when the job finishes.

In some cases this can interfere with fairshare scheduling if two users who have the same priority and the same number of shares submit jobs at the same time.

To avoid these problems, you can modify the dynamic priority calculation by using one or more of the following weighting factors:

- Run time decay
- Historical run time decay
- Committed run time

Historical run time decay

By default, historical run time does not affect the dynamic priority. You can configure LSF so that the user's dynamic priority increases *gradually* after a job finishes. After a job is finished, its run time is saved as the historical run time of the job and the value can be used in calculating the dynamic priority, the same way LSF considers historical CPU time in calculating priority. LSF applies a decaying algorithm to the historical run time to gradually increase the dynamic priority over time after a job finishes.

Configure historical run time

1. Specify `ENABLE_HIST_RUN_TIME=Y` for the queue in `l sb. queues` or for the cluster in `l sb. params`.

Historical run time is added to the calculation of the dynamic priority so that the formula becomes the following:

```
dynamic priority = number_shares / (cpu_time * CPU_TIME_FACTOR +
(historical_run_time + run_time) * RUN_TIME_FACTOR + (1 + job_slots) *
RUN_JOB_FACTOR + fairshare_adjustment(struct*shareAdjustPair)
*FAIRSHARE_ADJUSTMENT_FACTOR)
```

historical_run_time—(measured in hours) of finished jobs accumulated in the user's share account file. LSF calculates the historical run time using the actual run time of finished jobs and a decay factor such that 1 hour of recently-used run time decays to 0.1 hours after an interval of time specified by `HIST_HOURS` in `l sb. params` (5 hours by default).

How mbatchd reconfiguration and restart affects historical run time

After restarting or reconfiguring `mbatchd`, the historical run time of finished jobs might be different, since it includes jobs that may have been cleaned from `mbatchd` before the restart. `mbatchd` restart only reads recently finished jobs from `l sb. events`, according to the value of `CLEAN_PERIOD` in `l sb. params`. Any jobs cleaned before restart are lost and are not included in the new calculation of the dynamic priority.

Example

The following fairshare parameters are configured in `l sb. params`:

```
CPU_TIME_FACTOR = 0
RUN_JOB_FACTOR = 0
RUN_TIME_FACTOR = 1
FAIRSHARE_ADJUSTMENT_FACTOR = 0
```

Note that in this configuration, only run time is considered in the calculation of dynamic priority. This simplifies the formula to the following:

$\text{dynamic priority} = \text{number_shares} / (\text{run_time} * \text{RUN_TIME_FACTOR})$

Without the historical run time, the dynamic priority increases suddenly as soon as the job finishes running because the run time becomes zero, which gives no chance for jobs pending for other users to start.

When historical run time is included in the priority calculation, the formula becomes:

$\text{dynamic priority} = \text{number_shares} / (\text{historical_run_time} + \text{run_time}) * \text{RUN_TIME_FACTOR}$

Now the dynamic priority increases gradually as the historical run time decays over time.

Run time decay

In a cluster running jobs of varied length, a user running only short jobs may always have a higher priority than a user running a long job. This can happen when historical run time decay is applied, decreasing the impact of the completed short jobs but not the longer job that is still running. To correct this, you can configure LSF to decay the run time of a job that is still running in the same manner historical run time decays.

Once a job is complete, the decayed run time is transferred to the historical run time where the decay continues. This equalizes the effect of short and long running jobs on user dynamic priority.

Note:

Running `badmadmin reconfig` or restarting `mbatchd` during a job's run time results in the decayed run time being recalculated. When a suspended job using run time decay is resumed, the decay time is based on the elapsed time.

Configure run time decay

1. Specify `HIST_HOURS` for the queue in `l sb. queues` or for the cluster in `l sb. params`.
2. Specify `RUN_TIME_DECAY=Y` for the queue in `l sb. queues` or for the cluster in `l sb. params`.

The run time used in the calculation of the dynamic priority so that the formula becomes the following:

```
dynamic priority = number_shares / (cpu_time * CPU_TIME_FACTOR +
(historical_run_time + run_time) * RUN_TIME_FACTOR + (1 + job_slots) *
RUN_JOB_FACTOR + fairshare_adjustment(struct*shareAdjustPair)
*FAIRSHARE_ADJUSTMENT_FACTOR)
```

run_time—(measured in hours) of running jobs accumulated in the user's share account file. LSF calculates the decayed run time using the actual run time of running jobs and a decay factor such that 1 hour of recently-used run time decays to 0.1 hours after an interval of time specified by `HIST_HOURS` for the queue in `l sb. queues` or for the cluster in `l sb. params` (5 hours by default).

Committed run time weighting factor

Committed run time is the run time requested at job submission with the `-W` option of `bsub`, or in the queue configuration with the `RUNLIMIT` parameter. By default, committed run time does not affect the dynamic priority.

While the job is running, the actual run time is subtracted from the committed run time. The user's dynamic priority decreases *immediately* to its lowest expected value, and is maintained at that value until the job finishes. Job run time is accumulated as usual, and historical run time, if any, is decayed.

When the job finishes, the committed run time is set to zero and the actual run time is added to the historical run time for future use. The dynamic priority increases gradually until it reaches its maximum value.

Providing a weighting factor in the run time portion of the dynamic priority calculation prevents a “job dispatching burst” where one user monopolizes job slots because of the latency in computing run time.

Limitation

If you use queue-level fairshare, and a running job has a committed run time, you should not switch that job to or from a fairshare queue (using `bswitch`). The fairshare calculations will not be correct.

Run time displayed by `bqueues` and `bhpart`

The run time displayed by `bqueues` and `bhpart` is the sum of the actual, accumulated run time and the historical run time, but does not include the committed run time.

Configure committed run time

1. Set a value for the `COMMITTED_RUN_TIME_FACTOR` parameter for the queue in `l sb. queues` or for the cluster in `l sb. params`. You should also specify a `RUN_TIME_FACTOR`, to prevent the user’s dynamic priority from increasing as the run time increases.

If you have also enabled the use of historical run time, the dynamic priority is calculated according to the following formula:

$$\text{dynamic priority} = \text{number_shares} / (\text{cpu_time} * \text{CPU_TIME_FACTOR} + (\text{historical_run_time} + \text{run_time}) * \text{RUN_TIME_FACTOR} + (\text{committed_run_time} - \text{run_time}) * \text{COMMITTED_RUN_TIME_FACTOR} + (1 + \text{job_slots}) * \text{RUN_JOB_FACTOR} + \text{fairshare_adjustment}(\text{struct} * \text{shareAdjustPair}) * \text{FAIRSHARE_ADJUSTMENT_FACTOR})$$

committed_run_time—The run time requested at job submission with the `-W` option of `bsub`, or in the queue configuration with the `RUNLIMIT` parameter. This calculation measures the committed run time in hours.

In the calculation of a user’s dynamic priority, `COMMITTED_RUN_TIME_FACTOR` determines the relative importance of the committed run time in the calculation. If the `-W` option of `bsub` is not specified at job submission and a `RUNLIMIT` has not been set for the queue, the committed run time is not considered.

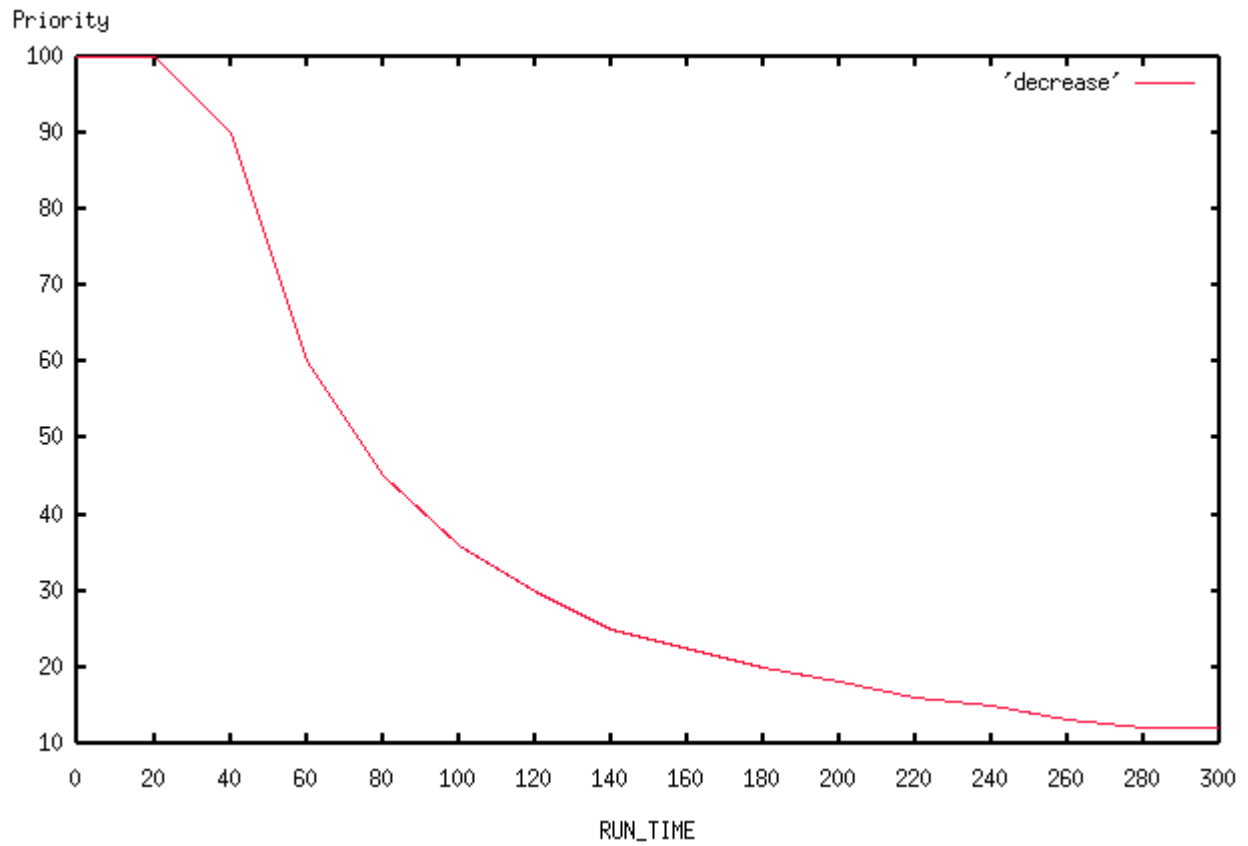
`COMMITTED_RUN_TIME_FACTOR` can be any positive value between 0.0 and 1.0. The default value set in `l sb. params` is 0.0. As the value of `COMMITTED_RUN_TIME_FACTOR` approaches 1.0, more weight is given to the committed run time in the calculation of the dynamic priority.

Example

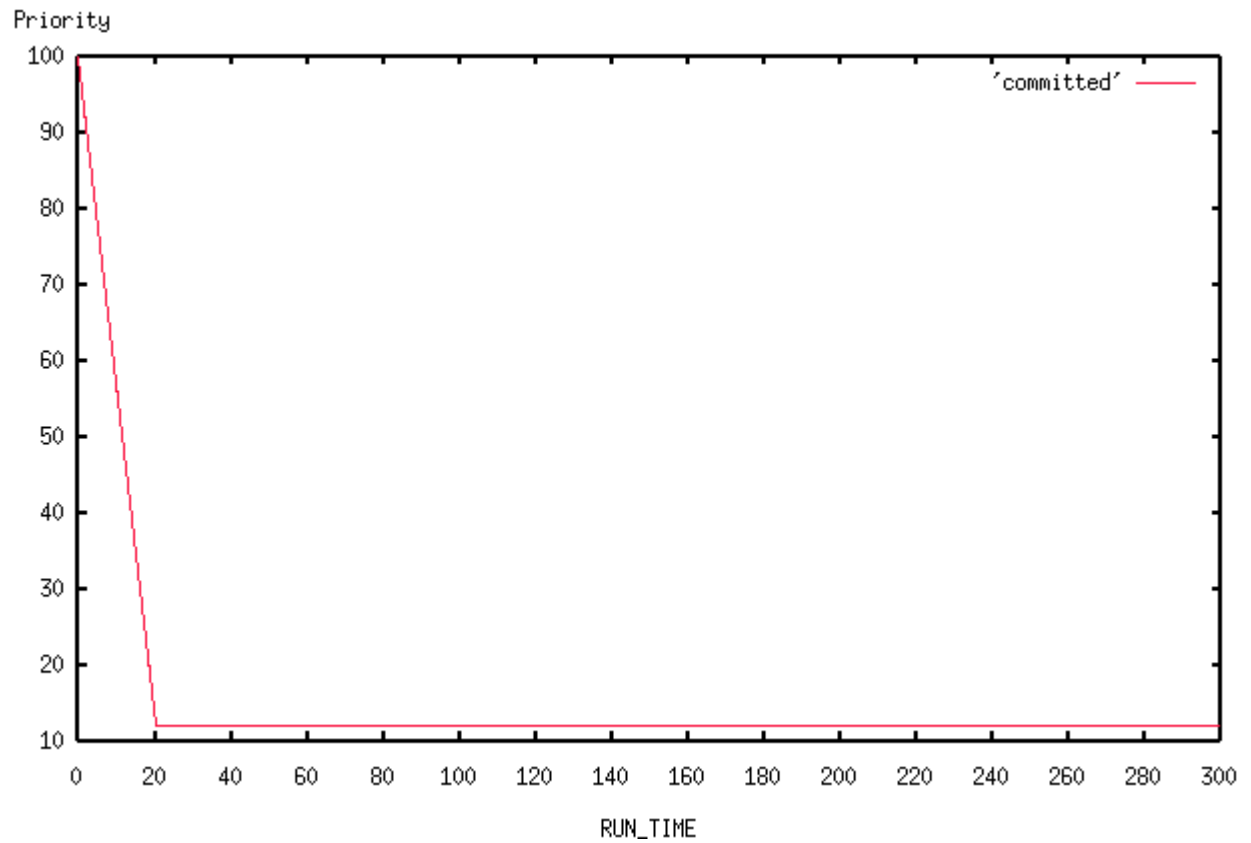
The following fairshare parameters are configured in `l sb. params`:

```
CPU_TIME_FACTOR = 0
RUN_JOB_FACTOR = 0
RUN_TIME_FACTOR = 1
FAIRSHARE_ADJUSTMENT_FACTOR = 0
COMMITTED_RUN_TIME_FACTOR = 1
```

Without a committed run time factor, dynamic priority for the job owner drops gradually while a job is running:



When a committed run time factor is included in the priority calculation, the dynamic priority drops as soon as the job is dispatched, rather than gradually dropping as the job runs:



How fairshare affects job dispatch order

Within a queue, jobs are dispatched according to the queue's scheduling policy.

- For FCFS queues, the dispatch order depends on the order of jobs in the queue (which depends on job priority and submission time, and can also be modified by the job owner).
- For fairshare queues, the dispatch order depends on dynamic share priority, then order of jobs in the queue (which is not necessarily the order in which they are submitted to the queue).

A user's priority gets higher when they use less than their fair share of the cluster's resources. When a user has the highest priority, LSF considers one of their jobs first, even if other users are ahead of them in the queue.

If there are only one user's jobs pending, and you do not use hierarchical fairshare, then there is no resource contention between users, so the fairshare policies have no effect and jobs are dispatched as usual.

Job dispatch order among queues of equivalent priority

The order of dispatch depends on the order of the queues in the queue configuration file. The first queue in the list is the first to be scheduled.

Jobs in a fairshare queue are always considered as a group, so the scheduler attempts to place all jobs in the queue before beginning to schedule the next queue.

Jobs in an FCFS queue are always scheduled along with jobs from other FCFS queues of the same priority (as if all the jobs belonged to the same queue).

Example

In a cluster, queues A, B, and C are configured in that order and have equal queue priority.

Jobs with equal job priority are submitted to each queue in this order: C B A B A.

- If all queues are FCFS queues, order of dispatch is C B A B A (queue A is first; queues B and C are the same priority as A; all jobs are scheduled in FCFS order).
- If all queues are fairshare queues, order of dispatch is AA BB C (queue A is first; all jobs in the queue are scheduled; then queue B, then C).
- If A and C are fairshare, and B is FCFS, order of dispatch is AA B B C (queue A jobs are scheduled according to user priority; then queue B jobs are scheduled in FCFS order; then queue C jobs are scheduled according to user priority)
- If A and C are FCFS, and B is fairshare, order of dispatch is C A A BB (queue A is first; queue A and C jobs are scheduled in FCFS order, then queue B jobs are scheduled according to user priority)
- If any of these queues uses cross-queue fairshare, the other queues must also use cross-queue fairshare and belong to the same set, or they cannot have the same queue priority.

Host partition user-based fairshare

User-based fairshare policies configured at the host level handle resource contention across multiple queues.

You can define a different fairshare policy for every host partition. If multiple queues use the host partition, a user has the same priority across multiple queues.

To run a job on a host that has fairshare, users must have a share assignment (USER_SHARES in the HostPartition section of `lsb.hosts`). Even cluster administrators cannot submit jobs to a fairshare host if they do not have a share assignment.

View host partition information

1. Use `bhpart` to view the following information:

- Host partitions configured in your cluster
- Number of shares (for each user or group in a host partition)
- Dynamic share priority (for each user or group in a host partition)
- Number of started jobs
- Number of reserved jobs
- CPU time, in seconds (cumulative CPU time for all members of the group, recursively)
- Run time, in seconds (historical and actual run time for all members of the group, recursively)

```
% bhpart Partition1
HOST_PARTITION_NAME: Partition1
HOSTS: hostA hostB hostC

SHARE_INFO_FOR: Partition1/
USER/GROUP SHARES PRIORITY STARTED RESERVED CPU_TIME RUN_TIME
group1      100      5.440      5         0       200.0    1324
```

Configure host partition fairshare scheduling

1. To configure host partition fairshare, define a host partition in `lsb.hosts`.

Use the following format.

```
Begin HostPartition
HPART_NAME = Partition1
HOSTS = hostA hostB ~hostC
USER_SHARES = [groupA@, 3] [groupB, 7] [default, 1]
End HostPartition
```

- A host cannot belong to multiple partitions.
- Optional: Use the reserved host name `all` to configure a single partition that applies to all hosts in a cluster.
- Optional: Use the not operator (`~`) to exclude hosts or host groups from the list of hosts in the host partition.
- Hosts in a host partition cannot participate in queue-based fairshare.

Hosts that are not included in any host partition are controlled by FCFS scheduling policy instead of fairshare scheduling policy.

Queue-level user-based fairshare

User-based fairshare policies configured at the queue level handle resource contention among users in the same queue. You can define a different fairshare policy for every queue, even if they share the same hosts. A user's priority is calculated separately for each queue.

To submit jobs to a fairshare queue, users must be allowed to use the queue (USERS in `l sb. queues`) and must have a share assignment (FAIRSHARE in `l sb. queues`). Even cluster and queue administrators cannot submit jobs to a fairshare queue if they do not have a share assignment.

If the default user group set in `DEFAULT_USER_GROUP` (`l sb. params`) does not have shares assigned in a fairshare queue, jobs can still run from the default user group and are charged to the highest priority account the user can access in the queue. The default user group should have shares assigned in most fairshare queues to ensure jobs run smoothly.

Job submitted with a user group (`bsub -G`) which is no longer valid when the job runs charge the default user group (if defined) or the highest priority account the user can access in the queue (if no default user group is defined). In such cases `bj obs -l` output shows the submission user group, along with the updated SAAP (share attribute account path).

By default, user share accounts are created for users in each user group, whether they have active jobs or not. When many user groups in the fairshare policy have `al l` as a member, the memory used creating user share accounts on `mbatchd` startup may be noticeable. Limit the number of share accounts created to active users (and all members of the default user group) by setting `LSB_SACCT_ONE_UG=Y` in `l sf. conf`.

View queue-level fairshare information

1. To find out if a queue is a fairshare queue, run `bqueues -l`. If you see "USER_SHARES" in the output, then a fairshare policy is configured for the queue.

Configure queue-level fairshare

1. To configure a fairshare queue, define FAIRSHARE in `l sb. queues` and specify a share assignment for all users of the queue:

```
FAIRSHARE = USER_SHARES[ [ user, number_shares] . . . ]
```

- You must specify at least one user share assignment.
- Enclose the list in square brackets, as shown.
- Enclose each user share assignment in square brackets, as shown.

Cross-queue user-based fairshare

User-based fairshare policies configured at the queue level handle resource contention across multiple queues.

Apply the same fairshare policy to several queues

With cross-queue fairshare, the same user-based fairshare policy can apply to several queues at the same time. You define the fairshare policy in a *master queue* and list *slave queues* to which the same fairshare policy applies; slave queues inherit the same fairshare policy as your master queue. For job scheduling purposes, this is equivalent to having one queue with one fairshare tree.

In this way, if a user submits jobs to different queues, user priority is calculated by taking into account all the jobs the user has submitted across the defined queues.

To submit jobs to a fairshare queue, users must be allowed to use the queue (USERS in l sb. queues) and must have a share assignment (FAIRSHARE in l sb. queues). Even cluster and queue administrators cannot submit jobs to a fairshare queue if they do not have a share assignment.

User and queue priority

By default, a user has the same priority across the master and slave queues. If the same user submits several jobs to these queues, user priority is calculated by taking into account all the jobs the user has submitted across the master-slave set.

If DISPATCH_ORDER=QUEUE is set in the master queue, jobs are dispatched according to queue priorities first, then user priority. This avoids having users with higher fairshare priority getting jobs dispatched from low-priority queues.

Jobs from users with lower fairshare priorities who have pending jobs in higher priority queues are dispatched before jobs in lower priority queues. Jobs in queues having the same priority are dispatched according to user priority.

Queues that are not part of the ordered cross-queue fairshare can have any priority. Their priority can fall within the priority range of cross-queue fairshare queues and they can be inserted between two queues using the same fairshare tree.

View cross-queue fairshare information

1. Run `bqueues -l` to know if a queue is part of cross-queue fairshare.

The FAIRSHARE_QUEUES parameter indicates cross-queue fairshare. The first queue listed in the FAIRSHARE_QUEUES parameter is the master queue—the queue in which fairshare is configured; all other queues listed inherit the fairshare policy from the master queue.

All queues that participate in the same cross-queue fairshare display the same fairshare information (SCHEDULING POLICIES, FAIRSHARE_QUEUES, USER_SHARES, SHARE_INFO_FOR) when `bqueues -l` is used. Fairshare information applies to all the jobs running in all the queues in the master-slave set.

`bqueues -l` also displays DISPATCH_ORDER in the master queue if it is defined.

bqueues										
QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
normal	30	Open: Active	-	-	-	-	1	1	0	0
short	40	Open: Active	-	4	2	-	1	0	1	0
license	50	Open: Active	10	1	1	-	1	0	1	0
bqueues -l normal										
QUEUE: normal -- For normal low priority jobs, running only if hosts are lightly loaded. This is the default queue.										


```

PARAMETERS/STATISTICS
PRIO NICE STATUS      MAX JL/U JL/P JL/H NJOBS  PEND  RUN  SSUSP  USUSP  RSV
30    20  Open: Inact_Win -    -    -    -    -    1    1    0    0    0    0
SCHEDULING PARAMETERS
r15s  r1m  r15m  ut      pg    io    ls    it      tmp    swp    mem
loadSched -    -    -    -    -    -    -    -    -    -    -
loadStop  -    -    -    -    -    -    -    -    -    -    -

      cpuspeed  bandwidth
loadSched -    -
loadStop  -    -

SCHEDULING POLICIES: FAIRSHARE
FAIRSHARE_QUEUES: normal short license
USER_SHARES: [user1, 100] [default, 1]
SHARE_INFO_FOR: normal/
USER/GROUP  SHARES  PRIORITY  STARTED  RESERVED  CPU_TIME  RUN_TIME  ADJUST user1      100
9.645      2      0      0.2      7034      0.000
USERS: all users
HOSTS: all
...

bqueues -l short
QUEUE: short
PARAMETERS/STATISTICS
PRIO NICE STATUS
MAX JL/U JL/P JL/H NJOBS  PEND  RUN  SSUSP  USUSP  RSV
40    20  Open: Inact_Win -    4    2    -    1    0      1    0    0    0
SCHEDULING PARAMETERS
r15s  r1m  r15m  ut      pg    io    ls    it      tmp    swp    mem
loadSched -    -    -    -    -    -    -    -    -    -    -
loadStop  -    -    -    -    -    -    -    -    -    -    -

      cpuspeed  bandwidth
loadSched -    -
loadStop  -    -

SCHEDULING POLICIES: FAIRSHARE
FAIRSHARE_QUEUES: normal short license
USER_SHARES: [user1, 100] [default, 1]
SHARE_INFO_FOR: short/
USER/GROUP  SHARES  PRIORITY  STARTED  RESERVED  CPU_TIME  RUN_TIME
user1      100      9.645      2      0      0.2      7034
USERS: all users
HOSTS: all
...

```

Configure cross-queue fairshare

- FAIRSHARE must be defined in the master queue. If it is also defined in the queues listed in FAIRSHARE_QUEUES, it will be ignored.
- Cross-queue fairshare can be defined more than once within lsb. queues. You can define several sets of master-slave queues. However, a queue cannot belong to more than one master-slave set. For example, you can define:
 - In master queue normal : FAIRSHARE_QUEUES=short license
 - In master queue priority: FAIRSHARE_QUEUES=night owners

You cannot, however, define night, owners, or priority as slaves in the normal queue; or normal, short and license as slaves in the priority queue; or short, license, night, owners as master queues of their own.

- Cross-queue fairshare cannot be used with host partition fairshare. It is part of queue-level fairshare.

- Decide to which queues in your cluster cross-queue fairshare will apply.

For example, in your cluster you may have the queues normal, priority, short, and license and you want cross-queue fairshare to apply only to normal, license, and short.

2. Define fairshare policies in your master queue.

In the queue you want to be the master, for example `normal`, define the following in `lsb.queues`:

- `FAIRSHARE` and specify a share assignment for all users of the queue.
- `FAIRSHARE_QUEUES` and list slave queues to which the defined fairshare policy will also apply
- `PRIORITY` to indicate the priority of the queue.

```
Begin Queue
QUEUE_NAME      = queue1
PRIORITY        = 30
NICE            = 20
FAIRSHARE       = USER_SHARES[ [user1, 100] [default, 1]]
FAIRSHARE_QUEUES = queue2 queue3
DESCRIPTION     = For normal low priority jobs, running only if hosts are lightly
loaded.
End Queue
```

3. In all the slave queues listed in `FAIRSHARE_QUEUES`, define all queue values as desired.

For example:

```
Begin Queue
QUEUE_NAME      = queue2
PRIORITY        = 40
NICE            = 20
UJOB_LIMIT     = 4
PJOB_LIMIT     = 2
End Queue

Begin Queue
QUEUE_NAME      = queue3
PRIORITY        = 50
NICE            = 10
PREEMPTION     = PREEMPTIVE
QJOB_LIMIT     = 10
UJOB_LIMIT     = 1
PJOB_LIMIT     = 1
End Queue
```

Control job dispatch order in cross-queue fairshare

DISPATCH_ORDER parameter (lsb.queues)

Use `DISPATCH_ORDER=QUEUE` in the master queue to define an *ordered* cross-queue fairshare set. `DISPATCH_ORDER` indicates that jobs are dispatched according to the order of queue priorities, not user fairshare priority.

Priority range in cross-queue fairshare

By default, the range of priority defined for queues in cross-queue fairshare cannot be used with any other queues. The priority of queues that are not part of the cross-queue fairshare cannot fall between the priority range of cross-queue fairshare queues.

For example, you have 4 queues: `queue1`, `queue2`, `queue3`, and `queue4`. You configure cross-queue fairshare for `queue1`, `queue2`, and `queue3`, and assign priorities of 30, 40, 50 respectively. The priority of `queue4` (which is not part of the cross-queue fairshare) cannot fall between 30 and 50, but it can be any number up to 29 or higher than 50. It does not matter if `queue4` is a fairshare queue or FCFS queue.

If `DISPATCH_ORDER=QUEUE` is set in the master queue, queues that are not part of the ordered cross-queue fairshare can have any priority. Their priority can fall within the priority range of cross-queue fairshare queues and they can be inserted between two queues using the same fairshare tree. In the example above, `queue4` can have any priority, including a priority falling between the priority range of the cross-queue fairshare queues (30-50).

Jobs from equal priority queues

- If two or more *non-fairshare* queues have the same priority, their jobs are dispatched first-come, first-served based on submission time or job ID as if they come from the same queue.
- If two or more *fairshare* queues have the same priority, jobs are dispatched in the order the queues are listed in `lsb. queues`.

Hierarchical user-based fairshare

For both queue and host partitions, hierarchical user-based fairshare lets you allocate resources to users in a hierarchical manner.

By default, when shares are assigned to a group, group members compete for resources according to FCFS policy. If you use hierarchical fairshare, you control the way shares that are assigned collectively are divided among group members.

If groups have subgroups, you can configure additional levels of share assignments, resulting in a multi-level share tree that becomes part of the fairshare policy.

How hierarchical fairshare affects dynamic share priority

When you use hierarchical fairshare, the dynamic share priority formula does not change, but LSF measures the resource consumption for all levels of the share tree. To calculate the dynamic priority of a group, LSF uses the resource consumption of all the jobs in the queue or host partition that belong to users in the group and all its subgroups, recursively.

How hierarchical fairshare affects job dispatch order

LSF uses the dynamic share priority of a user or group to find out which user's job to run next. If you use hierarchical fairshare, LSF works through the share tree from the top level down, and compares the dynamic priority of users and groups at each level, until the user with the highest dynamic priority is a single user, or a group that has no subgroups.

View hierarchical share information for a group

1. Use `bugroup -l` to find out if you belong to a group, and what the share distribution is.

```
bugroup -l
GROUP_NAME: group1
USERS: group2/ group3/
SHARES: [group2, 20] [group3, 10]

GROUP_NAME: group2
USERS: user1 user2 user3
SHARES: [others, 10] [user3, 4]

GROUP_NAME: group3
USERS: all
SHARES: [user2, 10] [default, 5]
```

This command displays all the share trees that are configured, even if they are not used in any fairshare policy.

View hierarchical share information for a host partition

By default, `bhpart` displays only the top level share accounts associated with the partition.

1. Use `bhpart -r` to display the group information recursively.

The output lists all the groups in the share tree, starting from the top level, and displays the following information:

- Number of shares
- Dynamic share priority (LSF compares dynamic priorities of users who belong to same group, at the same level)
- Number of started jobs

- Number of reserved jobs
- CPU time, in seconds (cumulative CPU time for all members of the group, recursively)
- Run time, in seconds (historical and actual run time for all members of the group, recursively)

bhpart -r Partition1

HOST_PARTITION_NAME: Partition1

HOSTS: HostA

SHARE_INFO_FOR: Partition1/

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME
group1	40	1.867	5	0	48.4	17618
group2	20	0.775	6	0	607.7	24664

SHARE_INFO_FOR: Partition1/group2/

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME
user1	8	1.144	1	0	9.6	5108
user2	2	0.667	0	0	0.0	0
others	1	0.046	5	0	598.1	19556

Configure hierarchical fairshare

To define a hierarchical fairshare policy, configure the top-level share assignment in `lsb.queues` or `lsb.hosts`, as usual. Then, for any group of users affected by the fairshare policy, configure a share tree in the `UserGroup` section of `lsb.users`. This specifies how shares assigned to the group, collectively, are distributed among the individual users or subgroups.

If shares are assigned to members of any group individually, using `@`, there can be no further hierarchical fairshare within that group. The shares are assigned recursively to all members of all subgroups, regardless of further share distributions defined in `lsb.users`. The group members and members of all subgroups compete for resources according to FCFS policy.

You can choose to define a hierarchical share tree for some groups but not others. If you do not define a share tree for any group or subgroup, members compete for resources according to FCFS policy.

Configure a share tree

1. Group membership is already defined in the `UserGroup` section of `lsb.users`. To configure a share tree, use the `USER_SHARES` column to describe how the shares are distributed in a hierarchical manner. Use the following format.

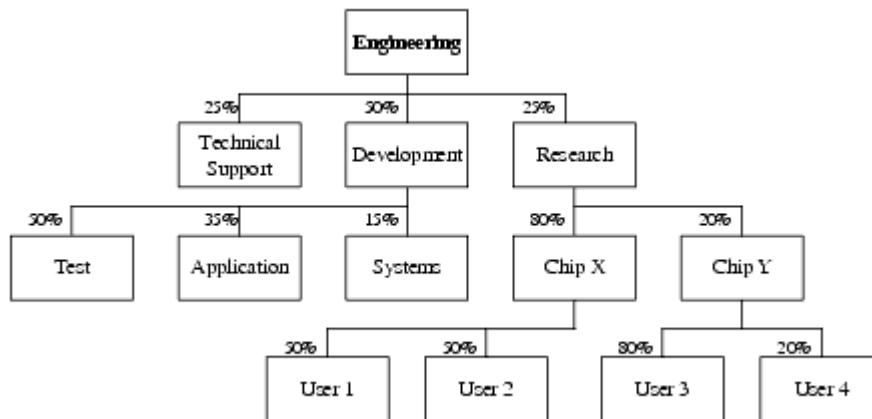
```

Begin UserGroup
GROUP_NAME    GROUP_MEMBER    USER_SHARES
GroupB        (User1 User2)              ()
GroupC        (User3 User4)    ([User3, 3] [User4, 4])
GroupA        (GroupB GroupC User5) ([User5, 1] [default, 10])
End UserGroup

```

- User groups must be defined before they can be used (in the `GROUP_MEMBER` column) to define other groups.
- Shares (in the `USER_SHARES` column) can only be assigned to user groups in the `GROUP_MEMBER` column.
- The keyword `all` refers to all users, not all user groups.
- Enclose the share assignment list in parentheses, as shown, even if you do not specify any user share assignments.

An Engineering queue or host partition organizes users hierarchically, and divides the shares as shown. It does not matter what the actual number of shares assigned at each level is.



The Development group gets the largest share (50%) of the resources in the event of contention. Shares assigned to the Development group can be further divided among the Systems, Application, and Test groups, which receive 15%, 35%, and 50%, respectively. At the lowest level, individual users compete for these shares as usual.

One way to measure a user's importance is to multiply their percentage of the resources at every level of the share tree. For example, User 1 is entitled to 10% of the available resources ($.50 \times .80 \times .25 = .10$) and User 3 is entitled to 4% ($.80 \times .20 \times .25 = .04$). However, if Research has the highest dynamic share priority among the 3 groups at the top level, and Chip Y has a higher dynamic priority than Chip X, the next comparison is between User 3 and User 4, so the importance of User 1 is not relevant. The dynamic priority of User 1 is not even calculated at this point.

Queue-based fairshare

When a priority is set in a queue configuration, a high priority queue tries to dispatch as many jobs as it can before allowing lower priority queues to dispatch any job. Lower priority queues are blocked until the higher priority queue cannot dispatch any more jobs. However, it may be desirable to give some preference to lower priority queues and regulate the flow of jobs from the queue.

Queue-based fairshare allows flexible slot allocation per queue as an alternative to absolute queue priorities by enforcing a *soft job slot limit* on a queue. This allows you to organize the priorities of your work and tune the number of jobs dispatched from a queue so that no single queue monopolizes cluster resources, leaving other queues waiting to dispatch jobs.

You can balance the distribution of job slots among queues by configuring a ratio of jobs waiting to be dispatched from each queue. LSF then attempts to dispatch a certain percentage of jobs from each queue, and does not attempt to drain the highest priority queue entirely first.

When queues compete, the allocated slots per queue are kept within the limits of the configured share. If only one queue in the pool has jobs, that queue can use all the available resources and can span its usage across all hosts it could potentially run jobs on.

Manage pools of queues

You can configure your queues into a *pool*, which is a named group of queues using the same set of hosts. A pool is entitled to a slice of the available job slots. You can configure as many pools as you need, but each pool must use the same set of hosts. There can be queues in the cluster that do not belong to any pool yet share some hosts used by a pool.

How LSF allocates slots for a pool of queues

During job scheduling, LSF orders the queues within each pool based on the shares the queues are entitled to. The number of running jobs (or job slots in use) is maintained at the percentage level specified for the queue. When a queue has no pending jobs, leftover slots are redistributed to other queues in the pool with jobs pending.

The total number of slots in each pool is constant; it is equal to the number of slots in use plus the number of free slots to the maximum job slot limit configured either in `lsb. hosts (MXJ)` or in `lsb. resources` for a host or host group. The accumulation of slots in use by the queue is used in ordering the queues for dispatch.

Job limits and host limits are enforced by the scheduler. For example, if LSF determines that a queue is eligible to run 50 jobs, but the queue has a job limit of 40 jobs, no more than 40 jobs will run. The remaining 10 job slots are redistributed among other queues belonging to the same pool, or make them available to other queues that are configured to use them.

Accumulated slots in use

As queues run the jobs allocated to them, LSF accumulates the slots each queue has used and decays this value over time, so that each queue is not allocated more slots than it deserves, and other queues in the pool have a chance to run their share of jobs.

Interaction with other scheduling policies

- Queues participating in a queue-based fairshare pool cannot be preemptive or preemptable.
- You should not configure slot reservation (`SLOT_RESERVE`) in queues that use queue-based fairshare.

- Cross-queue user-based fairshare (FAIRSHARE_QUEUES) can undo the dispatching decisions of queue-based fairshare. Cross-queue user-based fairshare queues should not be part of a queue-based fairshare pool.
- When MAX_SLOTS_IN_POOL, SLOT_RESERVE, and BACKFILL are defined (in 1 sb. queues) for the same queue, jobs in the queue cannot backfill using slots reserved by other jobs in the same queue.

Examples

Three queues using two hosts each with maximum job slot limit of 6 for a total of 12 slots to be allocated:

- queue1 shares 50% of slots to be allocated = $2 * 6 * 0.5 = 6$ slots
- queue2 shares 30% of slots to be allocated = $2 * 6 * 0.3 = 3.6 \rightarrow 4$ slots
- queue3 shares 20% of slots to be allocated = $2 * 6 * 0.2 = 2.4 \rightarrow 3$ slots; however, since the total cannot be more than 12, queue3 is actually allocated only 2 slots.

Four queues using two hosts each with maximum job slot limit of 6 for a total of 12 slots; queue4 does not belong to any pool.

- queue1 shares 50% of slots to be allocated = $2 * 6 * 0.5 = 6$
- queue2 shares 30% of slots to be allocated = $2 * 6 * 0.3 = 3.6 \rightarrow 4$
- queue3 shares 20% of slots to be allocated = $2 * 6 * 0.2 = 2.4 \rightarrow 2$
- queue4 shares no slots with other queues

queue4 causes the total number of slots to be less than the total free and in use by the queue1, queue2, and queue3 that do belong to the pool. It is possible that the pool may get all its shares used up by queue4, and jobs from the pool will remain pending.

queue1, queue2, and queue3 belong to one pool, queue6, queue7, and queue8 belong to another pool, and queue4 and queue5 do not belong to any pool.

LSF orders the queues in the two pools from higher-priority queue to lower-priority queue (queue1 is highest and queue8 is lowest):

```
queue1 -> queue2 -> queue3 -> queue6 -> queue7 -> queue8
```

If the queue belongs to a pool, jobs are dispatched from the highest priority queue first. Queues that do not belong to any pool (queue4 and queue5) are merged into this ordered list according to their priority, but LSF dispatches as many jobs from the non-pool queues as it can:

```
queue1 -> queue2 -> queue3 -> queue4 -> queue5 -> queue6 ->
queue7 -> queue8
```


Slot allocation per queue

Configure as many pools as you need in `l sb. queues`.

SLOT_SHARE parameter

The `SLOT_SHARE` parameter represents the percentage of running jobs (job slots) in use from the queue. `SLOT_SHARE` must be greater than zero (0) and less than or equal to 100.

The sum of `SLOT_SHARE` for all queues in the pool does not need to be 100%. It can be more or less, depending on your needs.

SLOT_POOL parameter

The `SLOT_POOL` parameter is the name of the pool of job slots the queue belongs to. A queue can only belong to one pool. All queues in the pool must share the same set of hosts.

MAX_SLOTS_IN_POOL parameter

The optional parameter `MAX_SLOTS_IN_POOL` sets a limit on the number of slots available for a slot pool. This parameter is defined in the first queue of the slot pool in `l sb. queues`.

USE_PRIORITY_IN_POOL parameter

The optional parameter `USE_PRIORITY_IN_POOL` enables LSF scheduling to allocate any unused slots in the pool to jobs based on the job priority across the queues in the slot pool. This parameter is defined in the first queue of the slot pool in `l sb. queues`.

Host job slot limit

The hosts used by the pool must have a maximum job slot limit, configured either in `l sb. hosts (MXJ)` or `l sb. resources (HOSTS and SLOTS)`.

Configure slot allocation per queue

1. For each queue that uses queue-based fairshare, define the following in `l sb. queues`:
 - a) `SLOT_SHARE`
 - b) `SLOT_POOL`
2. Optional: Define the following in `l sb. queues` for each queue that uses queue-based fairshare:
 - a) `HOSTS` to list the hosts that can receive jobs from the queue
If no hosts are defined for the queue, the default is all hosts.

Tip:
Hosts for queue-based fairshare cannot be in a host partition.

 - b) `PRIORITY` to indicate the priority of the queue.
3. Optional: Define the following in `l sb. queues` for the first queue in each slot pool:
 - a) `MAX_SLOTS_IN_POOL` to set the maximum number of slots available for use in the slot pool.
 - b) `USE_PRIORITY_IN_POOL` to allow allocation of any unused slots in the slot pool based on the job priority across queues in the slot pool.
4. For each host used by the pool, define a maximum job slot limit, either in `l sb. hosts (MXJ)` or `l sb. resources (HOSTS and SLOTS)`.

Configure two pools

The following example configures pool A with three queues, with different shares, using the hosts in host group groupA:

```
Begin Queue
QUEUE_NAME = queue1
PRIORITY   = 50
SLOT_POOL  = pool A
SLOT_SHARE = 50
HOSTS      = groupA
...
End Queue

Begin Queue
QUEUE_NAME = queue2
PRIORITY   = 48
SLOT_POOL  = pool A
SLOT_SHARE = 30
HOSTS      = groupA
...
End Queue

Begin Queue
QUEUE_NAME = queue3
PRIORITY   = 46
SLOT_POOL  = pool A
SLOT_SHARE = 20
HOSTS      = groupA
...
End Queue
```

The following configures a pool named pool B, with three queues with equal shares, using the hosts in host group groupB, setting a maximum number of slots for the pool (MAX_SLOTS_IN_POOL) and enabling a second round of scheduling based on job priority across the queues in the pool (USE_PRIORITY_IN_POOL):

```
Begin Queue
QUEUE_NAME = queue4
PRIORITY   = 44
SLOT_POOL  = pool B
SLOT_SHARE = 30
HOSTS      = groupB
MAX_SLOTS_IN_POOL=128
USE_PRIORITY_IN_POOL=Y
...
End Queue

Begin Queue
QUEUE_NAME = queue5
PRIORITY   = 43
SLOT_POOL  = pool B
SLOT_SHARE = 30
HOSTS      = groupB
...
End Queue

Begin Queue
QUEUE_NAME = queue6
PRIORITY   = 42
SLOT_POOL  = pool B
SLOT_SHARE = 30
HOSTS      = groupB
...
End Queue
```

View configured job slot share

1. Use `bqueues -l` to show the job slot share (SLOT_SHARE) and the hosts participating in the share pool (SLOT_POOL):

```

QUEUE: queue1
PARAMETERS/STATISTICS
PRI0 NICE STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SSUSP  USUSP  RSV
50   20  Open: Active      -   -   -   -   0    0    0    0    0    0
Interval for a host to accept two jobs is 0 seconds

STACKLIMIT MEMLIMIT
2048 K      5000 K

SCHEDULING PARAMETERS
loadSched   r15s  r1m  r15m  ut      pg    io    ls    it    tmp    swp    mem
loadStop    -    -    -    -      -    -    -    -    -    -    -

              cpuspeed    bandwidth
loadSched    -            -
loadStop     -            -

USERS:  all users
HOSTS:  groupA/
SLOT_SHARE: 50%
SLOT_POOL: pool A

```

View slot allocation of running jobs

1. Use `bhosts`, `bmgroup`, and `bqueues` to verify how LSF maintains the configured percentage of running jobs in each queue.

The queues configurations above use the following hosts groups:

```
bmgroup -r
GROUP_NAME  HOSTS
groupA      hosta hostb hostc
groupB      hostd hoste hostf
```

Each host has a maximum job slot limit of 5, for a total of 15 slots available to be allocated in each group:

```
bhosts
HOST_NAME  STATUS  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV
hosta      ok      -     5     5     5     0     0     0
hostb      ok      -     5     5     5     0     0     0
hostc      ok      -     5     5     5     0     0     0
hostd      ok      -     5     5     5     0     0     0
hoste      ok      -     5     5     5     0     0     0
hostf      ok      -     5     5     5     0     0     0
```

Pool named pool A contains `queue1`, `queue2`, and `queue3`. pool B contains `queue4`, `queue5`, and `queue6`. The `bqueues` command shows the number of running jobs in each queue:

```
bqueues
QUEUE_NAME  PRI O  STATUS  MAX  JL/U  JL/P  JL/H  NJOBS  PEND  RUN  SUSP
queue1      50   Open: Active  -   -   -   -   492  484   8   0
queue2      48   Open: Active  -   -   -   -   500  495   5   0
queue3      46   Open: Active  -   -   -   -   498  496   2   0
queue4      44   Open: Active  -   -   -   -   985  980   5   0
queue5      43   Open: Active  -   -   -   -   985  980   5   0
queue6      42   Open: Active  -   -   -   -   985  980   5   0
```

As a result: `queue1` has a 50% share and can run 8 jobs; `queue2` has a 30% share and can run 5 jobs; `queue3` has a 20% share and is entitled 3 slots, but since the total number of slots available must be 15, it can run 2 jobs; `queue4`, `queue5`, and `queue6` all share 30%, so 5 jobs are running in each queue.

Typical slot allocation scenarios

3 queues with SLOT_SHARE 50%, 30%, 20%, with 15 job slots

This scenario has three phases:

1. All three queues have jobs running, and LSF assigns the number of slots to queues as expected: 8, 5, 2. Though queue Genova deserves 3 slots, the total slot assignment must be 15, so Genova is allocated only 2 slots:

bqueues										
QUEUE_NAME	PRI O	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
Roma	50	Open: Active	-	-	-	-	1000	992	8	0
Verona	48	Open: Active	-	-	-	-	995	990	5	0
Genova	48	Open: Active	-	-	-	-	996	994	2	0

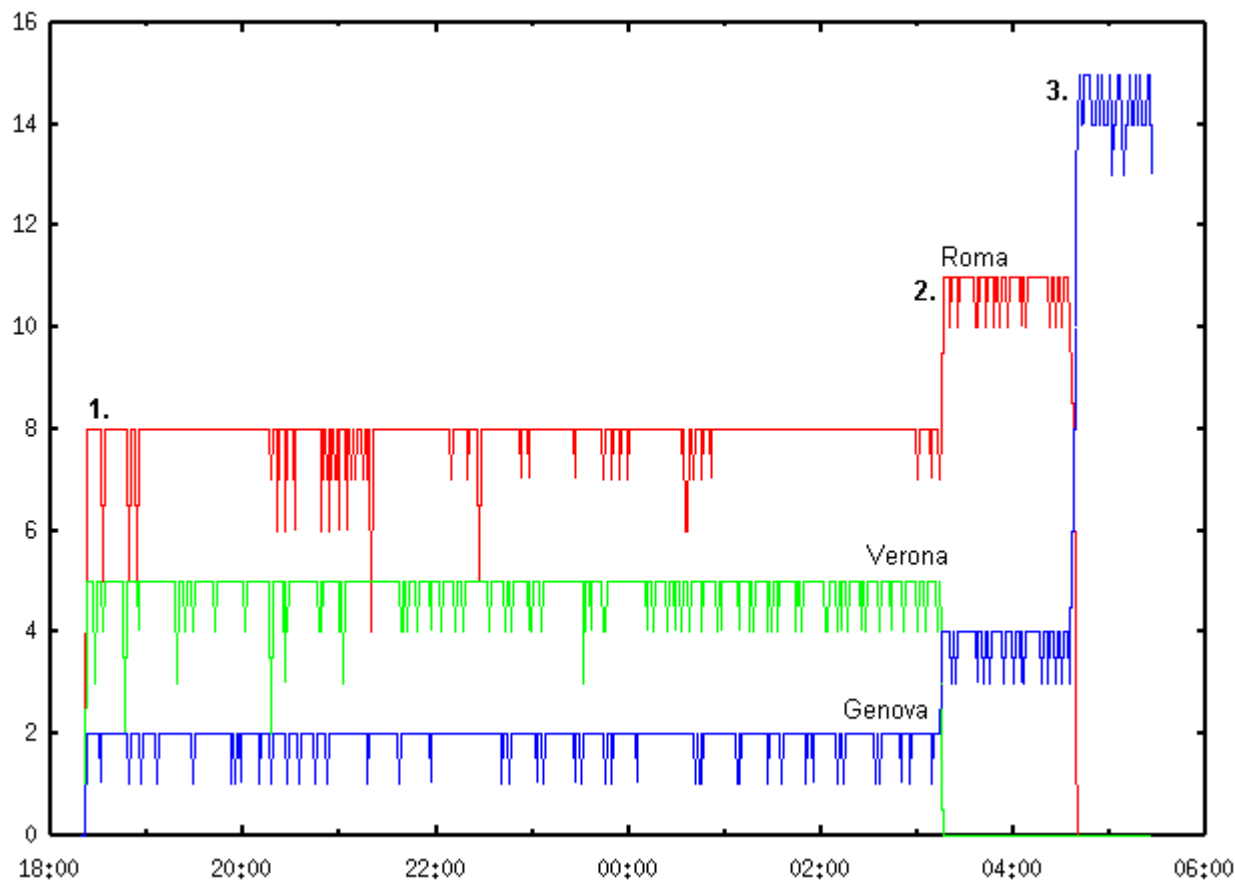
2. When queue Verona has done its work, queues Roma and Genova get their respective shares of 8 and 2. This leaves 4 slots to be redistributed to queues according to their shares: 50% (2 slots) to Roma, 20% (1 slot) to Genova. The one remaining slot is assigned to queue Roma again:

bqueues										
QUEUE_NAME	PRI O	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
Roma	50	Open: Active	-	-	-	-	231	221	11	0
Verona	48	Open: Active	-	-	-	-	0	0	0	0
Genova	48	Open: Active	-	-	-	-	496	491	4	0

3. When queues Roma and Verona have no more work to do, Genova can use all the available slots in the cluster:

bqueues										
QUEUE_NAME	PRI O	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
Roma	50	Open: Active	-	-	-	-	0	0	0	0
Verona	48	Open: Active	-	-	-	-	0	0	0	0
Genova	48	Open: Active	-	-	-	-	475	460	15	0

The following figure illustrates phases 1, 2, and 3:

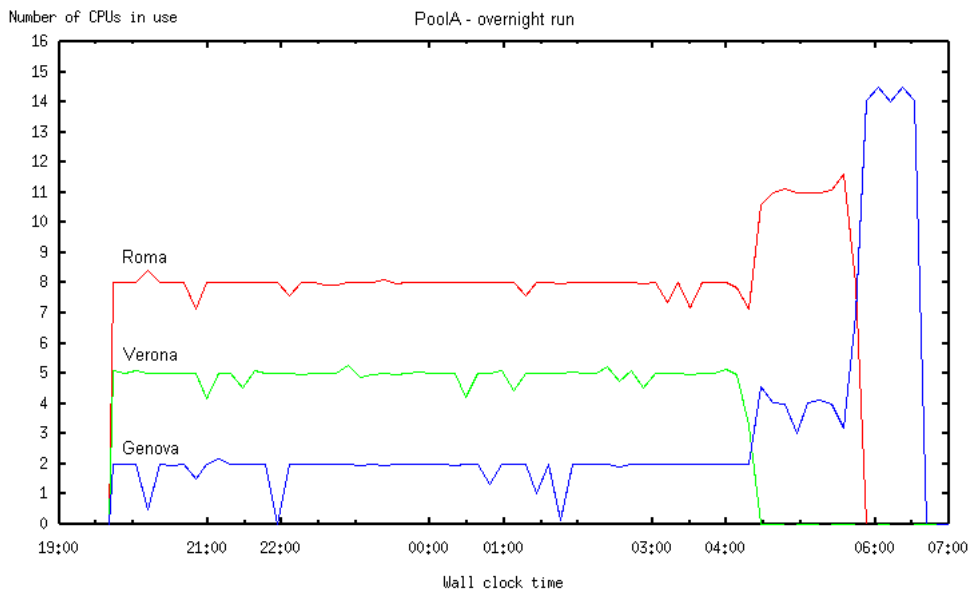


2 pools, 30 job slots, and 2 queues out of any pool

- pool A uses 15 slots and contains queues Roma (50% share, 8 slots), Verona (30% share, 5 slots), and Genova (20% share, 2 remaining slots to total 15).
- pool B with 15 slots containing queues Pi sa (30% share, 5 slots), Venezi a (30% share, 5 slots), and Bol ogna (30% share, 5 slots).
- Two other queues Mi l ano and Parma do not belong to any pool, but they can use the hosts of pool B. The queues from Mi l ano to Bol ogna all have the same priority.

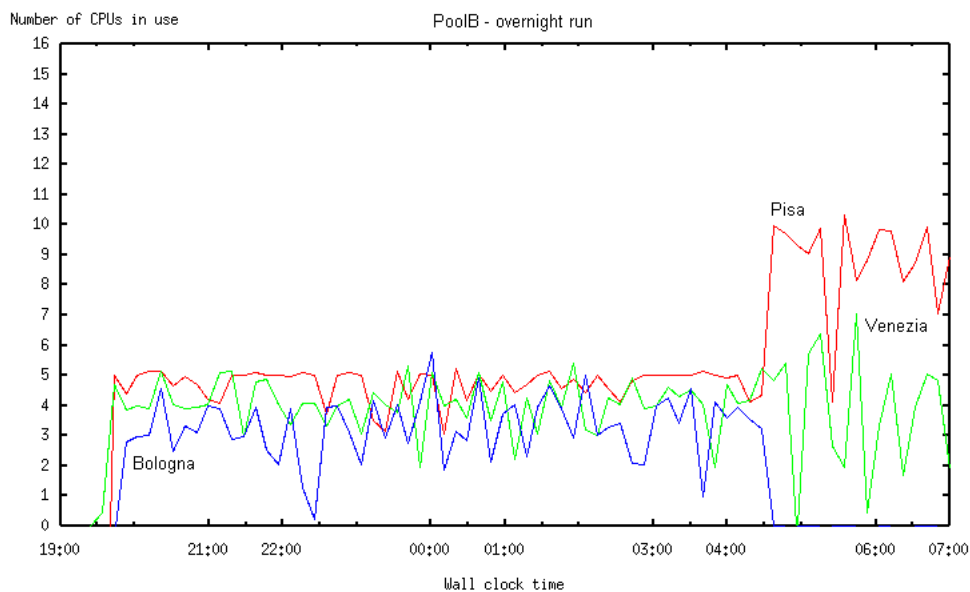
The queues Mi l ano and Parma run very short jobs that get submitted periodically in bursts. When no jobs are running in them, the distribution of jobs looks like this:

QUEUE_NAME	PRI O	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
Roma	50	Open: Active	-	-	-	-	1000	992	8	0
Verona	48	Open: Active	-	-	-	-	1000	995	5	0
Genova	48	Open: Active	-	-	-	-	1000	998	2	0
Pi sa	44	Open: Active	-	-	-	-	1000	995	5	0
Mi l ano	43	Open: Active	-	-	-	-	2	2	0	0
Parma	43	Open: Active	-	-	-	-	2	2	0	0
Venezi a	43	Open: Active	-	-	-	-	1000	995	5	0
Bol ogna	43	Open: Active	-	-	-	-	1000	995	5	0



When Milano and Parma have jobs, their higher priority reduces the share of slots free and in use by Venezia and Bologna:

QUEUE_NAME	PRI O	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
Roma	50	Open: Active	-	-	-	-	992	984	8	0
Verona	48	Open: Active	-	-	-	-	993	990	3	0
Genova	48	Open: Active	-	-	-	-	996	994	2	0
Pisa	44	Open: Active	-	-	-	-	995	990	5	0
Milano	43	Open: Active	-	-	-	-	10	7	3	0
Parma	43	Open: Active	-	-	-	-	11	8	3	0
Venezia	43	Open: Active	-	-	-	-	995	995	2	0
Bologna	43	Open: Active	-	-	-	-	995	995	2	0



Round-robin slot distribution — 13 queues and 2 pools

- Pool pool A has 3 hosts each with 7 slots for a total of 21 slots to be shared. The first 3 queues are part of the pool pool A sharing the CPUs with proportions 50% (11 slots), 30% (7 slots) and 20% (3 remaining slots to total 21 slots).
- The other 10 queues belong to pool pool B, which has 3 hosts each with 7 slots for a total of 21 slots to be shared. Each queue has 10% of the pool (3 slots).

The initial slot distribution looks like this:

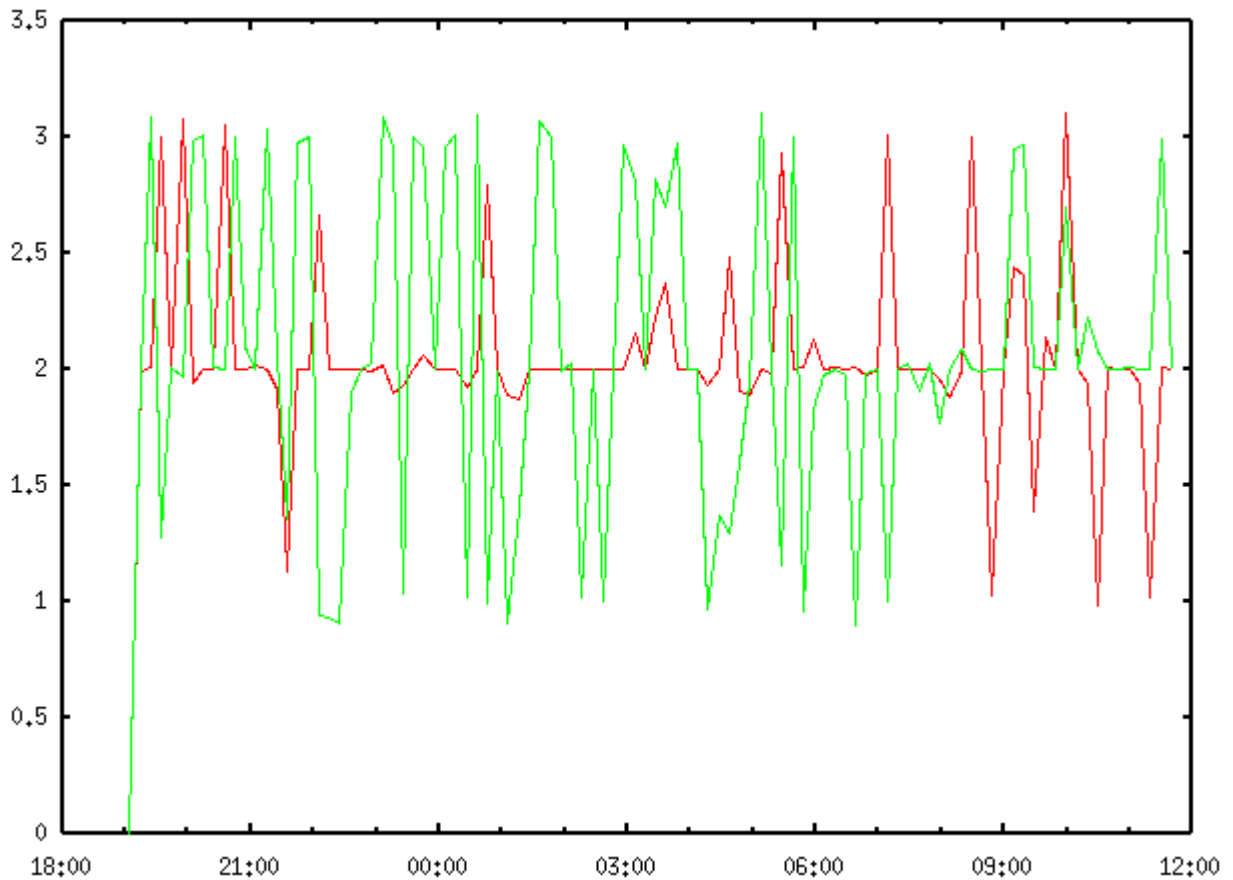
bqueues											
QUEUE_NAME	PRI O	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP	
Roma	50	Open: Active	-	-	-	-	15	6	11	0	
Verona	48	Open: Active	-	-	-	-	25	18	7	0	
Genova	47	Open: Active	-	-	-	-	460	455	3	0	
Pi sa	44	Open: Active	-	-	-	-	264	261	3	0	
Mi l ano	43	Open: Active	-	-	-	-	262	259	3	0	
Parma	42	Open: Active	-	-	-	-	260	257	3	0	
Bol o gna	40	Open: Active	-	-	-	-	260	257	3	0	
Sora	40	Open: Active	-	-	-	-	261	258	3	0	
Ferrara	40	Open: Active	-	-	-	-	258	255	3	0	
Napol i	40	Open: Active	-	-	-	-	259	256	3	0	
Li vorno	40	Open: Active	-	-	-	-	258	258	0	0	
Pal ermo	40	Open: Active	-	-	-	-	256	256	0	0	
Venezi a	4	Open: Active	-	-	-	-	255	255	0	0	

Initially, queues Li vorno, Pal ermo, and Venezi a in pool B are not assigned any slots because the first 7 higher priority queues have used all 21 slots available for allocation.

As jobs run and each queue accumulates used slots, LSF favors queues that have not run jobs yet. As jobs finish in the first 7 queues of pool B, slots are redistributed to the other queues that originally had no jobs (queues Li vorno, Pal ermo, and Venezi a). The total slot count remains 21 in all queues in pool B.

bqueues											
QUEUE_NAME	PRI O	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP	
Roma	50	Open: Active	-	-	-	-	15	6	9	0	
Verona	48	Open: Active	-	-	-	-	25	18	7	0	
Genova	47	Open: Active	-	-	-	-	460	455	5	0	
Pi sa	44	Open: Active	-	-	-	-	263	261	2	0	
Mi l ano	43	Open: Active	-	-	-	-	261	259	2	0	
Parma	42	Open: Active	-	-	-	-	259	257	2	0	
Bol o gna	40	Open: Active	-	-	-	-	259	257	2	0	
Sora	40	Open: Active	-	-	-	-	260	258	2	0	
Ferrara	40	Open: Active	-	-	-	-	257	255	2	0	
Napol i	40	Open: Active	-	-	-	-	258	256	2	0	
Li vorno	40	Open: Active	-	-	-	-	258	256	2	0	
Pal ermo	40	Open: Active	-	-	-	-	256	253	3	0	
Venezi a	4	Open: Active	-	-	-	-	255	253	2	0	

The following figure illustrates the round-robin distribution of slot allocations between queues Li vorno and Pal ermo:



How LSF rebalances slot usage

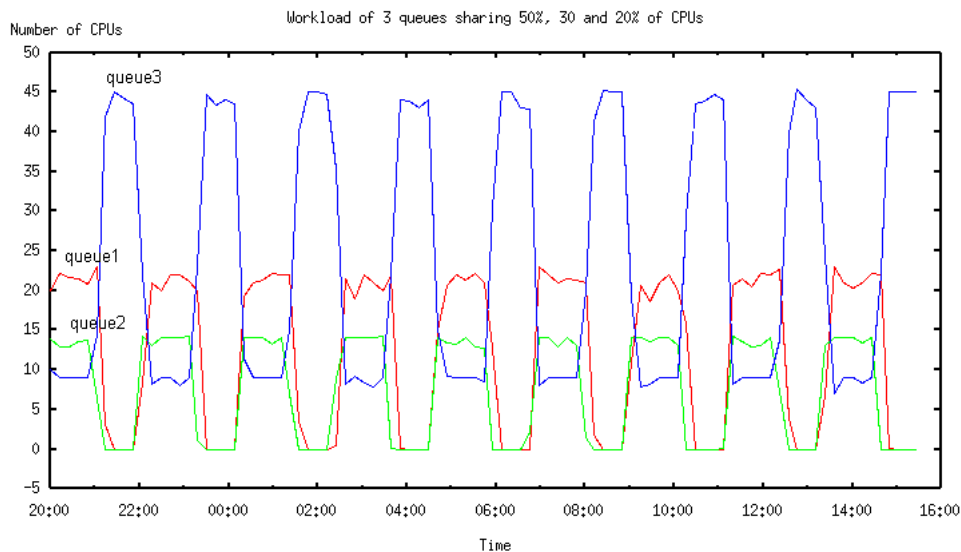
In the following examples, job runtime is not equal, but varies randomly over time.

3 queues in one pool with 50%, 30%, 20% shares

A pool configures 3 queues:

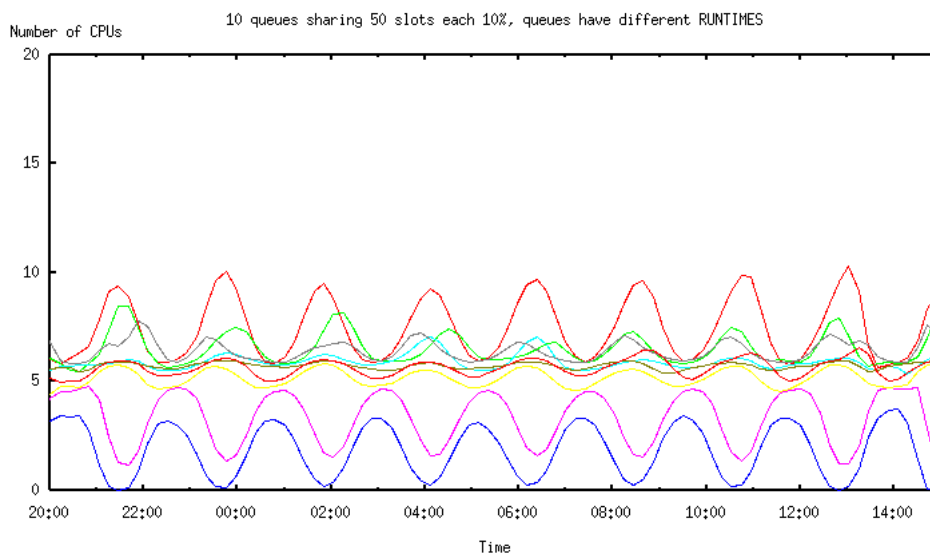
- queue1 50% with short-running jobs
- queue2 20% with short-running jobs
- queue3 30% with longer running jobs

As queue1 and queue2 finish their jobs, the number of jobs in queue3 expands, and as queue1 and queue2 get more work, LSF rebalances the usage:



10 queues sharing 10% each of 50 slots

In this example, queue1 (the curve with the highest peaks) has the longer running jobs and so has less accumulated slots in use over time. LSF accordingly rebalances the load when all queues compete for jobs to maintain a configured 10% usage share.



Users affected by multiple fairshare policies

If you belong to multiple user groups, which are controlled by different fairshare policies, each group probably has a different dynamic share priority at any given time. By default, if any one of these groups becomes the highest priority user, you could be the highest priority user in that group, and LSF would attempt to place your job.

To restrict the number of fairshare policies that will affect your job, submit your job and specify a single user group that your job will belong to, for the purposes of fairshare scheduling. LSF will not attempt to dispatch this job unless the group you specified is the highest priority user. If you become the highest priority user because of some other share assignment, another one of your jobs might be dispatched, but not this one.

Submit a job and specify a user group

Associate a job with a user group for fairshare scheduling.

1. Use `bsub -G` and specify a group that you belong to.

For example:

User 1 shares resources with groupA and groupB. User 1 is also a member of groupA, but not any other groups.

User 1 submits a job:

```
bsub sleep 100
```

By default, the job could be considered for dispatch if either User 1 or GroupA has highest dynamic share priority.

User 1 submits a job and associates the job with GroupA:

```
bsub -G groupA sleep 100
```

If User 1 is the highest priority user, this job will not be considered.

- User 1 can only associate the job with a group that he is a member of.
- User 1 cannot associate the job with his individual user account, because `bsub -G` only accepts group names.

Example with hierarchical fairshare

In the share tree, User 1 shares resources with GroupA at the top level. GroupA has 2 subgroups, B and C. GroupC has 1 subgroup, GroupD. User 1 also belongs to GroupB and GroupC.

User 1 submits a job:

```
bsub sleep 100
```

By default, the job could be considered for dispatch if either User 1, GroupB, or GroupC has highest dynamic share priority.

User 1 submits a job and associates the job with GroupB:

```
bsub -G groupB sleep 100
```

If User 1 or GroupC is the highest priority user, this job will not be considered.

- User 1 cannot associate the job with GroupC, because GroupC includes a subgroup.

- User 1 cannot associate the job with his individual user account, because `bsub -G` only accepts group names.

Ways to configure fairshare

Global fairshare

Global fairshare balances resource usage across the entire cluster according to one single fairshare policy. Resources used in one queue affect job dispatch order in another queue.

If two users compete for resources, their dynamic share priority is the same in every queue.

Configure global fairshare

1. To configure global fairshare, you must use host partition fairshare. Use the keyword `all` to configure a single partition that includes all the hosts in the cluster.

```
Begin HostPartition
HPART_NAME =GlobalPartition
HOSTS = all
USER_SHARES = [groupA@, 3] [groupB, 7] [default, 1]
End HostPartition
```

Chargeback fairshare

Chargeback fairshare lets competing users share the same hardware resources according to a fixed ratio. Each user is entitled to a specified portion of the available resources.

If two users compete for resources, the most important user is entitled to more resources.

Configure chargeback fairshare

1. To configure chargeback fairshare, put competing users in separate user groups and assign a fair number of shares to each group.

Example

Suppose two departments contributed to the purchase of a large system. The engineering department contributed 70 percent of the cost, and the accounting department 30 percent. Each department wants to get their money's worth from the system.

1. Define 2 user groups in `lsb. users`, one listing all the engineers, and one listing all the accountants.

```
Begin UserGroup
Group_Name Group_Member
eng_users (user6 user4)
acct_users (user2 user5)
End UserGroup
```

2. Configure a host partition for the host, and assign the shares appropriately.

```
Begin HostPartition
HPART_NAME = big_servers
HOSTS = hostH
USER_SHARES = [eng_users, 7] [acct_users, 3]
End HostPartition
```

Equal share

Equal share balances resource usage equally between users.

Configure equal share

1. To configure equal share, use the keyword `default` to define an equal share for every user.

```
Begin HostPartition
HPART_NAME = equal_share_partition
HOSTS = all
USER_SHARES = [default, 1]
End HostPartition
```

Priority user and static priority fairshare

There are two ways to configure fairshare so that a more important user's job always overrides the job of a less important user, regardless of resource use.

- **Priority User Fairshare:** Dynamic priority is calculated as usual, but more important and less important users are assigned a drastically different number of shares, so that resource use has virtually no effect on the dynamic priority: the user with the overwhelming majority of shares always goes first. However, if two users have a similar or equal number of shares, their resource use still determines which of them goes first. This is useful for isolating a group of high-priority or low-priority users, while allowing other fairshare policies to operate as usual most of the time.
- **Static Priority Fairshare:** Dynamic priority is no longer dynamic, because resource use is ignored. The user with the most shares always goes first. This is useful to configure multiple users in a descending order of priority.

Configure priority user fairshare

A queue is shared by key users and other users.

Priority user fairshare gives priority to important users, so their jobs override the jobs of other users. You can still use fairshare policies to balance resources among each group of users.

If two users compete for resources, and one of them is a priority user, the priority user's job always runs first.

1. Define a user group for priority users in `lsb. users`, naming it accordingly.

For example, `key_users`.

2. Configure fairshare and assign the overwhelming majority of shares to the key users:

```
Begin Queue
QUEUE_NAME = production
FAIRSHARE = USER_SHARES[[key_users@, 2000] [others, 1]]
...
End Queue
```

In the above example, key users have 2000 shares each, while other users together have only 1 share. This makes it virtually impossible for other users' jobs to get dispatched unless none of the users in the `key_users` group has jobs waiting to run.

If you want the same fairshare policy to apply to jobs from all queues, configure host partition fairshare in a similar way.

Configure static priority fairshare

Static priority fairshare assigns resources to the user with the most shares. Resource usage is ignored.

1. To implement static priority fairshare, edit `lsb. params` and set all the weighting factors used in the dynamic priority formula to 0 (zero).

- Set CPU_TIME_FACTOR to 0
- Set RUN_TIME_FACTOR to 0
- Set RUN_JOB_FACTOR to 0
- Set COMMITTED_RUN_TIME_FACTOR to 0
- Set FAIRSHARE_ADJUSTMENT_FACTOR to 0

The results is: $\text{dynamic priority} = \text{number_shares} / 0.01$ (if the denominator in the dynamic priority calculation is less than 0.01, LSF rounds up to 0.01)

If two users compete for resources, the most important user's job always runs first.

Resizable jobs and fairshare

Resizable jobs submitting into fairshare queues or host partitions are subject to fairshare scheduling policies. The dynamic priority of the user who submitted the job is the most important criterion. LSF treats pending resize allocation requests as a regular job and enforces the fairshare user priority policy to schedule them.

The dynamic priority of users depends on:

- Their share assignment
- The slots their jobs are currently consuming
- The resources their jobs consumed in the past
- The adjustment made by the fairshare plugin (libfairshareadjust.*)

Resizable job allocation changes affect the user priority calculation if the RUN_JOB_FACTOR or FAIRSHARE_ADJUSTMENT_FACTOR is greater than zero (0). Resize add requests increase number of slots in use and decrease user priority. Resize release requests decrease number of slots in use, and increase user priority. The faster a resizable job grows, the lower the user priority is, the less likely a pending allocation request can get more slots.

Note:

The effect of resizable job allocation changes when the Fairshare_adjustment_factor is greater than 0 depends on the user-defined fairshare adjustment plugin (libfairshareadjust.*).

After job allocation changes, bqueues and bhpart displays updated user priority.

Resource Preemption

About resource preemption

Preemptive scheduling and resource preemption

Resource preemption is a special type of preemptive scheduling. It is similar to job slot preemption.

Job slot preemption and resource preemption

If you enable preemptive scheduling, job slot preemption is always enabled. Resource preemption is optional. With resource preemption, you can configure preemptive scheduling based on other resources in addition to job slots.

Types of resource preemption

License Preemption	<p>If you have configured a custom resource to manage software application licenses that are shared throughout the cluster (Network Floating Licenses), you can use preemptive scheduling to make these licenses more available to high-priority queues.</p> <p>The license resource can be either static (network floating licenses managed within LSF) or dynamic and decreasing (network floating licenses outside of LSF control and measured with an ELIM).</p>
Other Resources	<p>Resource preemption works for any custom shared numeric resource (except increasing dynamic resources) so its use is not restricted to managing licenses. To preempt on a host-based resource, such as memory, you could configure a custom resource "shared" on only one host.</p>

Multiple resource preemption

If multiple resources are required, LSF can preempt multiple jobs, until sufficient resources are available. For example, one or more jobs might be preempted for a job that needs:

- Multiple job slots
- Multiple licenses for one software application
- Multiple resources, such as a job slot, a license, and memory
- More of a resource than can be obtained by preempting just one job

Use resource preemption

To allow your job to participate in resource preemption, you must use resource reservation to reserve the preemption resource (the cluster might be configured so that this occurs automatically). For dynamic resources, you must specify a duration also.

Resource reservation is part of resource requirement, which can be specified at the job level or at the queue level or application level.

You can use a task file to associate specific resource requirements with specific applications.

Dynamic resources

Specify duration	<p>If the preemption resource is dynamic, you must specify the duration part of the resource reservation string when you submit a preempting or preemptable job.</p>
------------------	--

Resources
outside the
control of LSF

If an ELIM is needed to determine the value of a dynamic resource (such as the number of software licenses available), LSF preempts jobs as necessary, then waits for ELIM to report that the resources are available before starting the high-priority job. By default, LSF waits 300 seconds (5 minutes) for resources to become available. This time can be increased (PREEMPTION_WAIT_TIME in `lsb.params`).

If the preempted jobs do not release the resources, or the resources have been intercepted by a non-LSF user, the ELIM does not report any more of the resource becoming available, and LSF might preempt more jobs to get the resources.

Requirements for resource preemption

- Resource preemption depends on all these conditions:
- The preemption resources must be configured (PREEMPTABLE_RESOURCES in l sb. params).
- Jobs must reserve the correct amount of the preemption resource, using resource reservation (the rusage part of the resource requirement string).
- For dynamic preemption resources, jobs must specify the duration part of the resource reservation string.
- Jobs that use the preemption resource must be spread out among multiple queues of different priority, and preemptive scheduling must be configured so that preemption can occur among these queues (preemption can only occur if jobs are in different queues).
- Only a releaseable resource can be a preemption resource. LSF must be configured to release the preemption resource when the job is suspended (RELEASE=Y in l sf. shared, which is the default). You must configure this no matter what your preemption action is.
- LSF's preemption behavior must be modified. By default, LSF's default preemption action does not allow an application to release any resources, except for job slots and static shared resources.

Custom job controls for resource preemption

Why you have to customize Platform LSF

By default, LSF's preemption action is to send a suspend signal (SIGSTOP) to stop the application. Some applications do not release resources when they get SIGSTOP. If this happens, the preemption resource does not become available, and the preempting job is not successful.

You modify LSF's default preemption behavior to make the application release the preemption resource when a job is preempted.

Customize the SUSPEND action

Ask your application vendor what job control signals or actions cause your application to suspend a job and release the preemption resources. You need to replace the default SUSPEND action (the SIGSTOP signal) with another signal or script that works properly with your application when it suspends the job. For example, your application might be able to catch SIGTSTP instead of SIGSTOP.

By default, LSF sends SIGCONT to resume suspended jobs. You should find out if this causes your application to take the resources back when it resumes the job (for example, if it checks out a license again). If not, you need to modify the RESUME action also.

Whatever changes you make to the SUSPEND job control affects all suspended jobs in the queue, including preempted jobs, jobs that are suspended because of load thresholds, and jobs that you suspend using LSF commands. Similarly, changes made to the RESUME job control also affect the whole queue.

Kill preempted jobs

If you want to use resource preemption, but cannot get your application to release or take back the resource, you can configure LSF to kill the low-priority job instead of suspending it. This method is less efficient because when you kill a job, you lose all the work, and you have to restart the job from the beginning.

- You can configure LSF to kill and requeue suspended jobs (use brequeue as the SUSPEND job control in lsb.queues). This kills all jobs suspended in the queue, not just preempted jobs.
- You can configure LSF to kill preempted jobs instead of suspending them (TERMINATE_WHEN=PREEMPT in lsb.queues). In this case, LSF does not restart the preempted job, you have to resubmit it manually.

Resource preemption steps

To make resource preemption useful, you may need to work through all of these steps.

1. Read.

Before you set up resource preemption, you should understand the following:

- Preemptive Scheduling
- Resource Preemption
- Resource Reservation
- Customizing Resources
- Customizing Job Controls

2. Plan.

When you plan how to set up resource preemption, consider:

- Custom job controls: Find out what signals or actions you can use with your application to control the preemption resource when you suspend and resume jobs.
- Existing cluster configuration: Your design might be based on preemptive queues or custom resources that are already configured in your cluster.
- Requirements for resource preemption: Your design must be able to work. For example, if the application license is the preemption resource, you cannot set up one queue for each type of application, because preemption occurs between different queues. If a host-based resource such as memory is the preemption resource, you cannot set up only one queue for each host, because preemption occurs when 2 jobs are competing for the same resource.

3. Write the ELIM.

4. Configure LSF.

a) 1 sb. queues

- Set PREEMPTION in at least one queue (to PREEMPTIVE in a high-priority queue, or to PREEMPTABLE in a low-priority queue).
- Set JOB_CONTROLS (or TERMINATE_WHEN) in the low-priority queues. Optional. Set RES_REQ to automatically reserve the custom resource.

b) 1 sf. shared

Define the custom resource in the Resource section.

c) 1 sb. params

- Set PREEMPTABLE_RESOURCES and specify the custom resource.
- Optional. Set PREEMPTION_WAIT_TIME to specify how many seconds to wait for dynamic resources to become available.
- Optional. Set PREEMPT_JOBTYPE to enable preemption of exclusive and backfill jobs. Specify one or both of the keywords EXCLUSIVE and BACKFILL. By default, exclusive and backfill jobs are only preempted if the exclusive low priority job is running on a host that is different than the one used by the preemptive high priority job.

d) 1 sf. cluster. *cluster_name*

Define how the custom resource is shared in the ResourceMap section.

e) 1 sf. task. *cluster_name*

Optional. Configure the RemoteTasks section to automatically reserve the custom resource.

5. Reconfigure LSF to make your changes take effect.

6. Operate.

- Use resource reservation to reserve the preemption resource (this might be configured to occur automatically). For dynamic resources, you must specify a duration as well as a quantity.
- Distribute jobs that use the preemption resource in way that allows preemption to occur between queues (this should happen as a result of the cluster design).

7. Track.

Use `bparams -l` to view information about preemption configuration in your cluster.

Configure resource precondition

1. Configure preemptive scheduling (PREEMPTION in l sb. queues).
2. Configure the precondition resources (PREEMPTABLE_RESOURCES in l sb. params).

Job slots are the default precondition resource. To define additional resources to use with preemptive scheduling, set PREEMPTABLE_RESOURCES in lsb.params, and specify the names of the custom resources as a space-separated list.

3. Customize the precondition action.

Preemptive scheduling uses the SUSPEND and RESUME job control actions to suspend and resume preempted jobs. For resource precondition, it is critical that the preempted job releases the resource. You must modify LSF default job controls to make resource precondition work.

- Suspend using a custom job control.

To modify the default suspend action, set JOB_CONTROLS in l sb. queues and use replace the SUSPEND job control with a script or a signal that your application can catch. Do this for all queues where there could be preemptable jobs using the precondition resources.

For example, if your application vendor tells you to use the SIGTSTP signal, set JOB_CONTROLS in l sb. queues and use SIGTSTP as the SUSPEND job control:

```
JOB_CONTROLS = SUSPEND [SIGTSTP]
```

- Kill jobs with brequeue.

To kill and requeue preempted jobs instead of suspending them, set JOB_CONTROLS in l sb. queues and use brequeue as the SUSPEND job control:

```
JOB_CONTROLS = SUSPEND [brequeue $LSB_JOBID]
```

Do this for all queues where there could be preemptable jobs using the precondition resources. This kills a preempted job, and then requeues it so that it has a chance to run and finish successfully.

- Kill jobs with TERMINATE_WHEN.

To kill preempted jobs instead of suspending them, set TERMINATE_WHEN in lsb.queues to PREEMPT. Do this for all queues where there could be preemptable jobs using the precondition resources.

If you do this, the preempted job does not get to run unless you resubmit it.

4. Optional. Configure the precondition wait time.

To specify how long LSF waits for the ELIM to report that the resources are available, set PREEMPTION_WAIT_TIME in l sb. params and specify the number of seconds to wait. You cannot specify any less than the default time (300 seconds).

For example, to make LSF wait for 8 minutes, specify

```
PREEMPTION_WAIT_TIME=480
```


License preemption example

Configuration

This example uses `LicenseA` as name of preemption resource.

lsf.shared

Add the resource to the Resource section.

```
Begin Resource
RESOURCENAME TYPE      INTERVAL INCREASING DESCRIPTION
LicenseA      Numeric 60          N          (custom application)
...
End Resource
```

lsf.cluster.cluster_name

Add the resource to the ResourceMap section

```
Begin ResourceMap
RESOURCENAME LOCATION
LicenseA      [all]
...
End ResourceMap
```

lsb.params

Add the resource to the list of preemption resources.

```
...
PREEMPTABLE_RESOURCES = LicenseA
...
```

lsb.queues

Define a higher priority queue to be a PREEMPTIVE queue by adding one line in the queue definition.

```
Begin Queue
QUEUE_NAME=high
PRIORITY=40
...
PREEMPTION=PREEMPTIVE
DESCRIPTION=jobs may preempt jobs in lower-priority queues
...
End Queue
```

Configure a job control action in a lower priority queue, let `SIGTSTP` be sent when the `SUSPEND` action is called, we assume your application can catch the signal `SIGTSTP` and release the resource (license) it used, then suspend itself. You should also make sure that your application can catch the signal `SIGCONT` while it is suspended, and consume (check out) the resource (license) again.

```
Begin Queue
QUEUE_NAME=low
PRIORITY=20
...
JOB_CONTROLS=SUSPEND[SIGTSTP] RESUME[SIGCONT] TERMINATE[SIGTERM]
DESCRIPTION=jobs preempted by jobs in higher-priority queues
...
End Queue
```

ELIM

Write an ELIM to report the current available number of Application A licenses. This ELIM starts on the master host.

Operation

Check how many LicenseA resources are available

Check the number of LicenseA existing in the cluster by using `bhosts -s LicenseA`. In this example, 2 licenses are available.

```
bhosts -s LicenseA
RESOURCE      TOTAL  RESERVED  LOCATION
LicenseA      2      0.0      hostA hostB ...
```

Use up all LicenseA resources

Submit 2 jobs to a low-priority queue to consume those 2 licenses.

```
bsub -J first -q low -R "rusage[LicenseA=1:duration=2]" your_app
bsub -J second -q low -R "rusage[LicenseA=1:duration=2]" your_app
```

After a while, those jobs are running and the LicenseA resource is used up.

```
bjobs
JOBID USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
201   you   RUN   low    hostx      hostA      /first    Aug 23 15:42
202   you   RUN   low    hostx      hostB      /second   Aug 23 15:43
```

```
bhosts -s LicenseA
RESOURCE      TOTAL  RESERVED  LOCATION
LicenseA      0      2.0      hostA hostB ...
```

Preempt a job for the LicenseA resource

Submit a job to a high-priority queue to preempt a job from a low-priority queue for the resource LicenseA.

```
bsub -J third -q high -R "rusage[LicenseA=1:duration=2]" your_app
```

After a while, the third job is running and the second job is suspended.

```
bjobs
JOBID USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
203   you   RUN   high   hostx      hostA      /third    Aug 23 15:48
201   you   RUN   low    hostx      hostA      /first    Aug 23 15:42
202   you   SSUSP low    hostx      hostB      /second   Aug 23 15:43
```

```
bhosts -s LicenseA
RESOURCE      TOTAL  RESERVED  LOCATION
LicenseA      0      2.0      hostA hostB ...
```

Memory preemption example

Configuration

This example uses `pre_mem` as the name of the preemption resource.

lsf.shared

Add the resource to the Resource section.

```
Begin Resource
RESOURCENAME TYPE      INTERVAL INCREASING DESCRIPTION
pre_mem      Numeric 60      N      (external memory usage reporter)
...
End Resource
```

lsf.cluster.cluster_name

Add the resource to the "ResourceMap" section.

```
Begin ResourceMap
RESOURCENAME LOCATION
pre_mem      ([hostA] [hostB] ... [hostX])
#List the hosts where you want memory preemption to occur.
... End ResourceMap
```

lsb.params

Add the resource to the list of preemption resources.

```
...
PREEMPTABLE_RESOURCES=pre_mem
...
```

lsb.queues

Define a higher-priority queue to be the PREEMPTIVE queue by adding one line in the queue definition.

```
Begin Queue
QUEUE_NAME=high
PRIORITY=40
...
PREEMPTION=PREEMPTIVE
DESCRIPTION=preempt jobs in lower-priority queues
...
End Queue
```

Configure a job control action in a lower-priority queue, and let SIGTSTP be sent when the SUSPEND action is called. This assumes your application can catch the signal SIGTSTP and release (free) the resource (memory) it used, then suspend itself. You should also make sure that your application can catch the signal SIGCONT while it is suspended, and consume the resource (memory) again.

```
Begin Queue
QUEUE_NAME=low
PRIORITY=20
...
JOB_CONTROLS=SUSPEND[SIGTSTP] RESUME[SIGCONT] TERMINATE[SIGTERM]
DESCRIPTION=jobs may be preempted by jobs in higher-priority queues
...
End Queue
```

ELIM

This is an example of an ELIM that reports the current value of `pre_mem`. This ELIM starts on all the hosts that have the `pre_mem` resource.

```
#!/bin/sh
host=`hostname`
```

```

while :
do
lslod > /dev/null 2>&1
if [ $? != 0 ] ; then exit 1
fi
memStr=`lslod -I mem -w $host|grep $host|awk '{print $3}'|sed 's/M//'\`
echo 1 pre_mem $memStr
expecting an integer, exit..."
sleep 60
done

```

Operation

Check how many pre_mem resources are available

Check the number of pre_mem existing on hostA by using `bhosts -s pre_mem` to display how much memory is available. In this example, 110 MB of memory is available on hostA.

```

bhosts -s pre_mem
RESOURCE TOTAL RESERVED LOCATION
pre_mem 110 0.0 hostA
pre_mem 50 0.0 hostB
...

```

Use up some pre_mem resources

Submit 1 job to a low-priority queue to consume 100 MB pre_mem. Assume the application mem_app consumes 100 MB memory after it starts.

```
bsub -J first -q low -R "rusage[pre_mem=100:duration=2]" mem_app
```

After a while, the first job is running and the pre_mem is reduced.

```

bjobs
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
301 you RUN low hostx hostA /first Aug 23 16:42

```

```

bhosts -s pre_mem
RESOURCE TOTAL RESERVED LOCATION
pre_mem 10 100.0 hostA
pre_mem 50 0.0 hostB
...

```

Preempt the job for pre_mem resources

Submit a job to a high-priority queue to preempt a job from low-priority queue to get the resource pre_mem.

```
bsub -J second -q high -R "rusage[pre_mem=100:duration=2]" mem_app
```

After a while, the second job is running and the first job was suspended.

```

bjobs
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
302 you RUN high hostx hostA /second Aug 23 16:48
301 you SSUSP low hostx hostA /first Aug 23 16:42

```

```

bhosts -s pre_mem
RESOURCE TOTAL RESERVED LOCATION
pre_mem 10 100.0 hostA
pre_mem 50 0.0 hostB
...

```

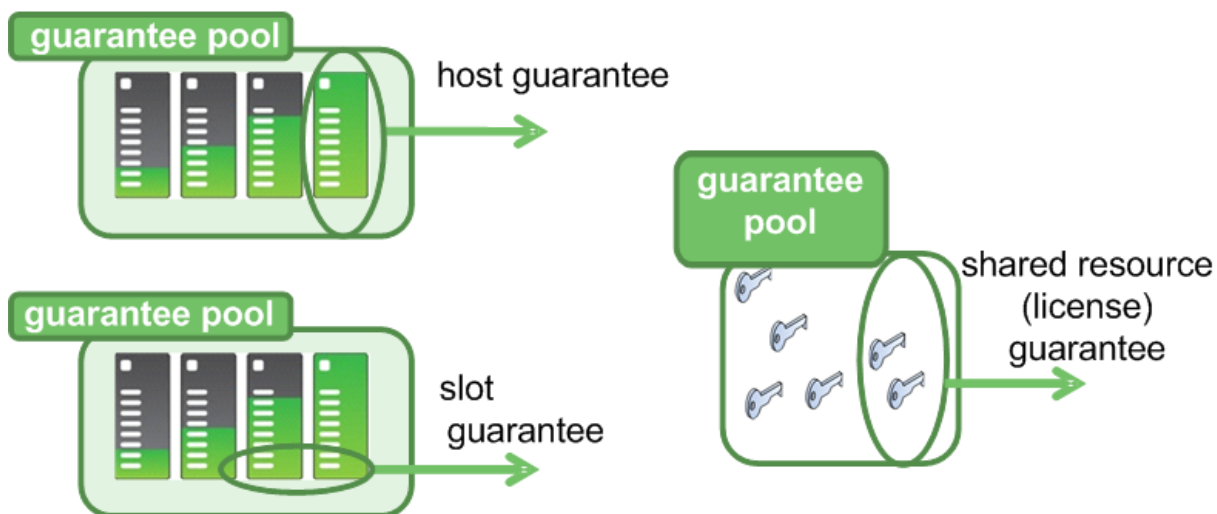
Guaranteed Resource Pools

About guaranteed resource pools

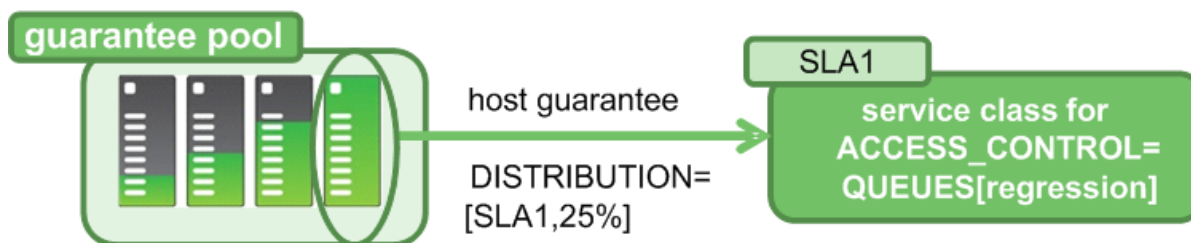
Guaranteed resource pools provide a minimum resource guarantee to consumers, and can optionally loan out guaranteed resources not in use. During job scheduling the order in which jobs are scheduled does not change, but some jobs have access to additional guaranteed resources. Once the guaranteed resources are used, jobs run outside the guarantee following whatever other scheduling features are configured.

Each guaranteed resource pool configured in `lsb. resources` contains resources, guarantee distribution policies, and (optionally) loan policies. Guarantees are made to Service Level Agreements (SLAs) defined in `lsb. serviceclasses`, identifying which consumers are guaranteed the resources. Only guarantee-type SLAs can be used with guaranteed resource pools.

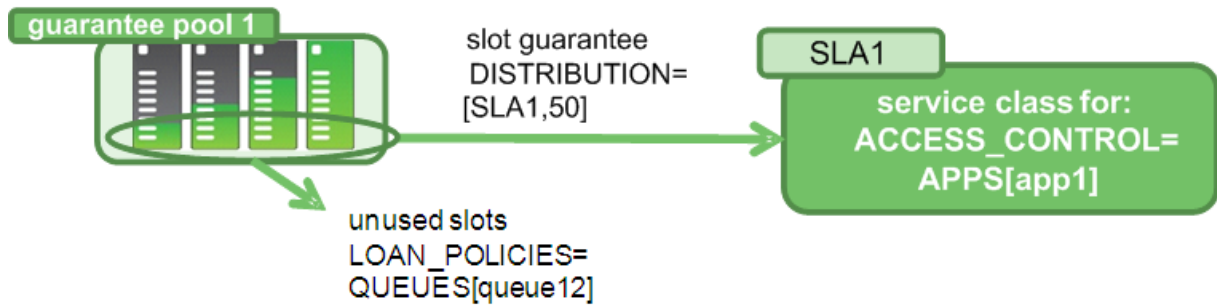
The resources within a guaranteed resource pool can be one of three types: hosts, slots, or licenses defined in Platform License Scheduler. A resource pool can only include one type of resources.



Each guaranteed resource pool defines specific guarantees in the `DISTRIBUTION` parameter, linking the resources to SLAs. Guarantees can be made for a percent of the pool, or for a fixed number of resources within the pool.



Optionally, loans can be configured for the entire resource pool. When loans are allowed, unused guaranteed resources within the pool can be loaned out when not in use. Different loan policies can be set, depending on which queues you want to borrow the resources and how quickly you need guaranteed resources to be available to consumers.



For more information about configuring guarantee-type SLAs, see [Resource-based service classes](#) on page 516

Guarantee

A guarantee provides jobs belonging to set consumers with specific resources (such as hosts). Jobs run using guaranteed resources when possible. Once the guaranteed resources are used, jobs run outside the guarantee following whatever other scheduling features are configured. Guarantees are configured within a guaranteed resource pool.

Guaranteed resource pool

A collection of resources and the guaranteed distribution of these resources to SLAs. Loan policies for unused guaranteed resources can optionally be included. Guaranteed resource pools are configured in the `lsb. resources` file in `GuaranteedResourcePool` sections, with guarantees made to one or more of the SLAs configured in the `lsb. serviceclasses` file.

Meeting a guarantee

A guarantee is met when all resources included in the guarantee have been provided to jobs running in the SLA configured to receive the guarantee. Jobs submitted to the SLA compete with other jobs for resources once the guarantee is met.

Consumers

Resources are guaranteed to consumers who can access the SLA, as set by `ACCESS_CONTROL` in the `ServiceClass` section in `lsb. serviceclasses`. Consumers can include users, user groups, queues, fairshare groups, application profiles, projects, or license projects.

Loans

When the resources allocated to a guarantee within a guaranteed resource pool are not all used, non-SLA jobs and jobs from other SLAs can run on these resources. To enable loans, loan policies must be configured for the guaranteed resource pool.

Loaned resources

Loaned resources are resources guaranteed to an SLA, but currently in use by jobs outside of the SLA.

Guaranteed resource pool considerations

A host-type guaranteed resource pool does not guarantee a specific host will always be reserved for a consumer. Instead, the guarantee is for any host in the pool. Similarly a slot-type pool guarantees slots on any of the hosts in the pool.

A host-type or slot-type guaranteed resource pool includes hosts in the following states:

- `ok`
- `closed_Busy`
- `closed_Excl`
- `closed_cu_Excl`
- `closed_Full`

Hosts in other states are temporarily excluded from the pool, and any SLA jobs running on hosts in other states are not counted towards the guarantee.

Advance reservation

Hosts within an advance reservation are excluded from guarantee resource pools.

Compute units

Configuring guaranteed resource pools and compute units with hosts in common is not recommended. If such configuration is required, do not submit jobs with compute unit requirements using the `maxcus`, `balance`, or `excl` keywords.

Queue-based fairshare

During loan scheduling, shares between queues are not preserved. If `SLOT_POOL` is defined in `lsb.queues` both the fairshare and guarantee limits apply.

Exclusive jobs

Using exclusive jobs with slot-type guaranteed resource pools is not recommended. Instead use host-type pools.

MultiCluster

Leased hosts can be used in a guaranteed resource pool by including a host group with remote hosts in the `HOSTS` parameter.

Preemption

Guarantee SLA jobs can only be preempted by queues with `SLA_GUARANTEES_IGNORE=Y`. Such queues cannot be preemptable.

Jobs scheduled using loaned resources cannot trigger preemption.

Guarantee SLA jobs can preempt other jobs, and can use preemption to meet guarantees.

Chunk jobs

Jobs running on loaned resources cannot be chunked.

Forced jobs (`brun`)

Jobs forced to run using `brun` can use resources regardless of guarantees.

Configuration overview of guaranteed resource pools

Guaranteed resource pools are defined in `lsb. resources` and used by consumers defined within SLAs in `lsb. serviceclasses`.

Note:

Hosts can only be included in one guaranteed resource pool.

1. Choose the type of resource pool to configure. The three possible types are hosts, slots, or resources (such as licenses).
2. Configure a `GuaranteedResourcePool` section in `lsb. resources`. Begin with the line `Begin GuaranteedResourcePool` and end with the line `End GuaranteedResourcePool`. Specify the following:
 - a) `NAME`; the name of the guaranteed resource pool.
 - b) `TYPE`; the guarantee type (slots, hosts, or resources).
 - c) `DISTRIBUTION`; share assignments for all SLAs using the resource pool. Can be percent or absolute numbers.
 - d) Optional parameters for the `GuaranteedResourcePool` section are `HOSTS`, `RES_SELECT`, `LOAN_POLICIES`, `DESCRIPTION`, and `SLOTS_PER_HOST`.

You can configure as many resource pools as you need. One resource pool can be used by several SLAs, and one SLA can access multiple resource pools.

For example:

```
Begin GuaranteedResourcePool
NAME = GRP1A
TYPE = hosts
HOSTS = host1 host2 hgb
DISTRIBUTION = [sla1, 3] [sla2, 25%] [sla3, 3]
DESCRIPTION = A resource pool used by sla1, sla2, and sla3.
End GuaranteedResourcePool
```

3. Configure a `ServiceClass` section in `lsb. serviceclasses` for each SLA. Begin with the line `Begin ServiceClass` and end with the line `End ServiceClass`. For each SLA you must specify:
 - a) `NAME`; the name of the SLA.
 - b) `GOALS = [GUARANTEE]`
 - c) Optional parameters for the `ServiceClass` section are `ACCESS_CONTROL`, `AUTO_ATTACH`, and `DESCRIPTION`.

You can configure as many service class sections as you need.

Important:

The name you use for your SLA cannot be the same as an existing host partition or user group name.

For example:

```
Begin ServiceClass
NAME = sla1
GOALS = [GUARANTEE]
ACCESS_CONTROL=USERS[ user1 user2 user3 ]
AUTO_ATTACH=Y
DESCRIPTION = A guarantee SLA with access restricted to specific users. Jobs from
these users will be attached to the SLA automatically and run on guaranteed
resources if possible.
End ServiceClass
```

Configure SLOTS_PER_HOST in a slot pool

Guaranteed resource pools with `TYPE=slots` can be used to spread jobs over hosts within a cluster by limiting the number of slots available on each host. This is useful for jobs requiring large amounts of memory.

`SLOTS_PER_HOST` limits the number of slots available on each host in the pool for all SLA jobs with a guarantee in the pool, even when jobs are running outside of the guarantee.

1. Configure a guaranteed resource pool in `lsb. resources` with `TYPE = slots` and `SLOTS_PER_HOST` defined.

```
Begin GuaranteedResourcePool
...
TYPE = slots
SLOTS_PER_HOST = 1
...
End GuaranteedResourcePool
```

`SLOTS_PER_HOST` limits the number of slots each host contributes to a slot-type resource pool.

Configure loans to specific queues

Loans allow queues to access unused resources from guaranteed resource pools.

1. Configure a guaranteed resource pool in `lsb. resources` with the required `NAME`, `TYPE`, and `DISTRIBUTION` parameters.
2. Add a loan policy to the guaranteed resource pool.

`LOAN_POLICIES= QUEUES[queue_name]` allows you to specify which queues can access loaned resources. Use the keyword `all` to loan to jobs from any queue.

For example, to allow loans to jobs from the queue `my_queue`:

```
Begin GuaranteedResourcePool
...
LOAN_POLICIES = QUEUES[my_queue]
...
End GuaranteedResourcePool
```

Configure loans to short jobs

Loans can allow queues to access unused resources from guaranteed resource pools.

1. Configure a guaranteed resource pool in `lsb. resources` with the required `NAME`, `TYPE`, and `DISTRIBUTION` parameters and a loan policy such as `QUEUES[queue1]`.
2. Add the policy `DURATION[minutes]` to the guaranteed resource pool configuration, where minutes is an integer.

`DURATION` allows you to set a maximum job runtime limit (or estimated runtime, whichever is shorter) for jobs to borrow resources. Omit `DURATION` completely to allow jobs with any runtime to borrow from the guarantee.

For example, to allow loans to jobs from any queue with a runtime of 10 minutes or less:

```
Begin GuaranteedResourcePool
...
LOAN_POLICIES = QUEUES[all] DURATION[10]
...
End GuaranteedResourcePool
```

Configure loans to stop when jobs are waiting for guaranteed resources

Loans can be restricted so that jobs have access to the loaned resources only when consumers with unused guaranteed resources do not have pending loads.

Restricting loans is useful when running parallel jobs that require several slots, or jobs that require large blocks of memory. With restricted loans enabled, small borrowing jobs will not delay jobs waiting for resources such as slots or memory to accumulate.

1. Configure a guaranteed resource pool in `l sb. resources` with a loan policy such as `QUEUES [queue1]`.
2. Add the policy `CLOSE_ON_DEMAND` to the guaranteed resource pool configuration. For example:

```
Begin GuaranteedResourcePool
...
LOAN_POLICIES = QUEUES[queue1] CLOSE_ON_DEMAND
...
End GuaranteedResourcePool
```

Configure a queue with access to all guaranteed resources

Queues with very high priority (such as administrator test queues) can be configured with access to all guaranteed resources, regardless of SLA demand.

1. Configure a queue in `l sb. queues` with `SLA_GUARANTEES_IGNORE = Y`.

Note:

Using `SLA_GUARANTEES_IGNORE=Y` defeats the purpose of guaranteeing resources. This should be used sparingly for low traffic queues only.

Configure preemptable loans

Guarantee SLA jobs can use preemption to reclaim loaned resources when preemption is configured.

Configuring preemption with guarantees returns loaned resources quickly even when loans are made to jobs of any length (without specifying maximum runtime using `DURATION`).

1. Configure a guaranteed resource pool in `l sb. resources` with a loan policy. For example:

```
Begin GuaranteedResourcePool
NAME = my_pool
TYPE = hosts
HOSTS = host1 host2 hgB
DISTRIBUTION = [sla1, 50%]
LOAN_POLICIES = QUEUES[loanQ]
DESCRIPTION = A resource pool used by sla1.
End GuaranteedResourcePool
```

2. Configure a ServiceClass section in `l sb. serviceclasses` for each SLA. For example:

```
Begin ServiceClass
NAME = SLA1
GOALS = [GUARANTEE]
AUTO_ATTACH = Y
ACCESS_CONTROL = QUEUES[SLA1_queue]
DESCRIPTION = Service class SLA1 uses GRP my_pool.
End ServiceClass
```

In this example, unused resources from the guaranteed resource pool are loaned to jobs of any length submitted to `loanQ`. All jobs submitted to `SLA1_queue` are automatically attached to `SLA1`, and are guaranteed 50% of the resources in `my_pool`.

3. Configure the queue using loaned resources in l sb. queues as preemptable and low priority. For example:

```
Begin Queue
QUEUE_NAME=loanQ
PREEMPTION=PREEMPTABLE [SLA1_queue]
PRIORITY=1
End Queue
```

Jobs submitted to the queue loanQ run on unused hosts in the guarantee for SLA1.

4. Configure the SLA job queue in l sb. queues as preemptive and higher priority:

```
Begin Queue
QUEUE_NAME=SLA1_queue
PREEMPTION=PREEMPTIVE [loanQ]
Priority=10
End Queue
```

Jobs submitted to SLA1 through the queue SLA1_queue can preempt jobs from loanQ in order to reclaim the guaranteed resources as quickly as possible.

View guaranteed resource pools

Guaranteed resource pool configuration includes the resource type, and distribution among consumers defined in the corresponding SLAs.

1. Run `bresources -g -l -m` to see details of the guaranteed resource pool configuration, including a list of hosts currently in the resource pool. For example:

```
> bresources -g -l -m
GUARANTEED RESOURCE POOL: Pool 1A

TYPE: slots
DISTRIBUTION: [sla1, 3] [sla2, 25%] [sla3, 3]
HOSTS: hg12

STATUS: ok

RESOURCE SUMMARY:
TOTAL          36
FREE           22

GUARANTEE CONFIGURED      15
GUARANTEE UNUSED          7

CONSUMERS      GUARANTEED  USED
sla1           3           2
sla2           9           1
sla3           3           5

Hosts currently in the resource pool:
host1 host2 host3 host5 host6 host7 host8 host10 host11
```

Goal-Oriented SLA-Driven Scheduling

Goal-oriented SLA scheduling

Goal-oriented SLA scheduling policies help you configure both your workload so jobs are completed on time, and your resource distribution so users get the resources they deserve. They enable you to focus on the “what and when” of your projects, not the low-level details of “how”.

Service-level agreements in LSF

A *service-level agreement* (SLA) defines a service and the parameters for delivery of the service. It specifies what a service provider and a service recipient agree to, defining the relationship between the provider and recipient with respect to a number of issues, among them:

- Services to be delivered
- Performance
- Tracking and reporting
- Problem management
- Resources guaranteed

The SLA scheduling policy defines how many jobs should be run from or resources allocated to each SLA in order to meet the configured goals.

Service class goals

SLAs use goals that are expressed in individual *service classes*. A service class contains the actual configured goals for the LSF system. The SLA defines the workload (jobs, services, or resources) and users that need the work done, while the service class that addresses the SLA defines individual goals, and if applicable a time window when the service class is active.

Service-level goals can be grouped into two mutually exclusive varieties: guarantee goals which are resource based, and time-based goals which include velocity, throughput, and deadline goals. Time-based goals allow control over the number of jobs running at any one time, while resource-based goals allow control over resource allocation.

You configure the following kinds of goals:

Guarantee goals

Specific resources reserved under a guarantee for jobs within the SLA. For example, reserve 50% of a host group for use by a certain queue. Guarantee goals are resource-based.

Deadline goals

A specified number of jobs should be completed within a specified time window. For example, run all jobs submitted over a weekend. Deadline goals are time-based.

Velocity goals

Expressed as concurrently running jobs. For example: maintain 10 running jobs between 9:00 a.m. and 5:00 p.m. Velocity goals are well suited for short jobs (run time less than one hour). Such jobs leave the system quickly, and configuring a velocity goal ensures a steady flow of jobs through the system. Velocity goals are time-based.

Throughput goals

Expressed as number of finished jobs per hour. For example: finish 15 jobs per hour between the hours of 6:00 p.m. and 7:00 a.m. Throughput goals are suitable for medium

to long running jobs. These jobs stay longer in the system, so you typically want to control their rate of completion rather than their flow. Throughput goals are time-based.

Combined goals

Time-based goals can be combined and apply during different time windows. You might want to set velocity goals to maximize quick work during the day, and set deadline and throughput goals to manage longer running work on nights and over weekends. Resource-based guarantee goals cannot be combined with other types of goals.

Resource-based service classes

Resource-based service classes control workload by guaranteeing specific resources to jobs within an SLA, ensuring jobs have priority within a dedicated share of resources. Once guaranteed resources are used up, SLA jobs are still able to run outside the guaranteed resource pool. A guaranteed resource pool can be based on hosts, slots, or shared resources such as licenses.

Job scheduling

Resource-based service class scheduling for guarantee goals differs from time-based service class scheduling; the order in which jobs are scheduled does not change. Jobs are considered for dispatch according to whatever other scheduling features are configured, but have access to additional guaranteed resources. LSF tries to place jobs within a guarantee if possible.

The scheduler keeps track of how many resources are in each guaranteed resource pool, how many are reserved for each guarantee, and how many can be shared. Jobs belonging to guarantee SLAs are permitted to use resources from the resource pool until guarantees are met. Once guarantees are met, SLA jobs start consuming resources outside of the guarantee.

Loans

Loaning of guaranteed resources gives jobs outside of an SLA limited access to resources not in use by guarantee SLA jobs. By default LSF reserves sufficient resources in each guaranteed resource pool to honor all guarantees. Different configuration options allow loans to specific queues, short jobs, or jobs of any length.

Any guaranteed resources remaining idle at the end of a scheduling session may be loaned to jobs if loaning is enabled in the guaranteed resource pool (l sb. resources).

Resource-based SLA considerations

MultiCluster

MultiCluster jobs under the job-forwarding model may be automatically attached to an SLA in the remote cluster using `AUTO_ATTACH`.

Preemption

Guarantee SLA jobs can preempt other jobs and use preemption to meet a guarantee. Guarantee SLA jobs can only be preempted by queues with `SLA_GUARANTEES_IGNORE=Y`.

EGO-enabled SLAs

Guarantee SLAs cannot be used with EGO-enabled SLAs (`ENABLE_DEFAULT_EGO_SLA` in l sb. params).

Configure resource-based service classes

Resource-based guarantee service classes require a resource pool in l sb. resources and a service class in l sb. serviceclasses.

Note:

Hosts can only be included in one guaranteed resource pool.

1. Configure a `GuaranteedResourcePool` section in `lsb.resources`. Begin with the line `Begin GuaranteedResourcePool` and end with the line `End GuaranteedResourcePool`.

You can configure as many resource pools as you need. One resource pool can be used by several SLAs, and one SLA can access multiple resource pools.

For example:

```
Begin GuaranteedResourcePool
NAME = GRP1A
TYPE = hosts
HOSTS = host1 host2 hgB
DISTRIBUTION = [sla1, 3] [sla2, 25%] [sla3, 3]
DESCRIPTION = A resource pool used by sla1, sla2, and sla3.
End GuaranteedResourcePool
```

2. Configure a `ServiceClass` section in `lsb.serviceclasses`. Begin with the line `Begin ServiceClass` and end with the line `End ServiceClass`.

You can configure as many service class sections as you need.

Important:

The name you use for your service class cannot be the same as an existing host partition or user group name.

For example:

```
Begin ServiceClass
NAME = sla1
GOALS = [GUARANTEE]
ACCESS_CONTROL=USERS[ james tony jessica ]
AUTO_ATTACH=Y
DESCRIPTION = A guarantee SLA with access restricted to specific users. Jobs from
these users will be attached to the SLA automatically and run on guaranteed
resources if possible.
End ServiceClass
```

Resource-based SLA examples

A host-type guarantee SLA

Hosts owned by specific departments or projects can be guaranteed to users simply and easily using SLAs. Guarantee SLAs allow you to configure guaranteed resources while ensuring unused resources are accessible to other users, within reason. This is achieved through loans to short jobs; The longest pending guarantee SLA jobs wait for the guaranteed resources to become available is the configured loan duration policy.

lsb.resources configuration:

```
Begin GuaranteedResourcePool
NAME = Proj2Pool
TYPE = hosts
DISTRIBUTION = [productsSLA, 30%] [accountingSLA, 20]
LOAN_POLICIES = QUEUES[shortJobs] DURATION[ 10]
End GuaranteedResourcePool
```

lsb.serviceclasses configuration:

```
Begin ServiceClass
NAME = productsSLA
GOALS = [GUARANTEE]
ACCESS_CONTROL = FAIRSHARE_GROUPS[products]
AUTO_ATTACH = Y
End ServiceClass
```

```

Begin ServiceClass
NAME = accountingSLA
GOALS = [GUARANTEE]
ACCESS_CONTROL = USERS[accountingUG]
AUTO_ATTACH = Y
End ServiceClass

```

bresources -g output shows the configured guaranteed resource pool:

> **bresources -g**

POOL_NAME	TYPE	STATUS	TOTAL	FREE	GUAR CONFIG	GUAR USED
productsGuarantee	hosts	ok	50	50	35	0

bsl a output shows the configured SLAs:

> **bsla**

```

SERVICE CLASS NAME: productsSLA
ACCESS CONTROL: FAIRSHARE_GROUPS[products/]
AUTO ATTACH: Y

```

GOAL: GUARANTEE

POOL NAME	TYPE	GUAR CONFIG	GUAR USED	TOTAL USED
Proj2Pool	hosts	15	0	0

```

SERVICE CLASS NAME: accountingSLA
ACCESS CONTROL: USERS[accountingUG/]
AUTO ATTACH: Y

```

GOAL: GUARANTEE

POOL NAME	TYPE	GUAR CONFIG	GUAR USED	TOTAL USED
Proj2Pool	hosts	20	0	0

Jobs submitted to fairshare queues by users in the fairshare group `products` are auto-attached to the guarantee SLA `productsSLA`, and allocated up to 30% of hosts in the resource pool. Jobs submitted by users in the `accountingUG` usergroup are auto-attached to the guarantee SLA `accountingSLA`, and allocated up to 20 hosts. Since a list of hosts or host groups is not included in the guaranteed resource pool configuration, all available hosts are included in the resource pool.

Once each guarantee is met, jobs can run outside the guarantee based on the overall job priority. Unused resources can be borrowed by jobs from queue `shortJobs` with runtimes (or estimated runtimes) of 10 minutes or less.

In this example loans could be enabled for all queues; restricting loans to a single queue containing only short jobs may improve scheduling performance.

A slot-type guarantee SLA

In some cases fairshare distributions consider only slot usage, so slot-type guarantee SLAs can be used to guarantee resources.

lsb.resources configuration:

```

Begin GuaranteedResourcePool
NAME = linuxPool
TYPE = slots
HOSTS = linuxHG
DISTRIBUTION = [engSLA, 30%] [devSLA, 35%]
LOAN_POLICIES = QUEUES[all] CLOSE_ON_DEMAND
End GuaranteedResourcePool

Begin GuaranteedResourcePool
NAME = solarisPool
TYPE = slots
HOSTS = linuxHG

```

```

DISTRIBUTION = [engSLA, 50%] [devSLA, 20%]
LOAN_POLICIES = QUEUES[all] CLOSE_ON_DEMAND
End GuaranteedResourcePool

```

lsb.serviceclasses configuration:

```

BeginServiceClass
NAME = engSLA
GOALS = [GUARANTEE]
ACCESS_CONTROL = FAIRSHARE_GROUPS[eng]
AUTO_ATTACH = Y
End ServiceClass

BeginServiceClass
NAME = devSLA
GOALS = [GUARANTEE]
ACCESS_CONTROL = FAIRSHARE_GROUPS[dev]
AUTO_ATTACH = Y
End ServiceClass

```

bresources -g output shows the configured guaranteed resource pool:

> **bresources -g**

POOL_NAME	TYPE	STATUS	TOTAL	FREE	GUAR CONFIG	GUAR USED
linuxPool	slots	ok	100	90	65	65
solarisPool	slots	ok	100	100	70	70

bsla output shows the configured SLAs:

> **bsla**

```

SERVICE CLASS NAME: engSLA
AUTO ATTACH: Y
ACCESS CONTROL: FAIRSHARE_GROUPS[eng/]

```

GOAL: GUARANTEE

POOL NAME	TYPE	GUAR CONFIG	GUAR USED	TOTAL USED
linuxPool	slots	30	0	0
solarisPool	slots	50	0	0

```

SERVICE CLASS NAME: devSLA
AUTO ATTACH: Y
ACCESS CONTROL: FAIRSHARE_GROUPS[dev/]

```

GOAL: GUARANTEE

POOL NAME	TYPE	GUAR CONFIG	GUAR USED	TOTAL USED
linuxPool	slots	35	0	0
solarisPool	slots	20	0	0

Jobs submitted by users in fairshare groups eng and dev are auto-attached to guarantee SLAs. Each guarantee SLA has a share in both the linuxPool resource pool and the solarisPool resource pool. Once the guarantees are met, additional SLA jobs can run on slots not reserved for guarantees.

Unused resources can be borrowed by jobs of any length from any queue, so long as there is no pending load on the guarantees.

Large memory jobs in a guarantee SLA

Clusters running large memory jobs can use guaranteed resource pools to limit the number of large memory jobs running on each host. By forming a slot-type resource pool limited to one slot on each host, guarantee SLA jobs using the resource pool will run on a single slot per host, spreading out memory consumption and improving performance.

lsb.resources configuration:

```

Begin GuaranteedResourcePool

```

```

NAME = bi gMemPool
TYPE = slots
HOSTS = bi gMemHG
SLOTS_PER_HOST = 1
DISTRIBUTION = [bi gMemSLA, 50%]
LOAN_POLICIES = QUEUES[all]
End GuaranteedResourcePool

```

lsb.serviceclasses configuration:

```

BeginServiceClass
NAME = bi gMemSLA
GOALS = [GUARANTEE]
ACCESS_CONTROL = QUEUES[bi gMem]
AUTO_ATTACH = Y
End ServiceClass

```

bresources -g output shows the configured guaranteed resource pool:

> **bresources -g**

POOL_NAME	TYPE	STATUS	TOTAL	FREE	GUAR CONFIG	GUAR USED
bi gMemPool	slots	ok	100	90	50	50

bsl a output shows the configured SLAs:

> **bsla**

```

SERVICE CLASS NAME: devSLA
ACCESS CONTROL: QUEUES[bi gMem]
AUTO ATTACH: Y

```

GOAL: GUARANTEE

POOL NAME	TYPE	GUAR CONFIG	GUAR USED	TOTAL USED
bi gMemPool	slots	50	0	0

Jobs submitted to queue bi gMem are auto-attached to the guarantee SLA bi gMemSLA, and allocated a single slot on one of the hosts in the hostgroup bi gMemHG (as configured in the guaranteed resource pool bi gMemPool). Unused slots can be borrowed by jobs of any length from any queue.

Time-based service classes

Time-based service classes configure workload based on the number of jobs running at any one time. Goals for deadline, throughput, and velocity of jobs ensure that your jobs are completed on time and reduce the risk of missed deadlines.

Time-based SLA scheduling makes use of other, lower level LSF policies like queues and host partitions to satisfy the service-level goal that the service class expresses. The decisions of a time-based service class are considered first before any queue or host partition decisions. Limits are still enforced with respect to lower level scheduling objects like queues, hosts, and users.

Optimum number of running jobs

As jobs are submitted, LSF determines the optimum number of job slots (or concurrently running jobs) needed for the time-based service class to meet its goals. LSF schedules a number of jobs at least equal to the optimum number of slots calculated for the service class.

LSF attempts to meet time-based goals in the most efficient way, using the optimum number of job slots so that other service classes or other types of work in the cluster can still progress. For example, in a time-based service class that defines a deadline goal, LSF spreads out the work over the entire time window for the goal, which avoids blocking other work by not allocating as many slots as possible at the beginning to finish earlier than the deadline.

You should submit time-based SLA jobs with a run time limit at the job level (- Woption), the application level (RUNLIMIT parameter in the application definition in `lsb. appl i cat i ons`), or the queue level (RUNLIMIT parameter in the queue definition in `lsb. queues`). You can also submit the job with a run time estimate defined at the application level (RUNTIME parameter in `lsb. appl i cat i ons`) instead of or in conjunction with the run time limit.

The following table describes how LSF uses the values that you provide for time-based SLA scheduling.

If you specify...	And...	Then...
A run time limit and a run time estimate	The run time estimate is less than or equal to the run time limit	LSF uses the run time estimate to compute the optimum number of running jobs.
A run time limit	You do not specify a run time estimate, or the estimate is greater than the limit	LSF uses the run time limit to compute the optimum number of running jobs.
A run time estimate	You do not specify a run time limit	LSF uses the run time estimate to compute the optimum number of running jobs.
Neither a run time limit nor a run time estimate		LSF automatically adjusts the optimum number of running jobs according to the observed run time of finished jobs.

Time-based service class priority

A higher value indicates a higher priority, relative to other time-based service classes. Similar to queue priority, time-based service classes access the cluster resources in priority order.

LSF schedules jobs from one time-based service class at a time, starting with the highest-priority service class. If multiple time-based service classes have the same priority, LSF runs the jobs from these service classes in the order the service classes are configured in `lsb. servi cecl asses`.

Time-based service class priority in LSF is completely independent of the UNIX scheduler's priority system for time-sharing processes. In LSF, the NICE parameter is used to set the UNIX time-sharing priority for batch jobs.

User groups for time-based service classes

You can control access to time-based SLAs by configuring a user group for the service class. If LSF user groups are specified in `lsb.users`, each user in the group can submit jobs to this service class. If a group contains a subgroup, the service class policy applies to each member in the subgroup recursively. The group can define fairshare among its members, and the SLA defined by the service class enforces the fairshare policy among the users in the user group configured for the SLA.

By default, all users in the cluster can submit jobs to the service class.

Time-based SLA limitations

MultiCluster

Platform MultiCluster does not support time-based SLAs.

Preemption

Time-based SLA jobs cannot be preempted. You should avoid running jobs belonging to an SLA in low priority queues.

Chunk jobs

SLA jobs will not get chunked. You should avoid submitting SLA jobs to a chunk job queue.

Resizable jobs

For resizable job allocation requests, since the job itself has already started to run, LSF bypasses dispatch rate checking and continues scheduling the allocation request.

Time-based SLA statistics files

Each time-based SLA goal generates a statistics file for monitoring and analyzing the system. When the goal becomes inactive the file is no longer updated. Files are created in the `LSB_SHAREDIR/cluster_name/logdir/SLA` directory. Each file name consists of the name of the service class and the goal type.

For example the file named `Quadra.deadline` is created for the deadline goal of the service class name `Quadra`. The following file named `Tofino.velocity` refers to a velocity goal of the service class named `Tofino`:

cat Tofino.velocity

```
# service class Tofino velocity, NJOBS, NPEND (NRUN + NSSUSP + NUSUSP), (NDONE + NEXIT)
17/9      15: 7: 34      1063782454 2 0 0 0 0
17/9      15: 8: 34      1063782514 2 0 0 0 0
17/9      15: 9: 34      1063782574 2 0 0 0 0
# service class Tofino velocity, NJOBS, NPEND (NRUN + NSSUSP + NUSUSP), (NDONE + NEXIT)
17/9      15: 10: 10     1063782610 2 0 0 0 0
```

Configure time-based service classes

Configure time-based service classes in `LSB_CONFDIR/cluster_name/configdir/lsb.servicelasses`.

1. Each ServiceClass section begins with the line `Begin ServiceClass` and ends with the line `End ServiceClass`. For time-based service classes you must specify:
 - a) A service class name
 - b) At least one goal (deadline, throughput, or velocity) and a time window when the goal is active
 - c) A service class priority

Other parameters are optional. You can configure as many service class sections as you need.

Important:

The name you use for your service class cannot be the same as an existing host partition or user group name.

Time-based configuration examples

- The service class `Sooke` defines one deadline goal that is active during working hours between 8:30 AM and 4:00 PM. All jobs in the service class should complete by the end of the specified time window. Outside of this time window, the SLA is inactive and jobs are scheduled without any goal being enforced:

```
Begin ServiceClass
NAME = Sooke
PRIORITY = 20
GOALS = [DEADLINE timeWindow (8:30-16:00)]
DESCRIPTION="working hours"
End ServiceClass
```

- The service class `Nanaimo` defines a deadline goal that is active during the weekends and at nights.

```
Begin ServiceClass
NAME = Nanaimo
PRIORITY = 20
GOALS = [DEADLINE timeWindow (5:18:00-1:8:30 20:00-8:30)]
DESCRIPTION="weekend nighttime regression tests"
End ServiceClass
```

- The service class `Sidney` defines a throughput goal of 6 jobs per hour that is always active:

```
Begin ServiceClass
NAME = Sidney
PRIORITY = 20
GOALS = [THROUGHPUT 6 timeWindow ()]
DESCRIPTION="constant throughput"
End ServiceClass
```

Tip:

To configure a time window that is always open, use the `timeWindow` keyword with empty parentheses.

- The service class `Tofino` defines two velocity goals in a 24 hour period. The first goal is to have a maximum of 10 concurrently running jobs during business hours (9:00 a.m. to 5:00 p.m.). The second goal is a maximum of 30 concurrently running jobs during off-hours (5:30 p.m. to 8:30 a.m.)

```
Begin ServiceClass
NAME = Tofino
PRIORITY = 20
GOALS = [VELOCITY 10 timeWindow (9:00-17:00)] \
        [VELOCITY 30 timeWindow (17:30-8:30)]
DESCRIPTION="day and night velocity"
End ServiceClass
```

- The service class Duncan defines a velocity goal that is active during working hours (9:00 a.m. to 5:30 p.m.) and a deadline goal that is active during off-hours (5:30 p.m. to 9:00 a.m.) Only users user1 and user2 can submit jobs to this service class.

```
Begin ServiceClass
NAME = Duncan
PRIORITY = 23
USER_GROUP = user1 user2
GOALS = [VELOCITY 8 timeWindow (9:00-17:30)] \
        [DEADLINE timeWindow (17:30-9:00)]
DESCRIPTION="Daytime/Nighttime SLA"
End ServiceClass
```

- The service class Tevere defines a combination similar to Duncan, but with a deadline goal that takes effect overnight and on weekends. During the working hours in weekdays the velocity goal favors a mix of short and medium jobs.

```
Begin ServiceClass
NAME = Tevere
PRIORITY = 20
GOALS = [VELOCITY 100 timeWindow (9:00-17:00)] \
        [DEADLINE timeWindow (17:30-8:30 5:17:30-1:8:30)]
DESCRIPTION="nine to five" End ServiceClass
```

Time-based SLA examples

A simple deadline goal

The following service class configures an SLA with a simple deadline goal with a half hour time window.

```
Begin ServiceClass
NAME = Quadra
PRIORITY = 20
GOALS = [DEADLINE timeWindow (16:15-16:45)]
DESCRIPTION = short window
End ServiceClass
```

Six jobs submitted with a run time of 5 minutes each will use 1 slot for the half hour time window. bsl a shows that the deadline can be met:

bsla Quadra

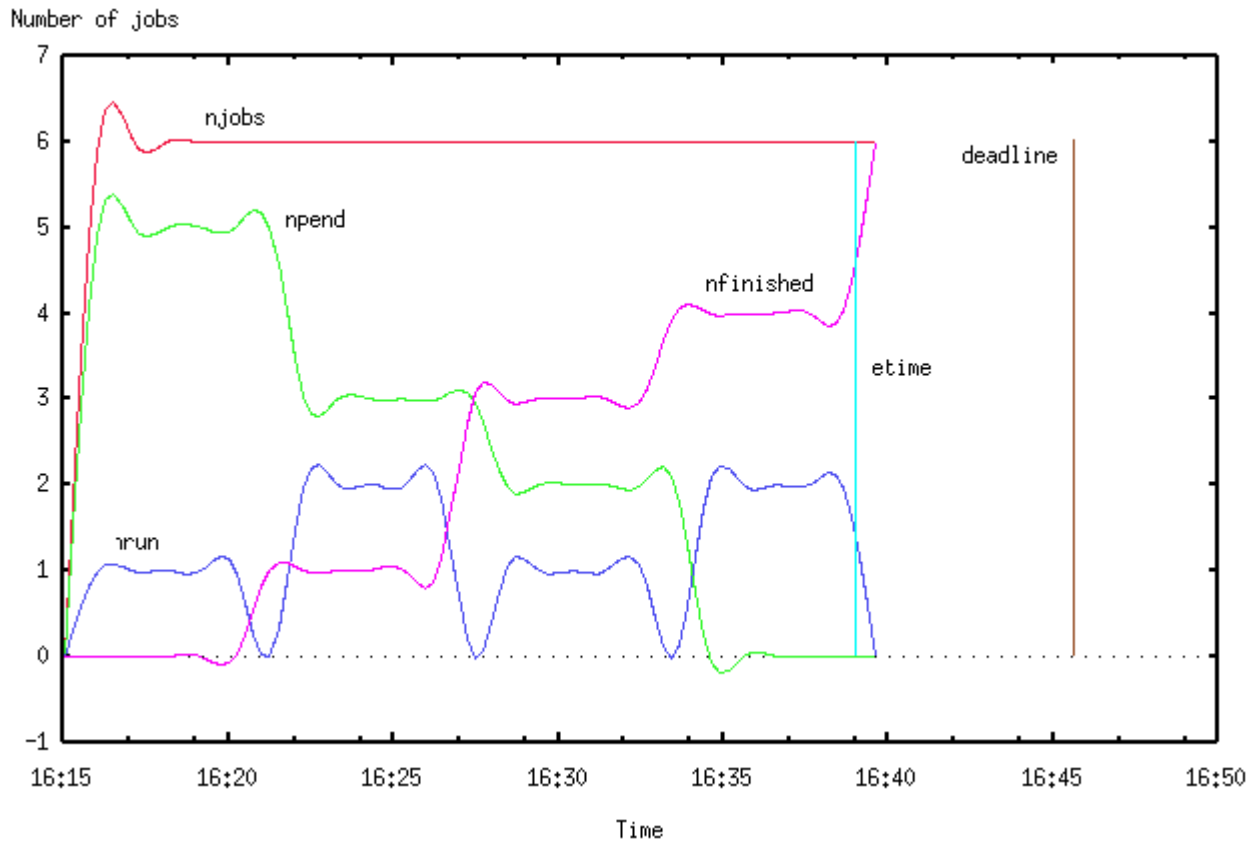
```
SERVICE CLASS NAME: Quadra
-- short window
PRIORITY: 20

GOAL: DEADLINE
ACTIVE WINDOW: (16:15-16:45)
STATUS: Active:On time
ESTIMATED FINISH TIME: (Wed Jul 2 16:38)
OPTIMUM NUMBER OF RUNNING JOBS: 1
```

NJOBS	PEND	RUN	SSUSP	USUSP	FINISH
6	5	1	0	0	0

The following illustrates the progress of the SLA to the deadline. The optimum number of running jobs in the service class (nrun) is maintained at a steady rate of 1 job at a time until near the completion of the SLA.

When the finished job curve (nfinished) meets the total number of jobs curve (njobs) the deadline is met. All jobs are finished well ahead of the actual configured deadline, and the goal of the SLA was met.



An overnight run with two service classes

bsl a shows the configuration and status of two service classes Qual i cum and Comox:

- Qual i cum has a deadline goal with a time window which is active overnight:

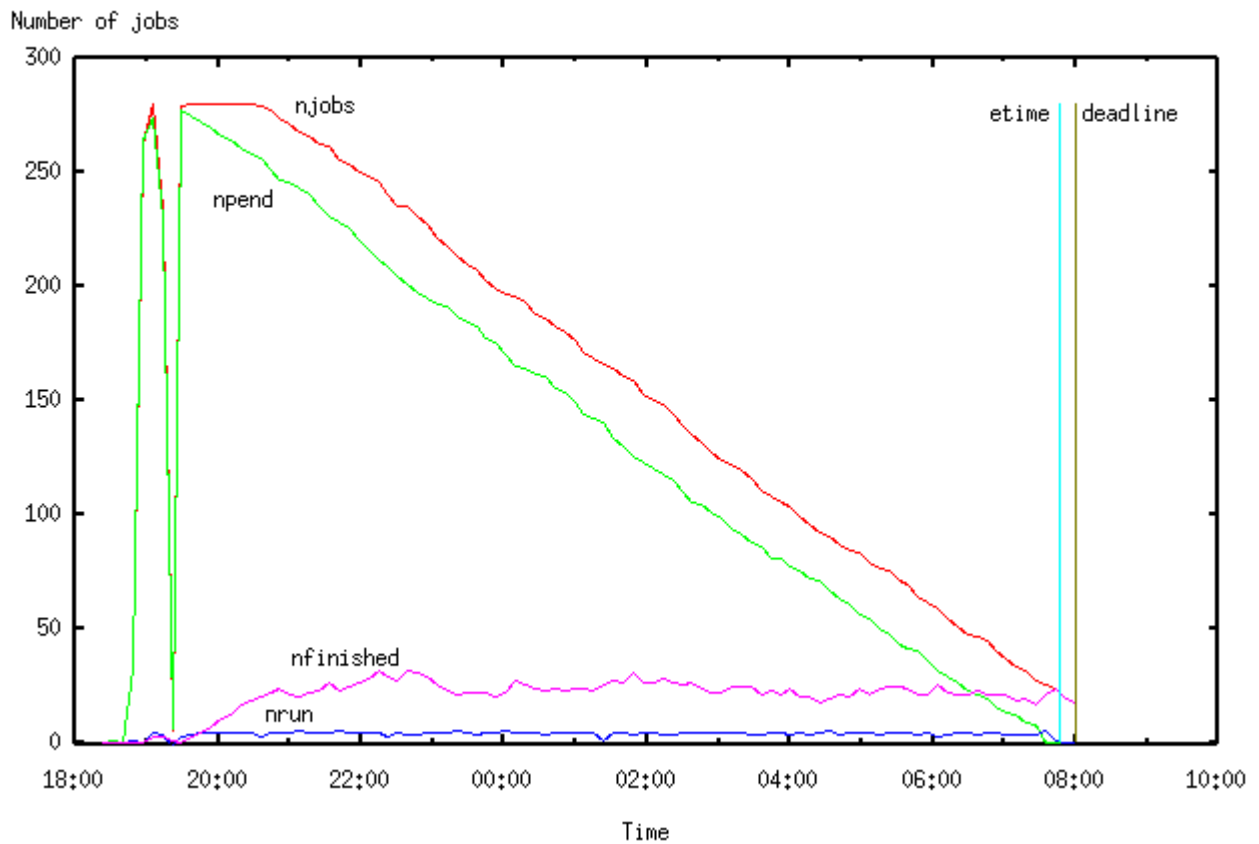
```
bsla Qualicum
SERVICE CLASS NAME:  Qual i cum
PRIORITY:  23

GOAL:  VELOCITY 8
ACTIVE WINDOW:  (8: 00- 18: 00)
STATUS:  Inactive
SLA THROUGHPUT:  0. 00 JOBS/CLEAN_PERIOD

GOAL:  DEADLINE
ACTIVE WINDOW:  (18: 00- 8: 00)
STATUS:  Active: On time
ESTIMATED FINISH TIME:  (Thu Jul 10 07: 53)
OPTIMUM NUMBER OF RUNNING JOBS:  2

NJOBS  PEND  RUN  SSUSP  USUSP  FINISH
280    278   2    0      0     0
```

The following illustrates the progress of the deadline SLA Qual i cum running 280 jobs overnight with random runtimes until the morning deadline. As with the simple deadline goal example, when the finished job curve (nfinished) meets the total number of jobs curve (njobs) the deadline is met with all jobs completed ahead of the configured deadline.



- Comox has a velocity goal of 2 concurrently running jobs that is always active:

bsla Comox

SERVICE CLASS NAME: Comox

PRIORITY: 20

GOAL: VELOCITY 2

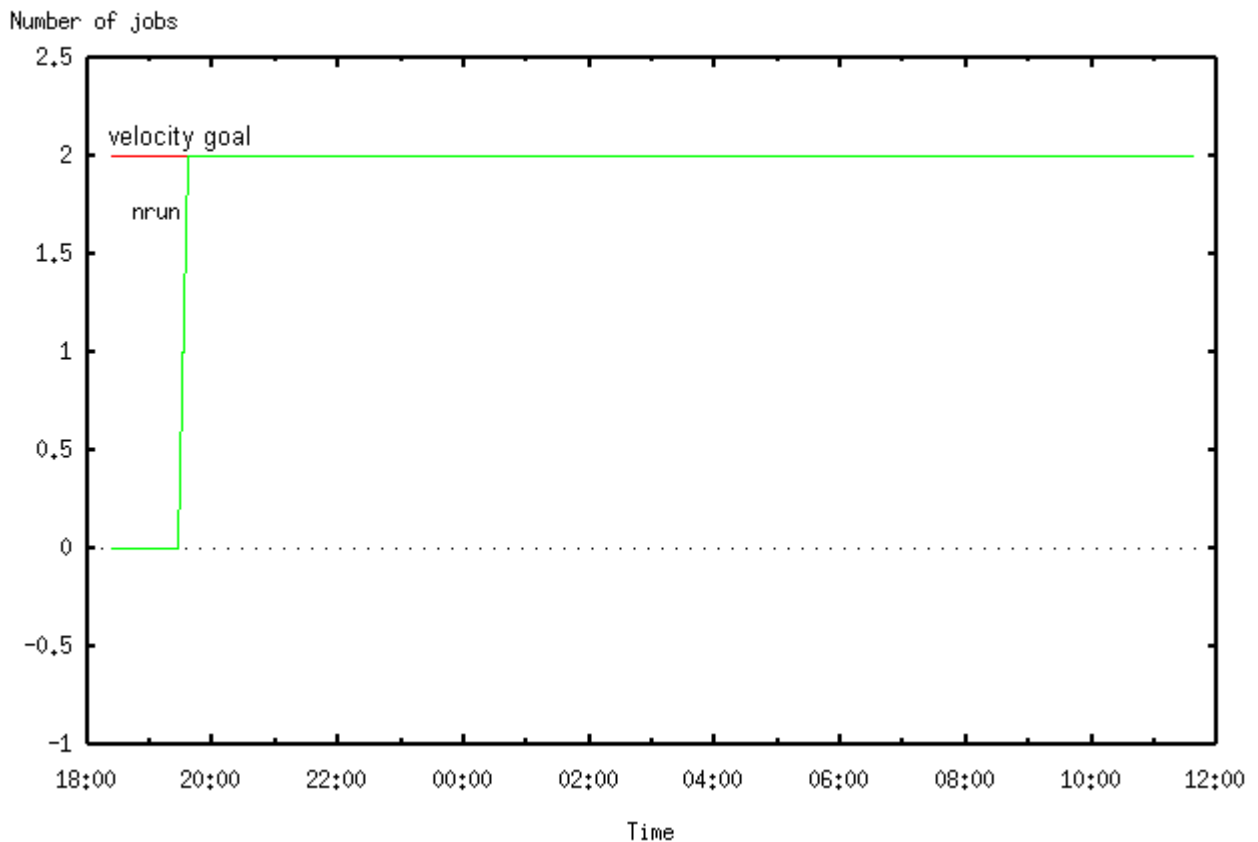
ACTIVE WINDOW: Always Open

STATUS: Active: On time

SLA THROUGHPUT: 2.00 JOBS/CLEAN_PERIOD

NJOBS	PEND	RUN	SSUSP	USUSP	FINISH
100	98	2	0	0	0

The following illustrates the progress of the velocity SLA Comox running 100 jobs with random runtimes over a 14 hour period.



Job groups and time-based SLAs

Job groups provide a method for assigning arbitrary labels to groups of jobs. Typically, job groups represent a project hierarchy. You can use `-g` with `-sla` at job submission to attach all jobs in a job group to a service class and have them scheduled as SLA jobs, subject to the scheduling policy of the SLA. Within the job group, resources are allocated to jobs on a fairshare basis.

All jobs submitted to a group under an SLA automatically belong to the SLA itself. You cannot modify a job group of a job that is attached to an SLA.

A job group hierarchy can belong to only one SLA.

It is not possible to have some jobs in a job group not part of the service class. Multiple job groups can be created under the same SLA. You can submit additional jobs to the job group without specifying the service class name again.

If the specified job group does not exist, it is created and attached to the SLA.

You can also use `-sl a` to specify a service class when you create a job group with `bgadd`.

View job groups attached to a time-based SLA (bjgroup)

1. Run `bj group` to display job groups attached to a time-based SLA:

```
bjgroup
```

GROUP_NAME	NJOBS	PEND	RUN	SSUSP	USUSP	FINISH	SLA	JLIMIT	OWNER
/fund1_grp	5	4	0	1	0	0	Venezia	1/5	user1
/fund2_grp	11	2	5	0	0	4	Venezia	5/5	user1
/bond_grp	2	2	0	0	0	0	Venezia	0/-	user2
/risk_grp	2	1	1	0	0	0	()	1/-	user2
/admi_grp	4	4	0	0	0	0	()	0/-	user2

bj group displays the name of the service class that the job group is attached to with `bgadd -sl a service_class_name`. If the job group is not attached to any service class, empty parentheses () are displayed in the SLA name column.

SLA CONTROL_ACTION parameter (lsb.serviceclasses)

Configure a specific action to occur when a time-based SLA is missing its goal.

1. Use the CONTROL_ACTION parameter in your service class to configure an action to be run if the time-based SLA goal is delayed for a specified number of minutes.

CONTROL_ACTION=VIOLATION_PERIOD[*minutes*] CMD [*action*]

If the SLA goal is delayed for longer than VIOLATION_PERIOD, the action specified by CMD is invoked. The violation period is reset and the action runs again if the SLA is still active when the violation period expires again. If the time-based SLA has multiple active goals that are in violation, the action is run for each of them.

Example

```
CONTROL_ACTION=VIOLATION_PERIOD[10] CMD [echo `date`: SLA is in violation
>> ! /tmp/sla_violation.log]
```

Submit jobs to a service class

The service class name where the job is to run is configured in `lsb. serviceclasses`. If the SLA does not exist or the user is not a member of the service class, the job is rejected.

If the SLA is not active or the guarantee SLA has used all guaranteed resources, LSF schedules jobs without enforcing any service-level goals. Jobs will flow through queues following queue priorities even if they are submitted with `-sla`, and will not make use of any guaranteed resources.

1. Run `bsub -sla service_class_name` to submit a job to a service class for SLA-driven scheduling.

```
bsub -W 15 -sla Duncan sleep 100
```

submits the UNIX command `sleep` together with its argument `100` as a job to the service class named `Duncan`.

Modify SLA jobs (bmod)

1. Run `bmod -sla` to modify the service class a job is attached to, or to attach a submitted job to a service class. Run `bmod -slan` to detach a job from a service class:

```
bmod -sla Duncan 2307
```

Attaches job `2307` to the service class `Duncan`.

```
bmod -slan 2307
```

Detaches job `2307` from the service class `Duncan`.

For all SLAs you cannot:

- Use `-sla` with other `bmod` options
- Modify the service class of jobs already attached to a job group

For time-based SLAs you cannot:

- Move job array elements from one service class to another, only entire job arrays

View configured guarantee resource pools

Resource-type SLAs have the host or slot guarantee configured within the guaranteed resource pool.

1. Run `bresources -g -l -m` to see details of the guaranteed resource pool configuration, including a list of hosts currently in the resource pool. For example:

Monitor the progress of an SLA (bsla)

1. Run `bsla` to display the properties of service classes configured in `lsb. serviceclasses` and dynamic information about the state of each configured service class.

Examples

- The guarantee SLA `bigMemSLA` has 10 slots guaranteed, limited to one slot per host.

```
bsla
SERVICE CLASS NAME:  bi gMemSLA
--
ACCESS CONTROL:  QUEUES[ normal ]
```

AUTO ATTACH: Y

GOAL: GUARANTEE

POOL NAME	TYPE	GUARANTEED	USED
bigMemPool	slots	10	0

- One velocity goal of service class Tofino is active and on time. The other configured velocity goal is inactive.

bsla

SERVICE CLASS NAME: Tofino
-- day and night velocity
PRIORITY: 20

GOAL: VELOCITY 30
ACTIVE WINDOW: (17:30-8:30)
STATUS: Inactive
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

GOAL: VELOCITY 10
ACTIVE WINDOW: (9:00-17:00)
STATUS: Active: On time
SLA THROUGHPUT: 10.00 JOBS/CLEAN_PERIOD

NJOBS	PEND	RUN	SSUSP	USUSP	FINISH
300	280	10	0	0	10

- The deadline goal of service class Sooke is not being met, and bsl a displays status Active: Delayed:

bsla

SERVICE CLASS NAME: Sooke
-- working hours
PRIORITY: 20

GOAL: DEADLINE
ACTIVE WINDOW: (8:30-19:00)
STATUS: Active: Delayed
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

ESTIMATED FINISH TIME: (Tue Oct 28 06:17)
OPTIMUM NUMBER OF RUNNING JOBS: 6
NJOBS PEND RUN SSUSP USUSP FINISH
40 39 1 0 0 0

- The configured velocity goal of the service class Duncan is active and on time. The configured deadline goal of the service class is inactive.

bsla Duncan

SERVICE CLASS NAME: Duncan
-- Daytime/Nighttime SLA
PRIORITY: 23
USER_GROUP: user1 user2

GOAL: VELOCITY 8
ACTIVE WINDOW: (9:00-17:30)
STATUS: Active: On time
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

GOAL: DEADLINE
ACTIVE WINDOW: (17:30-9:00)
STATUS: Inactive
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

NJOBS	PEND	RUN	SSUSP	USUSP	FINISH
0	0	0	0	0	0

- The throughput goal of service class Sidney is always active. bsl a displays:
 - Status as active and on time
 - An optimum number of 5 running jobs to meet the goal

- Actual throughput of 10 jobs per hour based on the last CLEAN_PERIOD

```

bsla Sidney
SERVICE CLASS NAME:  Si dney
-- constant throughput
PRIORITY:  20

GOAL:  THROUGHPUT 6
ACTIVE WINDOW: Always Open
STATUS:  Active:On time
SLA THROUGHPUT:  10.00 JOBS/CLEAN_PERIOD
OPTIMUM NUMBER OF RUNNING JOBS:  5

NJOB  PEND  RUN  SSUSP  USUSP  FINISH
110    95    5    0      0     10

```

View jobs running in an SLA (bjobs)

1. Run `bjobs -sla` to display jobs running in a service class:

```

bjobs -sla Sidney
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
136    user1  RUN   normal  hostA      hostA      sleep 100  Sep 28 13:24
137    user1  RUN   normal  hostA      hostB      sleep 100  Sep 28 13:25

```

For time-based SLAs, use `-sla with -g` to display job groups attached to a service class. Once a job group is attached to a time-based service class, all jobs submitted to that group are subject to the SLA.

Track historical behavior of an SLA (bacct)

1. Run `bacct` to display historical performance of a service class. For example, service classes `Sidney` and `Surrey` configure throughput goals.

```

bsla
SERVICE CLASS NAME:  Si dney
-- throughput 6
PRIORITY:  20

GOAL:  THROUGHPUT 6
ACTIVE WINDOW: Always Open
STATUS:  Active:On time
SLA THROUGHPUT:  10.00 JOBS/CLEAN_PERIOD
OPTIMUM NUMBER OF RUNNING JOBS:  5

NJOB  PEND  RUN  SSUSP  USUSP  FINISH
111    94    5    0      0     12
-----
SERVICE CLASS NAME:  Surrey
-- throughput 3
PRIORITY:  15

GOAL:  THROUGHPUT 3
ACTIVE WINDOW: Always Open
STATUS:  Active:On time
SLA THROUGHPUT:  4.00 JOBS/CLEAN_PERIOD
OPTIMUM NUMBER OF RUNNING JOBS:  4

NJOB  PEND  RUN  SSUSP  USUSP  FINISH
104    96    4    0      0     4

```

These two service classes have the following historical performance. For SLA `Sidney`, `bacct` shows a total throughput of 8.94 jobs per hour over a period of 20.58 hours:

```

bacct -sla Sidney
Accounting information about jobs that are:

```

- submitted by users user1,
- accounted on all projects.
- completed normally or exited
- executed on all hosts.
- submitted to all queues.
- accounted on service classes Sidney,

```

SUMMARY:      ( time unit: second )
Total number of done jobs:      183      Total number of exited jobs:      1
Total CPU time consumed:      40.0      Average CPU time consumed:      0.2
Maximum CPU time of a job:      0.3      Minimum CPU time of a job:      0.1
Total wait time in queues: 1947454.0
Average wait time in queue: 10584.0
Maximum wait time in queue: 18912.0      Minimum wait time in queue:      7.0
Average turnaround time:      12268 (seconds/job)
Maximum turnaround time:      22079      Minimum turnaround time:      1713
Average hog factor of a job: 0.00 ( cpu time / turnaround time )
Maximum hog factor of a job: 0.00      Minimum hog factor of a job: 0.00
Total throughput:      8.94 (jobs/hour) during 20.58 hours
Beginning time:      Oct 11 20: 23      Ending time:      Oct 12 16: 58

```

For SLA Surrey, bacct shows a total throughput of 4.36 jobs per hour over a period of 19.95 hours:

bacct -sla Surrey

Accounting information about jobs that are:

- submitted by users user1,
- accounted on all projects.
- completed normally or exited.
- executed on all hosts.
- submitted to all queues.
- accounted on service classes Surrey,

```

SUMMARY:      ( time unit: second )
Total number of done jobs:      87      Total number of exited jobs:      0
Total CPU time consumed:      18.0      Average CPU time consumed:      0.2
Maximum CPU time of a job:      0.3      Minimum CPU time of a job:      0.1
Total wait time in queues: 2371955.0
Average wait time in queue: 27263.8
Maximum wait time in queue: 39125.0      Minimum wait time in queue:      7.0
Average turnaround time:      30596 (seconds/job)
Maximum turnaround time:      44778      Minimum turnaround time:      3355
Average hog factor of a job: 0.00 ( cpu time / turnaround time )
Maximum hog factor of a job: 0.00      Minimum hog factor of a job: 0.00
Total throughput:      4.36 (jobs/hour) during 19.95 hours
Beginning time:      Oct 11 20: 50      Ending time:      Oct 12 16: 47

```

Because the run times are not uniform, both service classes actually achieve higher throughput than configured.

34

Exclusive Scheduling

Use exclusive scheduling

Exclusive scheduling gives a job exclusive use of the host that it runs on. LSF dispatches the job to a host that has no other jobs running, and does not place any more jobs on the host until the exclusive job is finished.

Compute unit exclusive scheduling gives a job exclusive use of the compute unit that it runs on.

How exclusive scheduling works

When an exclusive job (`bsub -x`) is submitted to an exclusive queue (`EXCLUSIVE = Y` or `=CU` in `lsb.queues`) and dispatched to a host, LSF locks the host (`lockU` status) until the job finishes.

LSF cannot place an exclusive job unless there is a host that has no jobs running on it.

To make sure exclusive jobs can be placed promptly, configure some hosts to run one job at a time. Otherwise, a job could wait indefinitely for a host in a busy cluster to become completely idle.

Resizable jobs

For pending allocation requests with resizable exclusive jobs, LSF does not allocate slots on a host that is occupied by the original job. For newly allocated hosts, LSF locks the LIM if `LSB_DISABLE_LIMLOCK_EXCL=Y` is not defined in `lsf.conf`.

If an entire host is released by a job resize release request with exclusive jobs, LSF unlocks the LIM if `LSB_DISABLE_LIMLOCK_EXCL=Y` is not defined in `lsf.conf`.

Restriction:

Jobs with compute unit resource requirements cannot be auto-resizable.
Resizable jobs with compute unit resource requirements cannot increase job resource allocations, but can release allocated resources.

Configure an exclusive queue

- To configure an exclusive queue, set `EXCLUSIVE` in the queue definition (`lsb.queues`) to `Y`.
`EXCLUSIVE=CU` also configures the queue to accept exclusive jobs when no compute unit resource requirement is specified.

Configure a host to run one job at a time

- To make sure exclusive jobs can be placed promptly, configure some single-processor hosts to run one job at a time. To do so, set `SLOTS=1` and `HOSTS=all` in `lsb.resources`.

Submit a exclusive job

- To submit an exclusive job, use the `-x` option of `bsub` and submit the job to an exclusive queue.

Configure a compute unit exclusive queue

- To configure an exclusive queue, set `EXCLUSIVE` in the queue definition (`lsb.queues`) to `CU` [`cu_type`].

If no compute unit type is specified, the default compute unit type defined in `COMPUTE_UNIT_TYPES` (`lsb.params`) is used.

Submit a compute unit exclusive job

1. To submit an exclusive job, use the -R option of bsub and submit the job to a compute unit exclusive queue.

```
bsub -R "cu[excl]" my_job
```


Job Scheduling and Dispatch

Working with Application Profiles

Application profiles improve the management of applications by separating scheduling policies (preemption, fairshare, etc.) from application-level requirements, such as pre-execution and post-execution commands, resource limits, or job controls, job chunking, etc.

Manage application profiles

About application profiles

Use application profiles to map common execution requirements to application-specific job containers. For example, you can define different job types according to the properties of the applications that you use; your FLUENT jobs can have different execution requirements from your CATIA jobs, but they can all be submitted to the same queue.

The following application profile defines the execution requirements for the FLUENT application:

```
Begin Application
NAME = fluent
DESCRIPTION = FLUENT Version 6.2
CPULIMIT = 180/hostA # 3 hours of host hostA
FILELIMIT = 20000
DATA LIMIT = 20000 # jobs data segment limit
CORELIMIT = 20000
PROCLIMIT = 5 # job processor limit
PRE_EXEC = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
REQUEUE_EXIT_VALUES = 55 34 78
End Application
```

See the `lsb. applications` template file for additional application profile examples.

Add an application profile

1. Log in as the LSF administrator on any host in the cluster.
2. Edit `lsb. applications` to add the new application profile definition.

You can copy another application profile definition from this file as a starting point; remember to change the NAME of the copied profile.

3. Save the changes to `lsb. applications`.
4. Run `badminton reconfig` to reconfigure `mbatchd`.

Adding an application profile does not affect pending or running jobs.

Remove an application profile

Before removing an application profile, make sure there are no pending jobs associated with the application profile.

If there are jobs in the application profile, use `bmod -app` to move pending jobs to another application profile, then remove the application profile. Running jobs are not affected by removing the application profile associated with them,

Note:

You cannot remove a default application profile.

1. Log in as the LSF administrator on any host in the cluster.
2. Run `bmod -app` to move all pending jobs into another application profile.

If you leave pending jobs associated with an application profile that has been removed, they remain pending with the pending reason

Specified application profile does not exist

3. Edit `lsb. applications` and remove or comment out the definition for the application profile you want to remove.
4. Save the changes to `lsb. applications`.
5. Run `badm n reconfi g` to reconfigure `mbatchd`.

Define a default application profile

Define a default application profile that is used when a job is submitted without specifying an application profile,

1. Log in as the LSF administrator on any host in the cluster.
2. Set `DEFAULT_APPLICATION` in `lsb. params` to the name of the default application profile.

```
DEFAULT_APPLICATION=catia
```

3. Save the changes to `lsb. params`.
4. Run `badm n reconfi g` to reconfigure `mbatchd`.

Adding an application profile does not affect pending or running jobs.

Understand successful application exit values

Jobs that exit with one of the exit codes specified by `SUCCESS_EXIT_VALUES` in an application profile are marked as `DONE`. These exit values are not be counted in the `EXIT_RATE` calculation.

0 always indicates application success regardless of `SUCCESS_EXIT_VALUES`.

If both `SUCCESS_EXIT_VALUES` and `REQUEUE_EXIT_VALUES` are defined, job will be set to `PEND` state and requeued.

`SUCCESS_EXIT_VALUES` has no effect on pre-exec and post-exec commands. The value is only used for user jobs.

If the job exit value falls into `SUCCESS_EXIT_VALUES`, the job will be marked as `DONE`. Job dependencies on done jobs behave normally.

For parallel jobs, the exit status refers to the job exit status and not the exit status of individual tasks.

Exit codes for jobs terminated by LSF are excluded from success exit value even if they are specified in `SUCCESS_EXIT_VALUES`.

For example, if `SUCCESS_EXIT_VALUES=2` is defined, jobs exiting with 2 are marked as `DONE`. However, if LSF cannot find the current working directory, LSF terminates the job with exit code 2, and the job is marked as `EXIT`. The appropriate termination reason is displayed by `bacct`.

MultiCluster jobs

In the job forwarding model, for jobs sent to a remote cluster, jobs exiting with success exit codes defined in the remote cluster are considered done successfully.

In the lease model, the parameters of `lsb. applications` apply to jobs running on remote leased hosts as if they are running on local hosts.

Specify successful application exit values.

Use `SUCCESS_EXIT_VALUES` to specify a list of exit codes that will be considered as successful execution for the application.

1. Log in as the LSF administrator on any host in the cluster.

2. Set SUCCESS_EXIT_VALUES to specify a list of job success exit codes for the application.

```
SUCCESS_EXIT_VALUES=230 222 12
```

3. Save the changes to lsb. applications.
4. Run `badm n reconfi g` to reconfigure `mbatchd`.

Submit jobs to application profiles

Use the `-app` option of `bsub` to specify an application profile for the job.

1. Run `bsub -app` to submit jobs to an application profile.

```
bsub -app fluent -q overnight myjob
```

LSF rejects the job if the specified application profile does not exist.

Modify the application profile associated with a job

You can only modify the application profile for pending jobs.

1. Run `bmod -app application_profile_name` to modify the application profile of the job.

The `-appn` option dissociates the specified job from its application profile. If the application profile does not exist, the job is not modified.

```
bmod -app fluent 2308
```

Associates job 2308 with the application profile `fluent`.

```
bmod -appn 2308
```

Dissociates job 2308 from the application profile `fluent`.

Control jobs associated with application profiles

`bstop`, `bresume`, and `bkill` operate on jobs associated with the specified application profile. You must specify an existing application profile. If `job_ID` or 0 is not specified, only the most recently submitted qualifying job is operated on.

1. Run `bstop -app` to suspend jobs in an application profile.

```
bstop -app fluent 2280
```

Suspends job 2280 associated with the application profile `fluent`.

```
bstop -app fluent 0
```

Suspends all jobs associated with the application profile `fluent`.

2. Run `bresume -app` to resume jobs in an application profile.

```
bresume -app fluent 2280
```

Resumes job 2280 associated with the application profile `fluent`.

3. Run `bkill -app` to kill jobs in an application profile.

```
bkill -app fluent
```

Kills the most recently submitted job associated with the application profile `fluent` for the current user.

```
bkill -app fluent 0
```

Kills all jobs associated with the application profile `fluent` for the current user.

View application profile information

To view the...	Run...
Available application profiles	bapp
Detailed application profile information	bapp -l
Jobs associated with an application profile	bj obs -l - app <i>application_profile_name</i>
Accounting information for all jobs associated with an application profile	bacct -l - app <i>application_profile_name</i>
Job success and requeue exit code information	bapp -l bacct -l bhi st -l bj obs -l

View available application profiles

1. Run bapp. You can view a particular application profile or all profiles.

```
bapp
APPLI CATION_NAME  NJOBS  PEND  RUN  SUSP
fluent              0       0    0    0
catia               0       0    0    0
```

A dash (-) in any entry means that the column does not apply to the row.

View detailed application profile information

1. To see the complete configuration for each application profile, run bapp -l .

bapp -l also gives current statistics about the jobs in a particular application profile, such as the total number of jobs in the profile, the number of jobs running, suspended, and so on.

Specify application profile names to see the properties of specific application profiles.

```
bapp -l fluent
APPLICATION NAME: fluent
-- Application definition for Fluent v2.0
STATISTICS:
      NJOBS      PEND      RUN      SSUSP      USUSP      RSV
         0         0         0         0         0         0

PARAMETERS:
CPULIMIT
600.0 min of hostA
RUNLIMIT
200.0 min of hostA
PROCLIMIT
9
FILELIMIT DATALIMIT STACKLIMIT CORELIMIT MEMLIMIT SWAPLIMIT PROCESSLIMIT
THREADLIMIT
      800 K      100 K      900 K      700 K      300 K      1000 K      400      500
RERUNNABLE: Y
CHUNK_JOB_SIZE: 5
```

View jobs associated with application profiles

1. Run `bjobs -l -app application_profile_name`.

bjobs -l -app fluent

```
Job <1865>, User <user1>, Project <default>, Application <fluent>,
      Status <PSUSP>, Queue <normal>, Command <ls>
Tue Jun  6 11:52:05 2009: Submitted from host <hostA> with hold, CWD
      </clusters/lsf8.0/work/cluster1/logdir>;
```

PENDING REASONS:

Job was suspended by LSF admin or root while pending;

SCHEDULING PARAMETERS:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem	tl u
loadSched	-	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-	-
		cpuspeed		bandwi dth								
loadSched	-			-								
loadStop	-	-		-								

A dash (-) in any entry means that the column does not apply to the row.

Accounting information for all jobs associated with an application profile

1. Run `bacct -l -app application_profile_name`.

bacct -l -app fluent

Accounting information about jobs that are:

- submitted by users jchan,
- accounted on all projects.
- completed normally or exited
- executed on all hosts.
- submitted to all queues.
- accounted on all service classes.
- associated with application profiles: fluent

```
Job <207>, User <user1>, Project <default>, Application <fluent>, Status <DONE>
      , Queue <normal>, Command <dir>
Wed May 31 16:52:42 2009: Submitted from host <hostA>, CWD <$HOME/src/mainline/lsbatch/cmd>;
Wed May 31 16:52:48 2009: Dispatched to 10 Hosts/Processors <10*hostA>
Wed May 31 16:52:48 2009: Completed <done>.
```

Accounting information about this job:

CPU_T	WAIT	TURNAROUND	STATUS	HOG_FACTOR	MEM	SWAP
0.02	6	6	done	0.0035	2M	5M

```
...
SUMMARY:      ( time unit: second )
Total number of done jobs:      15      Total number of exited jobs:      4
Total CPU time consumed:      0.4      Average CPU time consumed:      0.0
Maximum CPU time of a job:      0.0      Minimum CPU time of a job:      0.0
Total wait time in queues: 5305.0
Average wait time in queue: 279.2
Maximum wait time in queue: 3577.0      Minimum wait time in queue:      2.0
Average turnaround time:      306 (seconds/job)
Maximum turnaround time:      3577      Minimum turnaround time:      5
Average hog factor of a job: 0.00 ( cpu time / turnaround time )
Maximum hog factor of a job: 0.01      Minimum hog factor of a job: 0.00
Total throughput:      0.14 (jobs/hour) during 139.98 hours
Beginning time:      May 31 16:52      Ending time:      Jun  6 12:51
```

View job success exit values and requeue exit code information

1. Run `bjobs -l` to see command-line requeue exit values if defined.

bjobs -l

```

Job <405>, User <user1>, Project <default>, Status <PSUSP>, Queue <normal>, Command
<myjob 1234>
Tue Dec 11 23:32:00 2009: Submitted from host <hostA> with hold, CWD </scratch/d
ev/lsfjobs/user1/work>, Requeue Exit Values <2>;
...

```

2. Run `bapp -l` to see `SUCCESS_EXIT_VALUES` when the parameter is defined in an application profile.

```

bapp -l
APPLICATION NAME: fluent
-- Run FLUENT applications
STATISTICS:
  NJOBS      PEND      RUN      SSUSP      USUSP      RSV
    0         0         0         0         0         0
PARAMETERS:
SUCCESS_EXIT_VALUES: 230 222 12
...

```

3. Run `bhist -l` to show command-line specified requeue exit values with `bsub` and modified requeue exit values with `bmod`.

```

bhist -l
Job <405>, User <user1>, Project <default>, Command <myjob 1234>
Tue Dec 11 23:32:00 2009: Submitted from host <hostA> with hold, to Queue
<normal>
                                l>, CWD </scratch/dev/lsfjobs/user1/work>, R
                                e-queue Exit Values <1>;
Tue Dec 11 23:33:14 2009: Parameters of Job are changed:
                                Requeue exit values changes to: 2;
...

```

4. Run `bhist -l` and `bacct -l` to see success exit values when a job is done successfully. If the job exited with default success exit value 0, `bhist` and `bacct` do not display the 0 exit value

```

bhist -l 405
Job <405>, User <user1>, Project <default>, Interactive pseudo-terminal mode, Co
mmand <myjob 1234>
...
Sun Oct 7 22:30:19 2009: Done successfully. Success Exit Code: 230 222 12.
...

```

```

bacct -l 405
...
Job <405>, User <user1>, Project <default>, Status <DONE>, Queue <normal>, Comma
nd <myjob 1234>
Wed Sep 26 18:37:47 2009: Submitted from host <hostA>, CWD </scratch/dev/lsfjobs/
user1/work>,
                                rk>;
Wed Sep 26 18:37:50 2009: Dispatched to <hostA>;
Wed Sep 26 18:37:51 2009: Completed <done>. Success Exit Code: 230 222 12.
...

```

How application profiles interact with queue and job parameters

Application profiles operate in conjunction with queue and job-level options. In general, you use application profile definitions to refine queue-level settings, or to exclude some jobs from queue-level parameters.

Application profile settings that override queue settings

The following application profile parameters override the corresponding queue setting:

- `CHKPNT_DIR`—overrides queue `CHKPNT=chkpnt_dir`
- `CHKPNT_PERIOD`—overrides queue `CHKPNT=chkpnt_period`
- `JOB_STARTER`
- `LOCAL_MAX_PREEEXEC_RETRY`
- `MAX_JOB_PREEMPT`
- `MAX_JOB_REQUEUE`
- `MAX_PREEEXEC_RETRY`
- `MAX_TOTAL_TIME_PREEMPT`
- `MIG`
- `NICE`
- `NO_PREEMPT_INTERVAL`
- `REMOTE_MAX_PREEEXEC_RETRY`
- `REQUEUE_EXIT_VALUES`
- `RESUME_CONTROL`—overrides queue `JOB_CONTROLS`
- `SUSPEND_CONTROL`—overrides queue `JOB_CONTROLS`
- `TERMINATE_CONTROL`—overrides queue `JOB_CONTROLS`

Application profile limits and queue limits

The following application profile limits override the corresponding queue-level soft limits:

- `CORELIMIT`
- `CPULIMIT`
- `DATALIMIT`
- `FILELIMIT`
- `MEMLIMIT`
- `PROCESSLIMIT`
- `RUNLIMIT`
- `STACKLIMIT`
- `SWAPLIMIT`
- `STACKLIMIT`
- `THREADLIMIT`

Job-level limits can override the application profile limits. The application profile limits cannot override queue-level hard limits.

Processor limits

PROCLIMIT in an application profile specifies the maximum number of slots that can be allocated to a job. For parallel jobs, PROCLIMIT is the maximum number of processors that can be allocated to the job.

You can optionally specify the minimum and default number of processors. All limits must be positive integers greater than or equal to 1 that satisfy the following relationship:

$$1 \leq \text{minimum} \leq \text{default} \leq \text{maximum}$$

Job-level processor limits (bsub -n) override application-level PROCLIMIT, which overrides queue-level PROCLIMIT. Job-level limits must fall within the maximum and minimum limits of the application profile and the queue.

Absolute run limits

If you want the scheduler to treat any run limits as absolute, define ABS_RUNLIMIT=Y in lsb.params or in lsb.applications for the application profile associated with your job. When ABS_RUNLIMIT=Y is defined in lsb.params or in the application profile, the run time limit is not normalized by the host CPU factor. Absolute wall-clock run time is used for all jobs submitted with a run limit configured.

Pre-execution

Queue-level pre-execution commands run *before* application-level pre-execution commands. Job level pre-execution commands (bsub -E) override application-level pre-execution commands.

Post-execution

When a job finishes, application-level post-execution commands run, followed by queue-level post-execution commands if any.

If both application-level and job-level post-execution commands (bsub -Ep) are specified, job level post-execution overrides application-level post-execution commands. Queue-level post-execution commands run after application-level post-execution and job-level post-execution commands

Chunk job scheduling

CHUNK_JOB_SIZE in an application profile ensures that jobs associated with the application are chunked together. CHUNK_JOB_SIZE=1 disables job chunk scheduling. Application-level job chunk definition overrides chunk job dispatch configured in the queue.

CHUNK_JOB_SIZE is ignored and jobs are not chunked under the following conditions:

- CPU limit greater than 30 minutes (CPULIMIT parameter in lsb.queues or lsb.applications)
- Run limit greater than 30 minutes (RUNLIMIT parameter in lsb.queues or lsb.applications)
- Run time estimate greater than 30 minutes (RUNTIME parameter in lsb.applications)

If CHUNK_JOB_DURATION is set in lsb.params, chunk jobs are accepted regardless of the value of CPULIMIT, RUNLIMIT or RUNTIME.

Rerunnable jobs

RERUNNABLE in an application profile overrides queue-level job rerun, and allows you to submit rerunnable jobs to a non-rerunnable queue. Job-level rerun (`bsub -r` or `bsub -rn`) overrides both the application profile and the queue.

Resource requirements

Application-level resource requirements can be simple (one requirement for all slots) or compound (different requirements for specified numbers of slots). When resource requirements are set at the application-level as well as the job-level or queue-level, the requirements are combined in different ways depending on whether they are simple or compound.

Simple job-level, application-level, and queue-level resource requirements are merged in the following manner:

- If resource requirements are not defined at the application level, simple job-level and simple queue-level resource requirements are merged.
- When simple application-level resource requirements are defined, simple job-level requirements usually take precedence. Specifically:

section	simple resource requirement multi-level behavior
select	all levels satisfied
same	all levels combined
order span cu	job-level section overwrites application-level section, which overwrites queue-level section (if a given level is present)
rusage	all levels merge if conflicts occur the job-level section overwrites the application-level section, which overwrites the queue-level section.

Compound application-level resource requirements are merged in the following manner:

- When a compound resource requirement is set at the application level, it will be ignored if any job-level resource requirements (simple or compound) are defined.
- In the event no job-level resource requirements are set, the compound application-level requirements interact with queue-level resource requirement strings in the following ways:
 - If no queue-level resource requirement is defined or a compound queue-level resource requirement is defined, the compound application-level requirement is used.
 - If a simple queue-level requirement is defined, the application-level and queue-level requirements combine as follows:

section	compound application and simple queue behavior
select	both levels satisfied; queue requirement applies to all compound terms
same	queue level ignored
order span	application-level section overwrites queue-level section (if a given level is present); queue requirement (if used) applies to all compound terms

section	compound application and simple queue behavior
rusage	<ul style="list-style-type: none"> • both levels merge • queue requirement if a job-based resource is applied to the first compound term, otherwise applies to all compound terms • if conflicts occur the application-level section overwrites the queue-level section. <p>For example: if the application-level requirement is $\text{num1} * \{ \text{rusage} [R1] \} + \text{num2} * \{ \text{rusage} [R2] \}$ and the queue-level requirement is $\text{rusage} [RQ]$ where RQ is a job resource, the merged requirement is $\text{num1} * \{ \text{rusage} [\text{merge}(R1, RQ)] \} + \text{num2} * \{ \text{rusage} [R2] \}$</p>

For internal load indices and duration, jobs are rejected if they specify resource reservation requirements at the job level or application level that exceed the requirements specified in the queue.

If `RES_REQ` is defined at the queue level and there are no load thresholds defined, the pending reasons for each individual load index will not be displayed by `bj obs`.

When `LSF_STRICT_RESREQ=Y` is configured in `lsf.conf`, resource requirement strings in `select` sections must conform to a more strict syntax. The strict resource requirement syntax only applies to the `select` section. It does not apply to the other resource requirement sections (`order`, `rusage`, `same`, `span`, or `cu`). When `LSF_STRICT_RESREQ=Y` in `lsf.conf`, LSF rejects resource requirement strings where an `rusage` section contains a non-consumable resource.

When the parameter `RESRSV_LIMIT` in `lsb.queues` is set, the merged application-level and job-level `rusage` consumable resource requirements must satisfy any limits set by `RESRSV_LIMIT`, or the job will be rejected.

Estimated runtime and runtime limits

Instead of specifying an explicit runtime limit for jobs, you can specify an *estimated* run time for jobs. LSF uses the estimated value for job scheduling purposes only, and does not kill jobs that exceed this value unless the jobs also exceed a defined runtime limit. The format of runtime estimate is same as `run limit` set by the `bsub -W` option or the `RUNLIMIT` parameter in `lsb.queues` and `lsb.appl icat i ons`.

Use `JOB_RUNLIMIT_RATIO` in `lsb.params` to limit the runtime estimate users can set. If `JOB_RUNLIMIT_RATIO` is set to 0 no restriction is applied to the runtime estimate. The ratio does not apply to the `RUNTIME` parameter in `lsb.appl icat i ons`.

The job-level runtime estimate setting overrides the `RUNTIME` setting in an application profile in `lsb.appl icat i ons`.

The following LSF features use the estimated runtime value to schedule jobs:

- Job chunking
- Advance reservation
- SLA
- Slot reservation
- Backfill

Define a runtime estimate

Define the `RUNTIME` parameter at the application level. Use the `bsub -We` option at the job-level.

You can specify the runtime estimate as hours and minutes, or minutes only. The following examples show an application-level runtime estimate of three hours and 30 minutes:

- **RUNTIME=3:30**

- **RUNTIME=210**

Configure normalized run time

LSF uses normalized run time for scheduling in order to account for different processing speeds of the execution hosts.

Tip:

If you want the scheduler to use wall-clock (absolute) run time instead of normalized run time, define `ABS_RUNLIMIT=Y` in the file `l sb. params` or in the file `l sb. appl i cat i ons` for the application associated with your job.

LSF calculates the normalized run time using the following formula:

$$\text{NORMALIZED_RUN_TIME} = \text{RUNTIME} * \text{CPU_Factor_Normalizati on_Host} / \text{CPU_Factor_Execute_Host}$$

You can specify a host name or host model with the runtime estimate so that LSF uses a specific host name or model as the normalization host. If you do not specify a host name or host model, LSF uses the CPU factor for the default normalization host as described in the following table.

If you define...	In the file...	Then...
DEFAULT_HOST_SPEC	l sb. queues	LSF selects the default normalization host for the queue.
DEFAULT_HOST_SPEC	l sb. params	LSF selects the default normalization host for the cluster.
No default host at either the queue or cluster level		LSF selects the submission host as the normalization host.

To specify a host name (defined in `l sf. cl uster. cl ustername`) or host model (defined in `l sf. shared`) as the normalization host, insert the "/" character between the minutes and the host name or model, as shown in the following examples:

```
RUNTIME=3: 30/hostA
bsub -We 3: 30/hostA
```

LSF calculates the normalized run time using the CPU factor defined for host A.

```
RUNTIME=210/Ultra5S
bsub -We 210/Ultra5S
```

LSF calculates the normalized run time using the CPU factor defined for host model Ultra5S.

Tip:

Use `l si nfo` to see host name and host model information.

Guidelines for defining a runtime estimate

1. You can define an estimated run time, along with a runtime limit (job level with `bsub -W`, application level with `RUNLIMIT` in `l sb. appl i cat i ons`, or queue level with `RUNLIMIT l sb. queues`).
2. If the runtime limit is defined, the job-level (-We) or application-level `RUNTIME` value must be less than or equal to the run limit. LSF ignores the estimated runtime value and uses the run limit value for scheduling when
 - The estimated runtime value exceeds the run limit value, or

- An estimated runtime value is not defined

Note:

When LSF uses the run limit value for scheduling, and the run limit is defined at more than one level, LSF uses the smallest run limit value to estimate the job duration.

- For chunk jobs, ensure that the estimated runtime value is
 - Less than the `CHUNK_JOB_DURATION` defined in the file `l sb. params`, or
 - Less than 30 minutes, if `CHUNK_JOB_DURATION` is not defined.

How estimated run time interacts with run limits

The following table includes all the expected behaviors for the combinations of job-level runtime estimate (-We), job-level run limit (-W), application-level runtime estimate (RUNTIME), application-level run limit (RUNLIMIT), queue-level run limit (RUNLIMIT, both default and hard limit). *Ratio* is the value of `JOB_RUNLIMIT_RATIO` defined in `l sb. params`. The dash (—) indicates no value is defined for the job.

Job-runtime estimate	Job-run limit	Application runtime estimate	Application run limit	Queue default run limit	Queue hard run limit	Result
T1	-	—	—	—	—	Job is accepted Jobs running longer than $T1 * ratio$ are killed
T1	$T2 > T1 * ratio$	—	—	—	—	Job is rejected
T1	$T2 \leq T1 * ratio$	—	—	—	—	Job is accepted Jobs running longer than T2 are killed
T1	$T2 \leq T1 * ratio$	T3	T4	—	—	Job is accepted Jobs running longer than T2 are killed T2 overrides T4 or $T1 * ratio$ overrides T4 T1 overrides T3
T1	$T2 \leq T1 * ratio$	—	—	T5	T6	Job is accepted Jobs running longer than T2 are killed If $T2 > T6$, the job is rejected
T1	—	T3	T4	—	—	Job is accepted Jobs running longer than $T1 * ratio$ are killed T2 overrides T4 or $T1 * ratio$ overrides T4 T1 overrides T3

Job-runtime estimate	Job-run limit	Application runtime estimate	Application run limit	Queue default run limit	Queue hard run limit	Result
T1	—	—	—	T5	T6	Job is accepted Jobs running longer than $T1 * ratio$ are killed If $T1 * ratio > T6$, the job is rejected

Resource Allocation Limits

About resource allocation limits

What resource allocation limits do

By default, resource *consumers* like users, hosts, queues, or projects are not limited in the resources available to them for running jobs. *Resource allocation limits* configured in `lsb. resources` restrict:

- The maximum amount of a resource requested by a job that can be allocated during job scheduling for different classes of jobs to start
- Which resource consumers the limits apply to

If all of the resource has been consumed, no more jobs can be started until some of the resource is released.

For example, by limiting maximum amount of memory for each of your hosts, you can make sure that your system operates at optimal performance. By defining a memory limit for some users submitting jobs to a particular queue and a specified set of hosts, you can prevent these users from using up all the memory in the system at one time.

Jobs must specify resource requirements

For limits to apply, the job must specify resource requirements (`bsub -R` usage string or `RES_REQ` in `lsb. queues`). For example, the a memory allocation limit of 4 MB is configured in `lsb. resources`:

```
Begin Limit
NAME = mem_limit
MEM = 4
End Limit
```

A job submitted with an `rsusage` resource requirement that exceeds this limit:

```
bsub -R "rsusage[mem=5]" uname
```

and remains pending:

bjobs -p 600

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
600	user1	PEND	normal	suplin02		uname	Aug 12 14:05

Resource (mem) limit defined cluster-wide has been reached;

A job is submitted with a resource requirement within the configured limit:

```
bsub -R"rsusage[mem=3]" sleep 100
```

is allowed to run:

bjobs

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
600	user1	PEND	normal	hostA		uname	Aug 12 14:05
604	user1	RUN	normal	hostA		sleep 100	Aug 12 14:09

Resource usage limits and resource allocation limits

Resource allocation limits are not the same as *resource usage limits*, which are enforced during job run time. For example, you set CPU limits, memory limits, and other limits that take effect after a job starts running.

Resource reservation limits and resource allocation limits

Resource allocation limits are not the same as queue-based *resource reservation limits*, which are enforced during job submission. The parameter `RESRSV_LIMIT` (in `lsb. queues`) specifies allowed ranges of resource values, and jobs submitted with resource requests outside of this range are rejected.

How Platform LSF enforces limits

Resource allocation limits are enforced so that they apply to:

- Several kinds of resources:
 - Job slots by host
 - Job slots per processor
 - Running and suspended jobs
 - Memory (MB or percentage)
 - Swap space (MB or percentage)
 - Tmp space (MB or percentage)
 - Software licenses
 - Other shared resources
- Several kinds of resource consumers:
 - Users and user groups (all users or per-user)
 - Hosts and host groups (all hosts or per-host)
 - Queues (all queues or per-queue)
 - Projects (all projects or per-project)
- All jobs in the cluster
- Combinations of consumers:
 - For jobs running on different hosts in the same queue
 - For jobs running from different queues on the same host

How Platform LSF counts resources

Resources on a host are not available if they are taken by jobs that have been started, but have not yet finished. This means running and suspended jobs count against the limits for queues, users, hosts, projects, and processors that they are associated with.

Job slot limits

Job slot limits can correspond to the maximum number of jobs that can run at any point in time. For example, a queue cannot start jobs if it has no job slots available, and jobs cannot run on hosts that have no available job slots.

Limits such as `QJOB_LIMIT` (1 sb. queues), `HJOB_LIMIT` (1 sb. queues), `UJOB_LIMIT` (1 sb. queues), `MXJ` (1 sb. hosts), `JL/U` (1 sb. hosts), `MAX_JOBS` (1 sb. users), and `MAX_PEND_JOBS` (1 sb. users) limit the number of job slots. When the workload is sequential, job slots are usually equivalent to jobs. For parallel or distributed applications, these are true job slot limits and not job limits.

Job limits

Job limits, specified by `JOBS` in a Limit section in `l sb. resources`, correspond to the maximum number of running and suspended jobs that can run at any point in time. If both job limits and job slot limits are configured, the most restrictive limit is applied.

Resource reservation and backfill

When processor or memory reservation occurs, the reserved resources count against the limits for users, queues, hosts, projects, and processors. When backfilling of parallel jobs occurs, the backfill jobs do not count against any limits.

MultiCluster

Limits apply only to the cluster where `lsb.resouces` is configured. If the cluster leases hosts from another cluster, limits are enforced on those hosts as if they were local hosts.

Switched jobs can exceed resource allocation limits

If a switched job (`bswitch`) has not been dispatched, then the job behaves as if it were submitted to the new queue in the first place, and the JOBS limit is enforced in the target queue.

If a switched job has been dispatched, then resource allocation limits like SWP, TMP, and JOBS can be exceeded in the target queue. For example, given the following JOBS limit configuration:

```
Begin Limit
USERS      QUEUES      SLOTS      TMP      JOBS
-          normal      -          20       2
-          short       -          20       2
End Limit
```

Submit 3 jobs to the normal queue, and 3 jobs to the short queue:

```
bsub -q normal -R"rusage[tmp=20]" sleep 1000
bsub -q short -R"rusage[tmp=20]" sleep 1000
```

`bjobs` shows 1 job in RUN state in each queue:

```
bjobs
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
16     user1  RUN   normal hosta       hosta     sleep 1000 Aug 30 16:26
17     user1  PEND  normal hosta       hosta     sleep 1000 Aug 30 16:26
18     user1  PEND  normal hosta       hosta     sleep 1000 Aug 30 16:26
19     user1  RUN   short  hosta       hosta     sleep 1000 Aug 30 16:26
20     user1  PEND  short  hosta       hosta     sleep 1000 Aug 30 16:26
21     user1  PEND  short  hosta       hosta     sleep 1000 Aug 30 16:26
```

`blimits` shows the TMP limit reached:

```
blimits
INTERNAL RESOURCE LIMITS:
NAME  USERS  QUEUES  SLOTS  TMP  JOBS
NONAME000 -      normal  -      20/20  1/2
NONAME001 -      short   -      20/20  1/2
```

Switch the running job in the normal queue to the short queue:

```
bswitch short 16
```

`bjobs` shows 2 jobs running in the short queue, and the second job running in the normal queue:

```
bjobs
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
17     user1  RUN   normal hosta       hosta     sleep 1000 Aug 30 16:26
18     user1  PEND  normal hosta       hosta     sleep 1000 Aug 30 16:26
19     user1  RUN   short  hosta       hosta     sleep 1000 Aug 30 16:26
16     user1  RUN   short  hosta       hosta     sleep 1000 Aug 30 16:26
20     user1  PEND  short  hosta       hosta     sleep 1000 Aug 30 16:26
21     user1  PEND  short  hosta       hosta     sleep 1000 Aug 30 16:26
```

`blimits` now shows the TMP limit exceeded and the JOBS limit reached in the short queue:

```
blimits
INTERNAL RESOURCE LIMITS:
NAME  USERS  QUEUES  SLOTS  TMP  JOBS
NONAME000 -      normal  -      20/20  1/2
NONAME001 -      short   -      40/20  2/2
```

Switch the running job in the normal queue to the short queue:

```
bswitch short 17
```

`bjobs` now shows 3 jobs running in the short queue and the third job running in the normal queue:

```
bjobs
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
18	user1	RUN	normal	hosta	hosta	sleep 1000	Aug 30 16: 26
19	user1	RUN	short	hosta	hosta	sleep 1000	Aug 30 16: 26
16	user1	RUN	short	hosta	hosta	sleep 1000	Aug 30 16: 26
17	user1	RUN	short	hosta	hosta	sleep 1000	Aug 30 16: 26
20	user1	PEND	short	hosta		sleep 1000	Aug 30 16: 26
21	user1	PEND	short	hosta		sleep 1000	Aug 30 16: 26

`blimits` shows both TMP and JOBS limits exceeded in the short queue:

blimits					
INTERNAL RESOURCE LIMITS:					
NAME	USERS	QUEUES	SLOTS	TMP	JOBS
NONAME000	-	normal	-	20/20	1/2
NONAME001	-	short	-	60/20	3/2

Limits for resource consumers

Host groups and compute units

If a limit is specified for a host group or compute unit, the total amount of a resource used by all hosts in that group or unit is counted. If a host is a member of more than one group, each job running on that host is counted against the limit for all groups to which the host belongs.

Limits for users and user groups

Jobs are normally queued on a first-come, first-served (FCFS) basis. It is possible for some users to abuse the system by submitting a large number of jobs; jobs from other users must wait until these jobs complete. Limiting resources by user prevents users from monopolizing all the resources.

Users can submit an unlimited number of jobs, but if they have reached their limit for any resource, the rest of their jobs stay pending, until some of their running jobs finish or resources become available.

If a limit is specified for a user group, the total amount of a resource used by all users in that group is counted. If a user is a member of more than one group, each of that user's jobs is counted against the limit for all groups to which that user belongs.

Use the keyword `all` to configure limits that apply to each user or user group in a cluster. This is useful if you have a large cluster but only want to exclude a few users from the limit definition.

You can use **ENFORCE_ONE_UG_LIMITS=Y** combined with `bsub -G` to have better control over limits when user groups have overlapping members. When set to Y, only the specified user group's limits (or those of any parent user group) are enforced. If set to N, the most restrictive job limits of any overlapping user/user group are enforced.

Per-user limits on users and groups

Per-user limits are enforced on each user or individually to each user in the user group listed. If a user group contains a subgroup, the limit also applies to each member in the subgroup recursively.

Per-user limits that use the keywords `all` apply to each user in a cluster. If user groups are configured, the limit applies to each member of the user group, not the group as a whole.

Resizable jobs

When a resize allocation request is scheduled for a resizable job, all resource allocation limits (job and slot) are enforced. Once the new allocation is satisfied, it consumes limits such as SLOTS, MEM, SWAP and TMP for queues, users, projects, hosts or cluster-wide. However the new allocation will not consume job limits such as job group limits, job array limits, and non-host level JOBS limit.

Releasing part of an allocation from a resizable job frees general limits that belong to the allocation, but not the actual job limits.

Configure resource allocation limits

lsb.resources file

Configure all resource allocation limits in one or more `Limit` sections in the `lsb.resources` file. Limit sections set limits for how much of the specified resources must be available for different classes of jobs to start, and which resource consumers the limits apply to.

The `Limit` section of `lsb.resources` does not support the keywords or format used in `lsb.users`, `lsb.hosts`, and `lsb.queues`. However, any existing job slot limit configuration in these files continues to apply.

Resource parameters

To limit ...	Set in a Limit section of lsb.resources ...
Total number of running and suspended (RUN, SSUSP, USUSP) jobs	JOBS
Total number of job slots that can be used by specific jobs	SLOTS
Jobs slots based on the number of processors on each host affected by the limit	SLOTS_PER_PROCESSOR and PER_HOST
Memory — if PER_HOST is set for the limit, the amount can be a percentage of memory on each host in the limit	MEM (MB or percentage)
Swap space — if PER_HOST is set for the limit, the amount can be a percentage of swap space on each host in the limit	SWP (MB or percentage)
Tmp space — if PER_HOST is set for the limit, the amount can be a percentage of tmp space on each host in the limit	TMP (MB or percentage)
Software licenses	LICENSE or RESOURCE
Any shared resource	RESOURCE

Consumer parameters

For jobs submitted ...	Set in a Limit section of lsb.resources ...
By all specified users or user groups	USERS
To all specified queues	QUEUES
To all specified hosts, host groups, or compute units	HOSTS
For all specified projects	PROJECTS
By each specified user or each member of the specified user groups	PER_USER
To each specified queue	PER_QUEUE
To each specified host or each member of specified host groups or compute units	PER_HOST

For jobs submitted ...	Set in a Limit section of lsb.resources ...
For each specified project	PER_PROJECT

Enable resource allocation limits

- To enable resource allocation limits in your cluster, you configure the resource allocation limits scheduling plugin `schmod_limit` in `lsb.modules`:

```

Begin PluginModule
SCH_PLUGIN          RB_PLUGIN          SCH_DISABLE_PHASES
schmod_default      ()                  ()
schmod_limit        ()                  ()
End PluginModule

```

Configure cluster-wide limits

- To configure limits that take effect for your entire cluster, configure limits in `lsb.resources`, but do not specify any consumers.

How resource allocation limits map to pre-version 7 job slot limits

Job slot limits are the only type of limit you can configure in `lsb.users`, `lsb.hosts`, and `lsb.queues`. You cannot configure limits for user groups, host groups, and projects in `lsb.users`, `lsb.hosts`, and `lsb.queues`. You should not configure any new resource allocation limits in `lsb.users`, `lsb.hosts`, and `lsb.queues`. Use `lsb.resources` to configure all new resource allocation limits, including job slot limits.

Job slot resources	Resource consumers (lsb.resources)					Equivalent existing limit (file)
(lsb.resources)	USERS	PER_USER	QUEUES	HOSTS	PER_HOST	
SLOTS	—	all	—	<i>host_name</i>	—	JL/U (lsb.hosts)
SLOTS_PER_PROCESS OR	<i>user_name</i>	—	—	—	all	JL/P (lsb.users)
SLOTS	—	all	<i>queue_name</i>	—	—	UJOB_LIMIT (lsb.queues)
SLOTS	—	all	—	—	—	MAX_JOBS (lsb.users)
SLOTS	—	—	<i>queue_name</i>	—	all	HJOB_LIMIT (lsb.queues)
SLOTS	—	—	—	<i>host_name</i>	—	MXJ (lsb.hosts)
SLOTS_PER_PROCESS OR	—	—	<i>queue_name</i>	—	all	PJOB_LIMIT (lsb.queues)

Job slot resources	Resource consumers (lsb.resources)					Equivalent existing limit (file)
(lsb.resources)	USERS	PER_USER	QUEUES	HOSTS	PER_HOST	
SLOTS	—	—	queue_name	—	—	QJOB_LIMIT (lsb.queues)

Limits for the following resources have no corresponding limit in lsb. users, lsb. hosts, and lsb. queues:

- JOBS
- LICENSE
- RESOURCE
- SWP
- TMP

Limit conflicts

Similar conflicting limits

For similar limits configured in lsb. resources, lsb. users, lsb. hosts, or lsb. queues, the most restrictive limit is used. For example, a slot limit of 3 for all users is configured in lsb. resources:

```
Begin Limit
NAME = user_limit1
USERS = all
SLOTS = 3
End Limit
```

This is similar, but *not equivalent* to an existing MAX_JOBS limit of 2 is configured in lsb. users.

busers

USER/GROUP	JL/P	MAX	NJOBS	PEND	RUN	SSUSP	USUSP	RSV
user1	-	2	4	2	2	0	0	0

user1 submits 4 jobs:

bjobs

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
816	user1	RUN	normal	hostA	hostA	sleep 1000	Jan 22 16:34
817	user1	RUN	normal	hostA	hostA	sleep 1000	Jan 22 16:34
818	user1	PEND	normal	hostA		sleep 1000	Jan 22 16:34
819	user1	PEND	normal	hostA		sleep 1000	Jan 22 16:34

Two jobs (818 and 819) remain pending because the more restrictive limit of 2 from lsb. users is enforced:

bjobs -p

JOBID	USER	STAT	QUEUE	FROM_HOST	JOB_NAME	SUBMIT_TIME
818	user1	PEND	normal	hostA	sleep 1000	Jan 22 16:34
The user has reached his/her job slot limit;						
819	user1	PEND	normal	hostA	sleep 1000	Jan 22 16:34
The user has reached his/her job slot limit;						

If the MAX_JOBS limit in lsb. users is 4:

busers

USER/GROUP	JL/P	MAX	NJOBS	PEND	RUN	SSUSP	USUSP	RSV
user1	-	4	4	1	3	0	0	0

and user 1 submits 4 jobs:

bjobs							
JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
824	user1	RUN	normal	hostA	hostA	sleep 1000	Jan 22 16: 38
825	user1	RUN	normal	hostA	hostA	sleep 1000	Jan 22 16: 38
826	user1	RUN	normal	hostA	hostA	sleep 1000	Jan 22 16: 38
827	user1	PEND	normal	hostA		sleep 1000	Jan 22 16: 38

Only one job (827) remains pending because the more restrictive limit of 3 in l sb. resources is enforced:

bjobs -p							
JOBID	USER	STAT	QUEUE	FROM_HOST	JOB_NAME	SUBMIT_TIME	
827	user1	PEND	normal	hostA	sleep 1000	Jan 22 16: 38	

Resource (slot) limit defined cluster-wide has been reached;

Equivalent conflicting limits

New limits in l sb. resources that are equivalent to existing limits in l sb. users, l sb. hosts, or l sb. queues, but with a different value override the existing limits. The equivalent limits in l sb. users, l sb. hosts, or l sb. queues are ignored, and the value of the new limit in l sb. resources is used.

For example, a *per-user* job slot limit in l sb. resources is equivalent to a MAX_JOBS limit in l sb. users, so only the l sb. resources limit is enforced, the limit in l sb. users is ignored:

```
Begin Limit
NAME = slot_limit
PER_USER =all
SLOTS = 3
End Limit
```

How job limits work

The JOBS parameter limits the maximum number of running or suspended jobs available to resource consumers. Limits are enforced depending on the number of jobs in RUN, SSUSP, and USUSP state.

Stop and resume jobs

Jobs stopped with bst op, go into USUSP status. LSF includes USUSP jobs in the count of running jobs, so the usage of JOBS limit will not change when you suspend a job.

Resuming a stopped job (bresume) changes job status to SSUSP. The job can enter RUN state, if the JOBS limit has not been exceeded. Lowering the JOBS limit before resuming the job can exceed the JOBS limit, and prevent SSUSP jobs from entering RUN state.

For example, JOBS=5, and 5 jobs are running in the cluster (JOBS has reached 5/5). Normally, the stopped job (in USUSP state) can later be resumed and begin running, returning to RUN state. If you reconfigure the JOBS limit to 4 before resuming the job, the JOBS usage becomes 5/4, and the job cannot run because the JOBS limit has been exceeded.

Preemption

The JOBS limit does not block preemption based on job slots. For example, if JOBS=2, and a host is already running 2 jobs in a preemptable queue, a new preemptive job can preempt a job on that host as long as the preemptive slots can be satisfied even though the JOBS limit has been reached.

Reservation and backfill

Reservation and backfill are still made at the job slot level, but despite a slot reservation being satisfied, the job may ultimately not run because the JOBS limit has been reached. This similar to a job not running because a license is not available.

Other jobs

- `brun` forces a pending job to run immediately on specified hosts. A job forced to run with `brun` is counted as a running job, which may violate JOBS limits. After the forced job starts, the JOBS limits may be exceeded.
- Requeued jobs (`brequeue`) are assigned PEND status or PSUSP. Usage of JOBS limit is decreased by the number of requeued jobs.
- Checkpointed jobs restarted with `brestart` start a new job based on the checkpoint of an existing job. Whether the new job can run depends on the limit policy (including the JOBS limit) that applies to the job. For example, if you checkpoint a job running on a host that has reached its JOBS limit, then restart it, the restarted job cannot run because the JOBS limit has been reached.
- For job arrays, you can define a maximum number of jobs that can run in the array at any given time. The JOBS limit, like other resource allocation limits, works in combination with the array limits. For example, if `JOBS=3` and the array limit is 4, at most 3 job elements can run in the array.
- For chunk jobs, only the running job among the jobs that are dispatched together in a chunk is counted against the JOBS limit. Jobs in WAIT state do not affect the JOBS limit usage.

Example limit configurations

Each set of limits is defined in a `Li mi t` section enclosed by `Begi n Li mi t` and `End Li mi t`.

Example 1

user1 is limited to 2 job slots on hostA, and user2's jobs on queue normal are limited to 20 MB of memory:

```

Begin Li mi t
NAME    HOSTS    SLOTS  MEM   SWP   TMP   USERS    QUEUES
Li mi t 1  hostA    2      -    -    -    user1    -
-        -        -      20    -    -    user2    normal
End Li mi t

```

Example 2

Set a job slot limit of 2 for user user1 submitting jobs to queue normal on host hosta for all projects, but only one job slot for all queues and hosts for project test:

```

Begin Li mi t
HOSTS  SLOTS  PROJECTS  USERS    QUEUES
hosta  2      -          user1    normal
-      1      test       user1    -
End Li mi t

```

Example 3

Limit usage of hosts in `license1` group:

- 10 jobs can run from normal queue
- Any number can run from short queue, but only can use 200 MB of memory in total
- Each other queue can run 30 jobs, each queue using up to 300 MB of memory in total

```

Begin Li mi t
HOSTS    SLOTS  MEM   PER_QUEUE
license1  10      -     normal
license1  -      200   short
license1  30      300   (all ~normal ~short)
End Li mi t

```

Example 4

All users in user group ugroup1 except user1 using queue1 and queue2 and running jobs on hosts in host group hgroup1 are limited to 2 job slots per processor on each host:

```
Begin Limit
NAME      = limit1
# Resources:
SLOTS_PER_PROCESSOR = 2
#Consumers:
QUEUES    = queue1 queue2
USERS     = ugroup1 ~user1
PER_HOST  = hgroup1
End Limit
```

Example 5

user1 and user2 can use all queues and all hosts in the cluster with a limit of 20 MB of available memory:

```
Begin Limit
NAME = 20_MB_mem
# Resources:
MEM = 20
# Consumers:
USERS = user1 user2
End Limit
```

Example 6

All users in user group ugroup1 can use queue1 and queue2 and run jobs on any host in host group hgroup1 sharing 10 job slots:

```
Begin Limit
NAME = 10_slot
# Resources:
SLOTS = 10
#Consumers:
QUEUES = queue1 queue2
USERS  = ugroup1
HOSTS  = hgroup1
End Limit
```

Example 7

All users in user group ugroup1 except user1 can use all queues but queue1 and run jobs with a limit of 10% of available memory on each host in host group hgroup1:

```
Begin Limit
NAME      = 10_percent_mem
# Resources:
MEM       = 10%
QUEUES    = all ~queue1
USERS     = ugroup1 ~user1
PER_HOST  = hgroup1
End Limit
```

Example 8

Limit users in the devel op group to 1 job on each host, and 50% of the memory on the host.

```
Begin Limit
NAME = devel op_group_limit
# Resources:
SLOTS = 1
MEM = 50%
#Consumers:
USERS = devel op
PER_HOST = all
End Limit
```

Example 9

Limit software license `lic1`, with quantity 100, where user 1 can use 90 licenses and all other users are restricted to 10.

```
Begin Limit
USERS      LICENSE
user1      ([lic1, 90])
(all ~user1) ([lic1, 10])
End Limit
```

`lic1` is defined as a decreasing numeric shared resource in `lsf.shared`.

To submit a job to use one `lic1` license, use the `rusage` string in the `-R` option of `bsub` specify the license:

```
bsub -R "rusage[lic1=1]" my-job
```

Example 10

Jobs from crash project can use 10 `lic1` licenses, while jobs from all other projects together can use 5.

```
Begin Limit
LICENSE    PROJECTS
([lic1, 10]) crash
([lic1, 5]) (all ~crash)
End Limit
```

`lic1` is defined as a decreasing numeric shared resource in `lsf.shared`.

Example 11

Limit all hosts to 1 job slot per processor:

```
Begin Limit
NAME              = default_limit
SLOTS_PER_PROCESSOR = 1
PER_HOST          = all
End Limit
```

Example 12

The short queue can have at most 200 running and suspended jobs:

```
Begin Limit
NAME      = shortq_limit
QUEUES    = short
JOBS      = 200
End Limit
```

View information about resource allocation limits

Your job may be pending because some configured resource allocation limit has been reached. Use the `blimits` command to show the dynamic counters of resource allocation limits configured in Limit sections in `lsb.resources`. `blimits` displays the current resource usage to show what limits may be blocking your job.

blimits command

The `blimits` command displays:

- Configured limit policy name
- Users (-u option)
- Queues (-q option)
- Hosts (-m option)
- Project names (-P option)
- Limits (SLOTS, MEM, TMP, SWP, JOBS)
- All resource configurations in `lsb.resources` (-c option). This is the same as `bresources` with no options.

Resources that have no configured limits or no limit usage are indicated by a dash (-). Limits are displayed in a USED/LIMIT format. For example, if a limit of 10 slots is configured and 3 slots are in use, then `blimits` displays the limit for SLOTS as 3/10.

If limits MEM, SWP, or TMP are configured as percentages, both the limit and the amount used are displayed in MB. For example, `lshosts` displays `maxmem` of 249 MB, and MEM is limited to 10% of available memory. If 10 MB out of 25 MB are used, `blimits` displays the limit for MEM as 10/25 (10 MB USED from a 25 MB LIMIT).

Configured limits and resource usage for built-in resources (slots, mem, tmp, and swp load indices, and number of running and suspended jobs) are displayed as INTERNAL RESOURCE LIMITS separately from custom external resources, which are shown as EXTERNAL RESOURCE LIMITS.

Limits are displayed for both the vertical tabular format and the horizontal format for Limit sections. If a vertical format Limit section has no name, `blimits` displays `NONAMEnnn` under the NAME column for these limits, where the unnamed limits are numbered in the order the vertical-format Limit sections appear in the `lsb.resources` file.

If a resource consumer is configured as `all`, the limit usage for that consumer is indicated by a dash (-).

`PER_HOST` slot limits are not displayed. The `bhosts` commands displays these as MXJ limits.

In MultiCluster, `blimits` returns the information about all limits in the local cluster.

Examples

For the following limit definitions:

```
Begin Limit
NAME = limit1
USERS = user1
PER_QUEUE = all
PER_HOST = hostA hostC
TMP = 30%
SWP = 50%
MEM = 10%
End Limit
```

```
Begin Limit
```

```
NAME = limit_ext1
PER_HOST = all
RESOURCE = ([user1_num, 30] [hc_num, 20])
End Limit
```

```
Begin Limit
NAME = limit2
QUEUES = short
JOBS = 200
End Limit
```

blimits displays the following:

blimits

INTERNAL RESOURCE LIMITS:

NAME	USERS	QUEUES	HOSTS	PROJECTS	SLOTS	MEM	TMP	SWP	JOBS
limit1	user1	q2	hostA@cluster1	-	-	10/25	-	10/258	-
limit1	user1	q3	hostA@cluster1	-	-	-	30/2953	-	-
limit1	user1	q4	hostC	-	-	40/590	-	-	-
limit2	-	short	-	-	-	-	-	50/200	-

EXTERNAL RESOURCE LIMITS:

NAME	USERS	QUEUES	HOSTS	PROJECTS	user1_num	hc_num
limit_ext1	-	-	hostA@cluster1	-	-	1/20
limit_ext1	-	-	hostC@cluster1	-	1/30	1/20

- In limit policy limit1, user1 submitting jobs to q2, q3, or q4 on hostA or hostC is limited to 30% tmp space, 50% swap space, and 10% available memory. No limits have been reached, so the jobs from user1 should run. For example, on hostA for jobs from q2, 10 MB of memory are used from a 25 MB limit and 10 MB of swap space are used from a 258 MB limit.
- In limit policy limit_ext1, external resource user1_num is limited to 30 per host and external resource hc_num is limited to 20 per host. Again, no limits have been reached, so the jobs requesting those resources should run.
- In limit policy limit2, the short queue can have at most 200 running and suspended jobs. 50 jobs are running or suspended against the 200 job limit. The limit has not been reached, so jobs can run in the short queue.

Reserving Resources

About resource reservation

When a job is dispatched, the system assumes that the resources that the job consumes will be reflected in the load information. However, many jobs do not consume the resources they require when they first start. Instead, they will typically use the resources over a period of time.

For example, a job requiring 100 MB of swap is dispatched to a host having 150 MB of available swap. The job starts off initially allocating 5 MB and gradually increases the amount consumed to 100 MB over a period of 30 minutes. During this period, another job requiring more than 50 MB of swap should not be started on the same host to avoid over-committing the resource.

Resources can be reserved to prevent overcommitment by LSF. Resource reservation requirements can be specified as part of the resource requirements when submitting a job, or can be configured into the queue level resource requirements.

Pending job resize allocation requests are not supported in slot reservation policies. Newly added or removed resources are reflected in the pending job predicted start time calculation.

Resource reservation limits

Maximum and minimum values for consumable resource requirements can be set for individual queues, so jobs will only be accepted if they have resource requirements within a specified range. This can be useful when queues are configured to run jobs with specific memory requirements, for example. Jobs requesting more memory than the maximum limit for the queue will not be accepted, and will not take memory resources away from the smaller memory jobs the queue is designed to run.

Resource reservation limits are set at the queue level by the parameter `RESRSV_LIMIT` in `lsb.queues`.

How resource reservation works

When deciding whether to schedule a job on a host, LSF considers the reserved resources of jobs that have previously started on that host. For each load index, the amount reserved by all jobs on that host is summed up and subtracted (or added if the index is increasing) from the current value of the resources as reported by the LIM to get amount available for scheduling new jobs:

```
available amount = current value - reserved amount for all jobs
```

For example:

```
bsub -R "rusage[tmp=30:duration=30:decay=1]" myjob
```

will reserve 30 MB of temp space for the job. As the job runs, the amount reserved will decrease at approximately 1 MB/minute such that the reserved amount is 0 after 30 minutes.

Queue-level and job-level resource reservation

The queue level resource requirement parameter `RES_REQ` may also specify the resource reservation. If a queue reserves certain amount of a resource (and the parameter `RESRSV_LIMIT` is not being used), you cannot reserve a greater amount of that resource at the job level.

For example, if the output of `bqueues -l` command contains:

```
RES_REQ:  rusage[mem=40: swp=80: tmp=100]
```

the following submission will be rejected since the requested amount of certain resources exceeds queue's specification:

```
bsub -R "rusage[mem=50:swp=100]" myjob
```


When both `RES_REQ` and `RESRSV_LIMIT` are set in `lsb.queues` for a consumable resource, the queue-level `RES_REQ` no longer acts as a hard limit for the merged `RES_REQ` usage values from the job and application levels. In this case only the limits set by `RESRSV_LIMIT` must be satisfied, and the queue-level `RES_REQ` acts as a default value.

Use resource reservation

Queue-level resource reservation

At the queue level, resource reservation allows you to specify the amount of resources to reserve for jobs in the queue. It also serves as the upper limits of resource reservation if a user also specifies it when submitting a job.

Queue-level resource reservation and pending reasons

The use of RES_REQ affects the pending reasons as displayed by `bj obs`. If RES_REQ is specified in the queue and the `loadSched` thresholds are not specified, then the pending reasons for each individual load index will not be displayed.

Configure resource reservation at the queue level

Queue-level resource reservations and resource reservation limits can be configured as parameters in `lsb. queues`.

1. Specify the amount of resources a job should reserve after it is started in the resource usage (`rusage`) section of the resource requirement string of the QUEUE section.

Examples

```
Begin Queue
RES_REQ = select [type==any] rusage[swp=100: mem=40: duration=60]
RESRSV_LIMIT = [mem=30, 100]
End Queue
```

This allows a job to be scheduled on any host that the queue is configured to use and reserves 100 MB of swap and 40 MB of memory for a duration of 60 minutes. The requested memory reservation of 40 MB falls inside the allowed limits set by RESRSV_LIMIT of 30 MB to 100 MB.

```
Begin Queue
RES_REQ = select [type==any] rusage[mem=20 | mem=10: swp=20]
End Queue
```

This allows a job to be scheduled on any host that the queue is configured to use. The job attempts to reserve 20 MB of memory, or 10 MB of memory and 20 MB of swap if the 20 MB of memory is unavailable. In this case no limits are defined by RESRSV_LIMIT.

Specify job-level resource reservation

1. To specify resource reservation at the job level, use `bsub -R` and include the resource usage section in the resource requirement string.

Configure per-resource reservation

1. To enable greater flexibility for reserving numeric resources are reserved by jobs, configure the `ReservationUsage` section in `lsb. resources` to reserve resources like license tokens per resource as `PER_JOB`, `PER_SLOT`, or `PER_HOST`:

```

Begin Reservati onUsage
RESOURCE          METHOD
licenseX          PER_JOB
licenseY          PER_HOST
licenseZ          PER_SLOT
End Reservati onUsage

```

Only user-defined numeric resources can be reserved. Builtin resources like mem, cpu, swp, etc. cannot be configured in the Reservati onUsage section.

The cluster-wide RESOURCE_RESERVE_PER_SLOT parameter in l sb. params is obsolete. Configuration in l sb. resources overrides RESOURCE_RESERVE_PER_SLOT if it also exists for the same resource.

RESOURCE_RESERVE_PER_SLOT parameter still controls resources not configured in l sb. resources. Resources not reserved in l sb. resources are reserved per job.

PER_HOST reservation means that for the parallel job, LSF reserves one instance of a for each host. For example, some application licenses are charged only once no matter how many applications are running provided those applications are running on the same host under the same user.

Assumptions and limitations

- Per-resource configuration defines resource usage for individual resources, but it does not change any existing resource limit behavior (PER_JOB, PER_SLOT).
- In a MultiCluster environment, you should configure resource usage in the scheduling cluster (submission cluster in lease model or receiving cluster in job forward model).
- The keyword pref in the compute unit resource string is ignored, and the default configuration order is used (pref=confi g).

Memory reservation for pending jobs

By default, the `rusage` string reserves resources for running jobs. Because resources are not reserved for pending jobs, some memory-intensive jobs could be pending indefinitely because smaller jobs take the resources immediately before the larger jobs can start running. The more memory a job requires, the worse the problem is.

Memory reservation for pending jobs solves this problem by reserving memory as it becomes available, until the total required memory specified on the `rusage` string is accumulated and the job can start. Use memory reservation for pending jobs if memory-intensive jobs often compete for memory with smaller jobs in your cluster.

Reserve host memory for pending jobs

1. Use the `RESOURCE_RESERVE` parameter in `lsb.queues` to reserve host memory for pending jobs.

The amount of memory reserved is based on the currently available memory when the job is pending. Reserved memory expires at the end of the time period represented by the number of dispatch cycles specified by the value of `MAX_RESERVE_TIME` set on the `RESOURCE_RESERVE` parameter.

Enable memory reservation for sequential jobs

1. Add the LSF scheduler plugin module name for resource reservation (`schmod_reserve`) to the `lsb.modules` file:

<code>Begin PluginModule</code>		
<code>SCH_PLUGIN</code>	<code>RB_PLUGIN</code>	<code>SCH_DISPATCHABLE_PHASES</code>
<code>schmod_default</code>	<code>()</code>	<code>()</code>
<code>schmod_reserve</code>	<code>()</code>	<code>()</code>
<code>schmod_preemption</code>	<code>()</code>	<code>()</code>
<code>End PluginModule</code>		

Configure `lsb.queues`

1. Set the `RESOURCE_RESERVE` parameter in a queue defined in `lsb.queues`.

If both `RESOURCE_RESERVE` and `SLOT_RESERVE` are defined in the same queue, job slot reservation and memory reservation are both enabled and an error is displayed when the cluster is reconfigured. `SLOT_RESERVE` is ignored.

Example queues

The following queue enables memory reservation for pending jobs:

```
Begin Queue
QUEUE_NAME = reservation
DESCRIPTION = For resource reservation
PRIORITY=40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
End Queue
```

Use memory reservation for pending jobs

1. Use the `rusage` string in the `-R` option to `bsub` or the `RES_REQ` parameter in `lsb.queues` to specify the amount of memory required for the job. Submit the job to a queue with `RESOURCE_RESERVE` configured.

Note:

Compound resource requirements do not support use of the `||` operator within the component `rusage` simple resource requirements, multiple `-R` options, or the `cu` section.

How memory reservation for pending jobs works

Amount of memory reserved

The amount of memory reserved is based on the currently available memory when the job is pending. For example, if LIM reports that a host has 300 MB of memory available, the job submitted by the following command:

```
bsub -R "rusage[mem=400]" -q reservation my_job
```

will be pending and reserve the 300 MB of available memory. As other jobs finish, the memory that becomes available is added to the reserved memory until 400 MB accumulates, and the job starts.

No memory is reserved if no job slots are available for the job because the job could not run anyway, so reserving memory would waste the resource.

Only memory is accumulated while the job is pending; other resources specified on the `rusage` string are only reserved when the job is running. Duration and decay have no effect on memory reservation while the job is pending.

How long memory is reserved (MAX_RESERVE_TIME)

Reserved memory expires at the end of the time period represented by the number of dispatch cycles specified by the value of `MAX_RESERVE_TIME` set on the `RESOURCE_RESERVE` parameter. If a job has not accumulated enough memory to start by the time `MAX_RESERVE_TIME` expires, it releases all its reserved memory so that other pending jobs can run. After the reservation time expires, the job cannot reserve slots or memory for one scheduling session, so other jobs have a chance to be dispatched. After one scheduling session, the job can reserve available resources again for another period specified by `MAX_RESERVE_TIME`.

Examples

lsb.queues

The following queues are defined in `lsb.queues`:

```
Begin Queue
QUEUE_NAME = reservation
DESCRIPTION = For resource reservation
PRIORITY=40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
End Queue
```

Assumptions

Assume one host in the cluster with 10 CPUs and 1 GB of free memory currently available.

Sequential jobs

Each of the following sequential jobs requires 400 MB of memory and runs for 300 minutes.

Job 1:

```
bsub -W 300 -R "rusage[mem=400]" -q reservation myjob1
```

The job starts running, using 400M of memory and one job slot.

Job 2:

Submitting a second job with same requirements yields the same result.

Job 3:

Submitting a third job with same requirements reserves one job slot, and reserves all free memory, if the amount of free memory is between 20 MB and 200 MB (some free memory may be used by the operating system or other software.)

Time-based slot reservation

Existing LSF slot reservation works in simple environments, where the host-based MXJ limit is the only constraint to job slot request. In complex environments, where more than one constraint exists (for example job topology or generic slot limit):

- Estimated job start time becomes inaccurate
- The scheduler makes a reservation decision that can postpone estimated job start time or decrease cluster utilization.

Current slot reservation by start time (RESERVE_BY_STARTTIME) resolves several reservation issues in multiple candidate host groups, but it cannot help on other cases:

- Special topology requests, like `span[pt i l e=n]` and `cu[]` keywords `balance`, `maxcus`, and `excl`.
- Only calculates and displays reservation if host has free slots. Reservations may change or disappear if there are no free CPUs; for example, if a backfill job takes all reserved CPUs.
- For HPC machines containing many internal nodes, host-level number of reserved slots is not enough for administrator and end user to tell which CPUs the job is reserving and waiting for.

Time-based slot reservation versus greedy slot reservation

With time-based reservation, a set of pending jobs get future allocation and an estimated start time so that the system can reserve a place for each job. Reservations use the estimated start time, which is based on future allocations.

Time-based resource reservation provides a more accurate predicted start time for pending jobs because LSF considers job scheduling constraints and requirements, including job topology and resource limits, for example.

Restriction:

Time-based reservation does not work with job chunking.

Start time and future allocation

The estimated start time for a future allocation is the earliest start time when all considered job constraints are satisfied in the future. There may be a small delay of a few minutes between the job finish time on which the estimate was based and the actual start time of the allocated job.

For compound resource requirement strings, the predicted start time is based on the simple resource requirement term (contained in the compound resource requirement) with the latest predicted start time.

If a job cannot be placed in a future allocation, the scheduler uses *greedy* slot reservation to reserve slots. Existing LSF slot reservation is a simple greedy algorithm:

- Only considers current available resources and minimal number of requested job slots to reserve as many slots as it is allowed
- For multiple exclusive candidate host groups, scheduler goes through those groups and makes reservation on the group that has the largest available slots
- For estimated start time, after making reservation, scheduler sorts all running jobs in ascending order based on their finish time and goes through this sorted job list to add up slots used by each running job till it satisfies minimal job slots request. The finish time of last visited job will be job estimated start time.

Reservation decisions made by greedy slot reservation do not have an accurate estimated start time or information about future allocation. The calculated job start time used for backfill scheduling is uncertain, so `bjobs` displays:

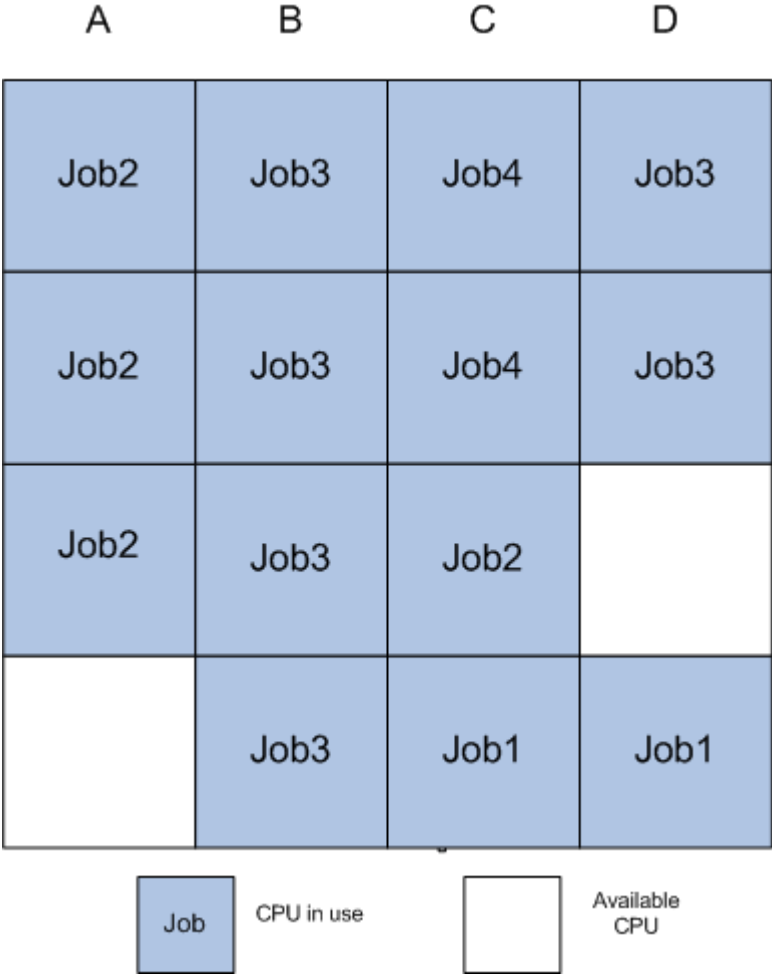
```
Job will start no sooner than indicated time stamp
```

Time-based reservation and greedy reservation compared

Start time prediction	Time-based reservation	Greedy reservation
Backfill scheduling if free slots are available	Yes	Yes
Correct with no job topology	Yes	Yes
Correct for job topology requests	Yes	No
Correct based on resource allocation limits	Yes (guaranteed if only two limits are defined)	No
Correct for memory requests	Yes	No
When no slots are free for reservation	Yes	No
Future allocation and reservation based on earliest start time	Yes	No
<code>bjobs</code> displays best estimate	Yes	No
<code>bjobs</code> displays predicted future allocation	Yes	No
Absolute predicted start time for all jobs	No	No
Advance reservation considered	No	No

Greedy reservation example

A cluster has four hosts: A, B, C and D, with 4 CPUs each. Four jobs are running in the cluster: Job1, Job2, Job3 and Job4. According to calculated job estimated start time, the job finish times (FT) have this order: $FT(\text{Job2}) < FT(\text{Job1}) < FT(\text{Job4}) < FT(\text{Job3})$.



Now, a user submits a high priority job. It pends because it requests `-n 6 -R "span[ptile=2]"`. This resource requirement means this pending job needs three hosts with two CPUs on each host. The default greedy slot reservation calculates job start time as the job finish time of Job4 because after Job4 finishes, three hosts with a minimum of two slots are available. Greedy reservation indicates that the pending job starts no sooner than when Job 2 finishes. In contrast, time-based reservation can determine that the pending job starts in 2 hours. It is a much more accurate reservation.

Configure time-based slot reservation

Greedy slot reservation is the default slot reservation mechanism and time-based slot reservation is disabled.

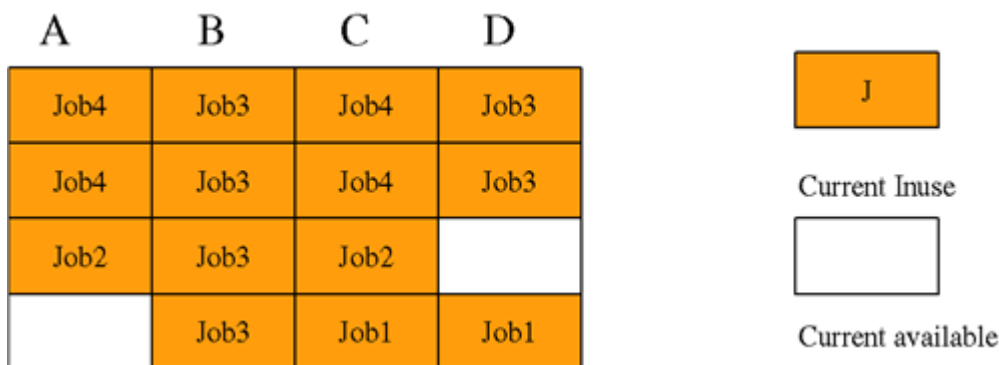
1. Use `LSB_TIME_RESERVE_NUMJOBS=maximum_reservation_jobs` in `lsf.conf` to enable time-based slot reservation. The value must be a positive integer.
LSB_TIME_RESERVE_NUMJOBS controls maximum number of jobs using time-based slot reservation. For example, if `LSB_TIME_RESERVE_NUMJOBS=4`, only the top 4 jobs will get their future allocation information.

2. Use `LSB_TIME_RESERVE_NUMJOBS=1` to allow only the highest priority job to get accurate start time prediction.

Smaller values are better than larger values because after the first pending job starts, the estimated start time of remaining jobs may be changed. For example, you could configure `LSB_TIME_RESERVE_NUMJOBS` based on the number of exclusive host partitions or host groups.

Scheduling examples

1. Job5 requests `-n 6 -R "span[ptile=2]"`, which will require three hosts with 2 CPUs on each host. As in the greedy slot reservation example, four jobs are running in the cluster: Job1, Job2, Job3 and Job4. Two CPUs are available now, 1 on host A, and 1 on host D:



2. Job2 finishes, freeing 2 more CPUs for future allocation, 1 on host A, and 1 on host C:



3. Job4 finishes, freeing 4 more CPUs for future allocation, 2 on host A, and 2 on host C:



4. Job1 finishes, freeing 2 more CPUs for future allocation, 1 on host C, and 1 host D:

A	B	C	D		
	Job3		Job3		Future available
	Job3		Job3		
	Job3				
	Job3				

5. Job5 can now be placed with 2 CPUs on host A, 2 CPUs on host C, and 2 CPUs on host D. The estimated start time is shown as the finish time of Job1:

A	B	C	D		
	Job3		Job3		Future allocation
	Job3		Job3		
Job5	Job3	Job5	Job5		
Job5	Job3	Job5	Job5		

Assumptions and limitations

- To get an accurate estimated start time, you must specify a run limit at the job level using the `bsub -W option`, in the queue by configuring `RUNLIMIT` in `lsb.queues`, or in the application by configuring `RUNLIMIT` in `lsb.applications`, or you must specify a run time estimate by defining the `RUNTIME` parameter in `lsb.applications`. If a run limit or a run time estimate is not defined, the scheduler will try to use CPU limit instead.
- Estimated start time is only relatively accurate according to current running job information. If running jobs finish earlier, estimated start time may be moved to earlier time. Only the highest priority job will get accurate predicted start time. The estimated start time for other jobs could be changed after the first job starts.
- Under time-based slot reservation, only information from currently running jobs is used for making reservation decisions.
- Estimated start time calculation does not consider Deadline scheduling.
- Estimated start time calculation does not consider Advance Reservation.
- Estimated start time calculation does not consider `DISPATCH_WINDOW` in `lsb.hosts` and `lsb.queue` configuration.
- If preemptive scheduling is used, the estimated start time may not be accurate. The scheduler may calculate an estimated time, but actually it may preempt other jobs to start earlier.
- For resizable jobs, time-based slot reservation does not schedule pending resize allocation requests. However, for resized running jobs, the allocation change is used when calculating pending job predicted start time and resource reservation. For example, if a running job uses 4 slots at the beginning, but added another 4 slots, after adding the new resources, LSF expects 8 slots to be available after the running job completes.

Slot limit enforcement

The following slot limits are enforced:

- Slot limits configured in `l sb. resources (SLOTS, PER_SLOT)`
- `MXJ, JL/U` in `l sb. hosts`
- `PJOB_LIMIT, HJOB_LIMIT, QJOB_LIMIT, UJOB_LIMIT` in `l sb. queues`

Memory request

To request memory resources, configure `RESOURCE_RESERVE` in `l sb. queues`.

When `RESOURCE_RESERVE` is used, LSF will consider memory and slot requests during time-based reservation calculation. LSF will not reserve slot or memory if any other resources are not satisfied.

If `SLOT_RESERVE` is configured, time-based reservation will not make a slot reservation if any other type of resource is not satisfied, including memory requests.

When `SLOT_RESERVE` is used, if job cannot run because of non-slot resources, including memory, time-based reservation will not reserve slots. For example, if job cannot run because it cannot get required license, job will be pending without any reservation

Host partition and queue-level scheduling

If host partitions are configured, LSF first schedules jobs on the host partitions and then goes through each queue to schedule jobs. The same job may be scheduled several times, one for each host partition and last one at queue-level. Available candidate hosts may be different for each time.

Because of this difference, the same job may get different estimated start times, future allocation, and reservation in different host partitions and queue-level scheduling. With time-based reservation configured, LSF always keeps the same reservation and future allocation with the earliest estimated start time.

bjobs displays future allocation information

- By default, job future allocation contains LSF host list and number of CPUs per host, for example:
`all oc=2*hostA 3*hostB`
- LSF integrations define their own future allocation string to override the default LSF allocation. For example, in `cpuset`, future allocation is displayed as:

```
all oc=2*mstatx01 2*mstatx00
```

Predicted start time may be postponed for some jobs

If a pending job cannot be placed in a future resource allocation, the scheduler can skip it in the start time reservation calculation and fall back to use greedy slot reservation. There are two possible reasons:

- The job slot request cannot be satisfied in the future allocation
- Other non-slot resources cannot be satisfied.

Either way, the scheduler continues calculating predicted start time for the remaining jobs without considering the skipped job.

Later, once the resource request of skipped job can be satisfied and placed in a future allocation, the scheduler reevaluates the predicted start time for the rest of jobs, which may potentially postpone their start times.

To minimize the overhead in recalculating the predicted start times to include previously skipped jobs, you should configure a small value for `LSB_TIME_RESERVE_NUMJOBS` in `l sf. conf`.

Reservation scenarios

Scenario 1

Even though no running jobs finish and no host status in cluster are changed, a job's future allocation may still change from time to time.

Why this happens

Each scheduling cycle, the scheduler recalculates a job's reservation information, estimated start time and opportunity for future allocation. The job candidate host list may be reordered according to current load. This reordered candidate host list will be used for the entire scheduling cycle, also including job future allocation calculation. So different order of candidate hosts may lead to different result of job future allocation. However, the job estimated start time should be the same.

For example, there are two hosts in cluster, hostA and hostB. 4 CPUs per host. Job 1 is running and occupying 2 CPUs on hostA and 2 CPUs on hostB. Job 2 requests 6 CPUs. If the order of hosts is hostA and hostB, then the future allocation of job 2 will be 4 CPUs on hostA 2 CPUs on hostB. If the order of hosts changes in the next scheduling cycle changes to hostB and hostA, then the future allocation of job 2 will be 4 CPUs on hostB 2 CPUs on hostA.

Scenario 2:

If you set JOB_ACCEPT_INTERVAL to non-zero value, after job is dispatched, within JOB_ACCEPT_INTERVAL period, pending job estimated start time and future allocation may momentarily fluctuate.

Why this happens

The scheduler does a time-based reservation calculation each cycle. If JOB_ACCEPT_INTERVAL is set to non-zero value, once a new job has been dispatched to a host, this host will not accept new job within JOB_ACCEPT_INTERVAL interval. Because the host will not be considered for the entire scheduling cycle, no time-based reservation calculation is done, which may result in slight change in job estimated start time and future allocation information. After JOB_ACCEPT_INTERVAL has passed, host will become available for time-based reservation calculation again, and the pending job estimated start time and future allocation will be accurate again.

Examples

Example 1

Three hosts, 4 CPUs each: qat24, qat25, and qat26. Job 11895 uses 4 slots on qat24 (10 hours). Job 11896 uses 4 slots on qat25 (12 hours), and job 11897 uses 2 slots on qat26 (9 hours).

Job 11898 is submitted and requests -n 6 -R "span[ptile=2]".

bjobs -l 11898

```
Job <11898>, User <user2>, Project <default>, Status <PEND>, Queue <challenge>,
      Job Priority <50>, Command <sleep 100000000>
```

```
..
RUNLIMIT
```

```
840.0 min of hostA
```

```
Fri Apr 22 15:18:56 2010: Reserved <2> job slots on host(s) <2*qat26>;
```

```
Sat Apr 23 03:28:46 2010: Estimated Job Start Time;
```

```
alloc=2*qat25 2*qat24 2*qat26.lsf.platform.com
```

Example 2

Two cpuset hosts, mstatx00 and mstatx01, 8 CPUs per host. Job 3873 uses 4*mstatx00 and will last for 10 hours. Job 3874 uses 4*mstatx01 and will run for 12 hours. Job 3875 uses 2*mstatx02 and 2*mstatx03, and will run for 13 hours.

Job 3876 is submitted and requests -n 4 -ext "cpuset[nodes=2]" -R "rusage[mem=100] span[ptile= 2]".

bjobs -l 3876

```
Job <3876>, User <user2>, Project <default>, Status <PEND>, Queue <sq32_s>, Command <sleep 33333>
Tue Dec 22 04:56:54: Submitted from host <mstatx00>, CWD <$HOME>, 4 Processors Requested,
Requested Resources <rusage[mem=100] span[ptile= 2]>;
RUNLIMIT
60.0 min of mstatx00 Tue Dec 22 06:07:38: Estimated job start time; alloc=2*mstatx01 2*mstatx00 ...
```

Example 3

Rerun example 1, but this time, use greedy slot reservation instead of time-based reservation:

bjobs -l 3876

```
Job <12103>, User <user2>, Project <default>, Status <PEND>, Queue <challenge>,
Job Priority <50>, Command <sleep 1000000>
Fri Apr 22 16:17:59 2010: Submitted from host <qat26>, CWD <$HOME>, 6 Processors Req
uested, Requested Resources <span[ptile=2]>;
RUNLIMIT
720.0 min of qat26
Fri Apr 22 16:18:09 2010: Reserved <2> job slots on host(s)
<2*qat26.lsf.platform.com>;
Sat Apr 23 01:39:13 2010: Job will start no sooner than indicated time stamp;
```

View resource reservation information

View host-level resource information (bhosts)

1. Use `bhosts -l` to show the amount of resources reserved on each host. In the following example, 143 MB of memory is reserved on hostA, and no memory is currently available on the host.

```
bhosts -l hostA
HOST hostA
STATUS      CPUF    JL/U    MAX    NJOBS    RUN    SSUSP    USUSP    RSV    DISPATCH_WINDOW
ok          20.00    -       4       2        1       0        0        1       -
CURRENT LOAD USED FOR SCHEDULING:
           r15s   r1m   r15m   ut    pg    io    ls    it    tmp    swp
mem
Total      1.5    1.2   2.0   91%   2.5    7    49    0   911M   915M
OM
Reserved   0.0    0.0   0.0   0%    0.0    0    0    0    0M     0M
143M
```

2. Use `bhosts -s` to view information about shared resources.

View queue-level resource information (bqueues)

1. Use `bqueues -l` to see the resource usage configured at the queue level.

```
bqueues -l reservation
QUEUE: reservation
-- For resource reservation

PARAMETERS/STATISTICS
PRIO NICE STATUS      MAX JL/U JL/P JL/H NJOBS  PEND  RUN SSUSP USUSP  RSV
40    0 Open: Active    -  -  -  -  -    4    0    0    0    0    4

SCHEDULING PARAMETERS
loadSched  -      -      -      -      -      -      -      -      -      -      -
loadStop   -      -      -      -      -      -      -      -      -      -      -

           cpuspeed    bandwidth
loadSched   -          -
loadStop    -          -

SCHEDULING POLICIES:  RESOURCE_RESERVE

USERS:  all users
HOSTS:  all
Maximum resource reservation time: 600 seconds
```

View reserved memory for pending jobs (bjobs)

If the job memory requirements cannot be satisfied, `bjobs -l` shows the pending reason. `bjobs -l` shows both reserved slots and reserved memory.

1. For example, the following job reserves 60 MB of memory on hostA:

```
bsub -m hostA -n 2 -q reservation -R"rusage[mem=60]" sleep 8888
Job <3> is submitted to queue <reservation>.
```

`bjobs -l` shows the reserved memory:

```
bjobs -lp
Job <3>, User <user1>, Project <default>, Status <PEND>, Queue <reservation>
```

```
, Command <sleep 8888>
Tue Jan 22 17:01:05 2010: Submitted from host <user1>, CWD </home/user1/>, 2 Processors
Requested, Requested Resources <rusage[mem=60]>, Specified Hosts <hostA>;
Tue Jan 22 17:01:15 2010: Reserved <1> job slot on host <hostA>;
Tue Jan 22 17:01:15 2010: Reserved <60> megabyte memory on host <60M*hostA>;
PENDING REASONS: Not enough job slot(s): hostA;
```

```
SCHEDULING PARAMETERS
r15s  r1m  r15m  ut    pg    io    ls    it    tmp    swp    mem
loadSched -    -    -    -    -    -    -    -    -    -    -
loadStop  -    -    -    -    -    -    -    -    -    -    -

          cpuspeed    bandwidth
loadSched -    -
loadStop  -    -
```

View per-resource reservation (bresources)

1. Use `bresources` to display per-resource reservation configurations from `lsb. resources`:

The following example displays all resource reservation configurations:

```
bresources -s
Begin ReservationUsage
RESOURCE          METHOD
licenseX          PER_JOB
licenseY          PER_HOST
licenseZ          PER_SLOT
End ReservationUsage
```

The following example displays only `licenseZ` configuration:

```
bresources -s licenseZ
RESOURCE          METHOD
licenseZ          PER_SLOT
```


Job Dependency and Job Priority

Job dependency terminology

- Job dependency: The start of a job depends on the state of other jobs.
- Parent jobs: Jobs that other jobs depend on.
- Child jobs: Jobs that cannot start until other jobs have reached a specific state.

Example: If job2 depends on job1 (meaning that job2 cannot start until job1 reaches a specific state), then job2 is the child job and job1 is the parent job.

Job dependency scheduling

About job dependency scheduling

Sometimes, whether a job should start depends on the result of another job. For example, a series of jobs could process input data, run a simulation, generate images based on the simulation output, and finally, record the images on a high-resolution film output device. Each step can only be performed after the previous step finishes successfully, and all subsequent steps must be aborted if any step fails.

Some jobs may not be considered complete until some post-job processing is performed. For example, a job may need to exit from a post-execution job script, clean up job files, or transfer job output after the job completes.

In LSF, any job can be dependent on other LSF jobs. When you submit a job, you use `bsub -w` to specify a dependency expression, usually based on the job states of preceding jobs.

LSF will not place your job unless this dependency expression evaluates to TRUE. If you specify a dependency on a job that LSF cannot find (such as a job that has not yet been submitted), your job submission fails.

Syntax

```
bsub -w 'dependency_expression'
```

The dependency expression is a logical expression composed of one or more dependency conditions.

- To make dependency expression of multiple conditions, use the following logical operators:
 - && (AND)
 - || (OR)
 - ! (NOT)
- Use parentheses to indicate the order of operations, if necessary.
- Enclose the dependency expression in single quotes (') to prevent the shell from interpreting special characters (space, any logic operator, or parentheses). If you use single quotes for the dependency expression, use double quotes for quoted items within it, such as job names.
- Job names specify only your own jobs, unless you are an LSF administrator.
- Use double quotes (") around job names that begin with a number.
- In Windows, enclose the dependency expression in double quotes (") when the expression contains a space. For example:
 - **bsub -w "exit(678, 0)"** requires double quotes in Windows.
 - **bsub -w 'exit(678,0)'** can use single quotes in Windows.
- In the job name, specify the wildcard character (*) at the end of a string, to indicate all jobs whose name begins with the string. For example, if you use `j obA*` as the job name, it specifies jobs named `j obA`, `j obA1`, `j obA_test`, `j obA. log`, etc.

Note:

Wildcard characters can only be used at the end of job name strings within the job dependency expression.

Multiple jobs with the same name

By default, if you use the job name to specify a dependency condition, and more than one of your jobs has the same name, all of your jobs that have that name must satisfy the test.

To change this behavior, set `JOB_DEP_LAST_SUB` in `lsb.params` to 1. Then, if more than one of your jobs has the same name, the test is done on the one submitted most recently.

Specify a job dependency

1. To specify job dependencies, use `bsub -w` to specify a dependency expression for the job.

Dependency conditions

The following dependency conditions can be used with any job:

- `done(job_ID | "job_name")`
- `ended(job_ID | "job_name")`
- `exit(job_ID [, [op] exit_code])`
- `exit("job_name" [, [op] exit_code])`
- `external(job_ID | "job_name", "status_text")`
- `job_ID | "job_name"`
- `post_done(job_ID | "job_name")`
- `post_err(job_ID | "job_name")`
- `started(job_ID | "job_name")`

done

Syntax

```
done(job_ID | "job_name")
```

Description

The job state is DONE.

ended

Syntax

```
ended(job_ID | "job_name")
```

Description

The job state is EXIT or DONE.

exit

Syntax

```
exit(job_ID | "job_name" [, [operator] exit_code])
```

where *operator* represents one of the following relational operators:

>
>=
<
<=
==
!=

Description

The job state is EXIT, and the job's exit code satisfies the comparison test.
If you specify an exit code with no operator, the test is for equality (== is assumed).
If you specify only the job, any exit code satisfies the test.

Examples

```
exit (myjob)
```

The job named `myjob` is in the EXIT state, and it does not matter what its exit code was.

```
exit (678, 0)
```

The job with job ID 678 is in the EXIT state, and terminated with exit code 0.

```
exit ("678", !=0)
```

The job named 678 is in the EXIT state, and terminated with any non-zero exit code.

external

Syntax

```
external(job_ID | "job_name", "status_text")
```

Specify the first word of the job status or message description (no spaces). Only the first word is evaluated.

Description

The job has the specified job status, or the text of the job's status begins with the specified word.

Job ID or job name

Syntax

```
job_ID | "job_name"
```

Description

If you specify a job without a dependency condition, the test is for the DONE state (LSF assumes the "done" dependency condition by default).

post_done

Syntax

post_done(*job_ID* | "*job_name*")

Description

The job state is POST_DONE (the post-processing of specified job has completed without errors).

post_err

Syntax

post_err(*job_ID* | "*job_name*")

Description

The job state is POST_ERR (the post-processing of specified job has completed with errors).

started

Syntax

started(*job_ID* | "*job_name*")

Description

The job state is:

- USUSP, SSUSP, DONE, or EXIT
- RUN and the job has a pre-execution command (`bsub -E`) that is done.

Advanced dependency conditions

If you use job arrays, you can specify additional dependency conditions that only work with job arrays.

To use other dependency conditions with array jobs, specify elements of a job array in the usual way.

Job dependency examples

```
bsub -J "JobA" -w 'done(JobB)' command
```

The simplest kind of dependency expression consists of only one dependency condition. For example, if JobA depends on the successful completion of JobB, submit the job as shown.

```
-w 'done(312) && (started(Job2) || exit("99Job"))'
```

The submitted job will not start until the job with the job ID of 312 has completed successfully, and either the job named Job2 has started, or the job named 99Job has terminated abnormally.

```
-w "210"
```

The submitted job will not start unless the job named 210 is finished.

View job dependencies

The `bj depinfo` command displays any dependencies that jobs have, either jobs that depend on a job or jobs that your job depends on.

By specifying `-r`, you get not only direct dependencies (job A depends on job B), but also indirect dependencies (job A depends on job B, job B depends on jobs C and D). You can also limit the number of levels returned using the `-r` option.

The `-l` option displays results in greater detail.

- To display all jobs that this job depends on:

bjdepinfo 123

JOBID	PARENT	PARENT_STATUS	PARENT_NAME	LEVEL
123	32522	RUN	JOB32522	1

- To display jobs that depend on a job you specify (display child jobs):

bjdepinfo -c 300

JOBID	CHILD	CHILD_STATUS	CHILD_NAME	LEVEL
300	310	PEND	JOB310	1
300	311	PEND	JOB311	1
300	312	PEND	JOB312	1

- To display the parent jobs that cause a job to pend:

bjdepinfo -p 100

These jobs are always pending because their dependency has not yet been satisfied.

JOBID	PARENT	PARENT_STATUS	PARENT_NAME	LEVEL
100	99	PEND	JOB99	1
100	98	PEND	JOB98	1
100	97	PEND	JOB97	1
100	30	PEND	JOB30	1

- Display more information about job dependencies including whether the condition has been satisfied or not and the condition that is on the job:

bjdepinfo -l 32522

Dependency condition of job <32522> is not satisfied: done(23455)

JOBID	PARENT	PARENT_STATUS	PARENT_NAME	LEVEL
32522	23455	RUN	JOB23455	1

- Display information about job dependencies that includes only direct dependencies and two levels of indirect dependencies:

bjdepinfo -r 3 -l 100

Dependency condition of job <100> is not satisfied: done(99) && ended(98) && done(97) && done(96)

JOBID	PARENT	PARENT_STATUS	PARENT_NAME	LEVEL
-------	--------	---------------	-------------	-------

100	99	PEND	JOB99	1
100	98	PEND	JOB98	1
100	97	PEND	JOB97	1
100	96	DONE	JOB96	1

Dependency condition of job <97> is not satisfied: done(89)

JOBID	PARENT	PARENT_STATUS	PARENT_NAME	LEVEL
-------	--------	---------------	-------------	-------

97	89	PEND	JOB89	2
----	----	------	-------	---

Dependency condition of job <89> is not satisfied: ended(86)

JOBID	PARENT	PARENT_STATUS	PARENT_NAME	LEVEL
-------	--------	---------------	-------------	-------

89	86	PEND	JOB86	3
----	----	------	-------	---

Job priorities

User-assigned job priority

User-assigned job priority provides controls that allow users to order their jobs with the jobs of other users in a queue. Job order is the first consideration to determine job eligibility for dispatch. Jobs are still subject to all scheduling policies regardless of job priority. Jobs with the same priority are ordered first come first served.

The job owner can change the priority of their own jobs. LSF and queue administrators can change the priority of all jobs in a queue.

User-assigned job priority is enabled for all queues in your cluster, and can be configured with automatic job priority escalation to automatically increase the priority of jobs that have been pending for a specified period of time.

Considerations

The `bt op` and `bbot` commands move jobs relative to other jobs of the same priority. These commands do not change job priority.

Configure job priority

1. To configure user-assigned job priority edit `lsb.params` and define `MAX_USER_PRIORITY`. This configuration applies to all queues in your cluster.

```
MAX_USER_PRIORITY=max_priority
```

Where:

max_priority

Specifies the maximum priority a user can assign to a job. Valid values are positive integers. Larger values represent higher priority; 1 is the lowest.

LSF and queue administrators can assign priority beyond *max_priority* for jobs they own.

2. Use `bparams -l` to display the value of `MAX_USER_PRIORITY`.

Example

```
MAX_USER_PRIORITY=100
```

Specifies that 100 is the maximum job priority that can be specified by a user.

Specify job priority

- Job priority is specified at submission using `bsub` and modified after submission using `bmod`. Jobs submitted without a priority are assigned the default priority of `MAX_USER_PRIORITY/2`.

```
bsub -sp priority bmod [-sp priority | -spn] job_ID
```

Where:

-sp priority

Specifies the job priority. Valid values for *priority* are any integers between 1 and `MAX_USER_PRIORITY` (displayed by `bparams -l`). Incorrect job priorities are rejected.

LSF and queue administrators can specify priorities beyond MAX_USER_PRIORITY for jobs they own.

-spn

Sets the job priority to the default priority of MAX_USER_PRIORITY/2 (displayed by bparams -l).

View job priority information

1. Use the following commands to view job history, the current status and system configurations:

- `bhist -l job_ID`

Displays the history of a job including changes in job priority.

- `bjobs -l [job_ID]`

Displays the current job priority and the job priority at submission time. Job priorities are changed by the job owner, LSF and queue administrators, and automatically when automatic job priority escalation is enabled.

- `bparams -l`

Displays values for:

- The maximum user priority, MAX_USER_PRIORITY
- The default submission priority, MAX_USER_PRIORITY/2
- The value and frequency used for automatic job priority escalation, JOB_PRIORITY_OVER_TIME

Automatic job priority escalation

Automatic job priority escalation automatically increases job priority of jobs that have been pending for a specified period of time. User-assigned job priority must also be configured.

As long as a job remains pending, LSF automatically increases the job priority beyond the maximum priority specified by MAX_USER_PRIORITY. Job priority is not increased beyond the value of `max_int` on your system.

Pending job resize allocation requests for resizable jobs inherit the job priority from the original job. When the priority of the allocation request gets adjusted, the priority of the original job is adjusted as well. The job priority of a running job is adjusted when there is an associated resize request for allocation growth. `bjobs` displays the updated job priority.

If necessary, a new pending resize request is regenerated after the job gets dispatched. The new job priority is used.

For queued and rerun jobs, the dynamic priority value is reset. For migrated jobs, the existing dynamic priority value is carried forward. The priority is recalculated based on the original value.

Configure job priority escalation

1. To configure job priority escalation edit `lsb.params` and define JOB_PRIORITY_OVER_TIME.

```
JOB_PRIORITY_OVER_TIME=increment/interval
```

Where:

increment

Specifies the value used to increase job priority every *interval* minutes. Valid values are positive integers.

interval

Specifies the frequency, in minutes, to *increment* job priority. Valid values are positive integers.

Note:

User-assigned job priority must also be configured,

2. Use `bparams -1` to display the values of `JOB_PRIORITY_OVER_TIME`.

Example

```
JOB_PRIORITY_OVER_TIME=3/20
```

Specifies that every 20 minute *interval increment* to job priority of pending jobs by 3.

Absolute job priority scheduling

Absolute job priority scheduling (APS) provides a mechanism to control the job dispatch order to prevent job starvation.

When configured in a queue, APS sorts pending jobs for dispatch according to a job priority value calculated based on several configurable job-related factors. Each job priority weighting factor can contain subfactors. Factors and subfactors can be independently assigned a weight.

APS provides administrators with detailed yet straightforward control of the job selection process.

- APS only sorts the jobs; job scheduling is still based on configured LSF scheduling policies. LSF attempts to schedule and dispatch jobs based on their order in the APS queue, but the dispatch order is not guaranteed.
- The job priority is calculated for pending jobs across multiple queues based on the sum of configurable factor values. Jobs are then ordered based on the calculated APS value.
- You can adjust the following for APS factors:
 - A weight for scaling each job-related factor and subfactor
 - Limits for each job-related factor and subfactor
 - A grace period for each factor and subfactor
- To configure absolute priority scheduling (APS) across multiple queues, define APS queue groups. When you submit a job to any queue in a group, the job's dispatch priority is calculated using the formula defined in the group's master queue.
- Administrators can also set a static system APS value for a job. A job with a system APS priority is guaranteed to have a higher priority than any calculated value. Jobs with higher system APS settings have priority over jobs with lower system APS settings.
- Administrators can use the ADMIN factor to manually adjust the calculated APS value for individual jobs.

Scheduling priority factors

To calculate the job priority, APS divides job-related information into several categories. Each category becomes a factor in the calculation of the scheduling priority. You can configure the weight, limit, and grace period of each factor to get the desired job dispatch order.

LSF sums the value of each factor based on the weight of each factor.

Factor weight	The weight of a factor expresses the importance of the factor in the absolute scheduling priority. The factor weight is multiplied by the value of the factor to change the factor value. A positive weight increases the importance of the factor, and a negative weight decreases the importance of a factor. Undefined factors have a weight of 0, which causes the factor to be ignored in the APS calculation.
Factor limit	The limit of a factor sets the minimum and maximum absolute value of each weighted factor. Factor limits must be positive values.
Factor grace period	Each factor can be configured with a grace period. The factor only counted as part of the APS value when the job has been pending for a long time and it exceeds the grace period.

Factors and subfactors

Factors	Subfactors	Metric
FS (user based fairshare factor)	The existing fairshare feature tunes the dynamic user priority	<p>The fairshare factor automatically adjusts the APS value based on dynamic user priority.</p> <p>FAIRSHARE must be defined in the queue. The FS factor is ignored for non-fairshare queues.</p> <p>The FS factor is influenced by the following fairshare parameters defined in <code>lsb.queues</code> or <code>lsb.params</code>:</p> <ul style="list-style-type: none"> • CPU_TIME_FACTOR • RUN_TIME_FACTOR • RUN_JOB_FACTOR • HIST_HOURS
RSRC (resource factors)	PROC	Requested processors is the max of <code>bsub -n min</code> , <code>max</code> , the min of <code>bsub -n min</code> , or the value of <code>PROCLIMIT</code> in <code>lsb.queues</code> .
	MEM	<p>Total real memory requested (in MB).</p> <p>Memory requests appearing to the right of a <code> </code> symbol in a usage string are ignored in the APS calculation.</p> <p>For multi-phase memory reservation, the APS value is based on the first phase of reserved memory.</p>
	SWAP	<p>Total swap space requested (in MB).</p> <p>As with MEM, swap space requests appearing to the right of a <code> </code> symbol in a usage string are ignored.</p>

Factors	Subfactors	Metric
WORK (job attributes)	JPRIORITY	<p>The job priority specified by:</p> <ul style="list-style-type: none"> • Default specified by MAX_USER_PRIORITY in l sb. params • Users with bsub - sp or bmod - sp • Automatic priority escalation with JOB_PRIORITY_OVER_TIME in l sb. params
	QPRIORITY	The priority of the submission queue.
ADMIN		<p>Administrators use bmod - aps to set this subfactor value for each job. A positive value increases the APS. A negative value decreases the APS. The ADMIN factor is added to the calculated APS value to change the factor value. The ADMIN factor applies to the entire job. You cannot configure separate weight, limit, or grace period factors. The ADMIN factor takes effect as soon as it is set.</p>

Where Platform LSF gets the job information for each factor

Factor or subfactor	Gets job information from...
MEM	<p>The value for jobs submitted with -R "rusage[mem]"</p> <p>For compound resource requirements submitted with -R "n1*{rusage[mem1]} + n2*{rusage[mem2]}" the value of MEM depends on whether resources are reserved per slot.</p> <ul style="list-style-type: none"> • If RESOURCE_RESERVE_PER_SLOT=N, then MEM=mem1+mem2 • If RESOURCE_RESERVE_PER_SLOT=Y, then MEM=n1*mem1+n2*mem2
SWAP	<p>The value for jobs submitted with -R "rusage[swp]"</p> <p>For compound resource requirements, SWAP is determined in the same manner as MEM.</p>
PROC	The value of <i>n</i> for jobs submitted with bsub - n (mi n, max), or the value of PROCLIMIT in l sb. queues
JPRIORITY	The dynamic priority of the job, updated every scheduling cycle and escalated by interval defined in JOB_PRIORITY_OVER_TIME defined in l sb. params
QPRIORITY	The priority of the job submission queue
FS	The fairshare priority value of the submission user

Enable absolute priority scheduling

1. Configure APS_PRIORITY in an absolute priority queue in l sb. queues.

APS_PRIORITY=WEIGHT[[*factor, value*] [*subfactor, value*]...] **LIMIT**[[*factor, value*] [*subfactor, value*]...] **GRACE_PERIOD**[[*factor, value*] [*subfactor, value*]...]

Pending jobs in the queue are ordered according to the calculated APS value.

If weight of a subfactor is defined, but the weight of parent factor is not defined, the parent factor weight is set as 1.

The WEIGHT and LIMIT factors are floating-point values. Specify a *value* for GRACE_PERIOD in seconds (*values*), minutes (*valuem*), or hours (*valueh*).

The default unit for grace period is hours.

For example, the following sets a grace period of 10 hours for the MEM factor, 10 minutes for the JPRIORITY factor, 10 seconds for the QPRIORITY factor, and 10 hours (default) for the RSRC factor:

```
GRACE_PERIOD[ [MEM, 10h] [JPRIORITY, 10m] [QPRIORITY, 10s] [RSRC, 10] ]
```

Note:

You cannot specify zero (0) for the WEIGHT, LIMIT, and GRACE_PERIOD of any factor or subfactor.

APS queues cannot configure cross-queue fairshare (FAIRSHARE_QUEUES) or host-partition fairshare.

Modify the system APS value (bmod)

The absolute scheduling priority for a newly submitted job is dynamic. Job priority is calculated and updated based on formula specified by APS_PRIORITY in the absolute priority queue.

You must be an administrator to modify the calculated APS value.

1. Run `bmod job_ID` to manually override the calculated APS value.
2. Run `bmod -apsn job_ID` to undo the previous `bmod -aps` setting.

Assign a static system priority and ADMIN factor value

1. Run `bmod -aps "system=value"` to assign a static job priority for a pending job.

The value cannot be zero (0).

In this case, job's absolute priority is not calculated. The system APS priority is guaranteed to be higher than any calculated APS priority value. Jobs with higher system APS settings have priority over jobs with lower system APS settings.

The system APS value set by `bmod -aps` is preserved after `mbatchd` reconfiguration or `mbatchd` restart.

Use the ADMIN factor to adjust the APS value

1. use `bmod -aps "admin=value"` to change the calculated APS value for a pending job.

The ADMIN factor is added to the calculated APS value to change the factor value. The absolute priority of the job is recalculated. The value cannot be zero (0).

A `bmod -aps` command always overrides the last `bmod -aps` commands

The ADMIN APS value set by `bmod -aps` is preserved after `mbatchd` reconfiguration or `mbatchd` restart.

Example bmod output

The following commands change the APS values for jobs 313 and 314:

```
bmod -aps "system=10" 313
Parameters of job <313> are being changed
bmod -aps "admin=10.00" 314
Parameters of job <314> are being changed
```

View modified APS values

1. Run `bjobs -aps` to see the effect of the changes:

```
bjobs -aps
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME  APS
313    user1  PEND  owners hostA      myjob  Feb 12 01:09  (10)
321    user1  PEND  owners hostA      myjob  Feb 12 01:09  -
314    user1  PEND  normal hostA      myjob  Feb 12 01:08  109.00
312    user1  PEND  normal hostA      myjob  Feb 12 01:08  99.00
315    user1  PEND  normal hostA      myjob  Feb 12 01:08  99.00
316    user1  PEND  normal hostA      myjob  Feb 12 01:08  99.00
```

2. Run `bjobs -l` to show APS values modified by the administrator:

```
bjobs -l
Job <313>, User <user1>, Project <default>, Service Class <SLASamples>, Status
<RUN>, Queue <normal>, Command <myjob>, System Absolute Priority <10>
Job <314>, User <user1>, Project <default>, Status <PEND>, Queue <normal>, Command
<myjob>, Admin factor value <10>
```

3. Use `bhist -l` to see historical information about administrator changes to APS values.

For example, after running these commands:

1. `bmod -aps "system=10" 108`
2. `bmod -aps "admin=20" 108`
3. `bmod -apsn 108`

`bhist -l` shows the sequence changes to job 108:

```
bhist -l
Job <108>, User <user1>, Project <default>, Command <sleep 10000>
Tue Feb 23 15:15:26 2010: Submitted from host <HostB>, to Queue <normal>, CWD </
scratch/user1>;
Tue Feb 23 15:15:40 2010: Parameters of Job are changed:
Absolute Priority Scheduling factor string changed to : system=10;
Tue Feb 23 15:15:48 2010: Parameters of Job are changed:
Absolute Priority Scheduling factor string changed to : admin=20;
Tue Feb 23 15:15:58 2010: Parameters of Job are changed:
Absolute Priority Scheduling factor string deleted;
Summary of time in seconds spent in various states by Tue Feb 23 15:16:02 2010
  PEND  PSUSP  RUN  USUSP  SSUSP  UNKWN  TOTAL
    36     0     0     0     0     0    36
```

Configure APS across multiple queues

1. Use `QUEUE_GROUP` in an absolute priority queue in `lsb.queues` to configure APS across multiple queues.

When APS is enabled in the queue with `APS_PRIORITY`, the `FAIRSHARE_QUEUES` parameter is ignored. The `QUEUE_GROUP` parameter replaces `FAIRSHARE_QUEUES`, which is obsolete in LSF 7.0.

For example, you want to schedule jobs from the normal queue and the short queue, factoring the job priority (weight 1) and queue priority (weight 10) in the APS value:

```
Begin Queue
QUEUE_NAME = normal
PRIORITY = 30
NICE = 20
APS_PRIORITY = WEIGHT [[JPRIORITY, 1] [QPRIORITY, 10]]
QUEUE_GROUP = short
DESCRIPTION = For normal low priority jobs, running only if hosts are lightly
loaded.
End Queue
...
Begin Queue
QUEUE_NAME = short
PRIORITY = 20
NICE = 20
End Queue
```

The APS value for jobs from the normal queue and the short queue are calculated as:

$$APS_PRIORITY = 1 * (1 * job_priority + 10 * queue_priority)$$

The first 1 is the weight of the WORK factor; the second 1 is the weight of the job priority subfactor; the 10 is the weight of queue priority subfactor.

If you want the job priority to increase based on the pending time, you must configure JOB_PRIORITY_OVER_TIME parameter in the lsb. params.

Extending the example, you now want to add user-based fairshare with a weight of 100 to the APS value in the normal queue:

```
Begin Queue
QUEUE_NAME = normal
PRIORITY = 30
NICE = 20
FAIRSHARE = USER_SHARES [[user1, 5000] [user2, 5000] [others, 1]]
APS_PRIORITY = WEIGHT [[JPRIORITY, 1] [QPRIORITY, 10] [FS, 100]]
QUEUE_GROUP = short
DESCRIPTION = For normal low priority jobs, running only if hosts are lightly
loaded.
End Queue
```

The APS value is now calculated as

$$APS_PRIORITY = 1 * (1 * job_priority + 10 * queue_priority) + 100 * user_priority$$

Finally, you now to add swap space to the APS value calculation. The APS configuration changes to:

$$APS_PRIORITY = WEIGHT [[JPRIORITY, 1] [QPRIORITY, 10] [FS, 100] [SWAP, -10]]$$

And the APS value is now calculated as

$$APS_PRIORITY = 1 * (1 * job_priority + 10 * queue_priority) + 100 * user_priority + 1 * (-10 * SWAP)$$

View pending job order by the APS value

1. Run `bjobs -aps` to see APS information for pending jobs in the order of absolute scheduling priority.

The order that the pending jobs are displayed is the order in which the jobs are considered for dispatch.

The APS value is calculated based on the current scheduling cycle, so jobs are not guaranteed to be dispatched in this order.

Pending jobs are ordered by APS value. Jobs with system APS values are listed first, from highest to lowest APS value. Jobs with calculated APS values are listed next ordered from high to low value.

Finally, jobs not in an APS queue are listed. Jobs with equal APS values are listed in order of submission time.

If queues are configured with the same priority, `bjobs -aps` may not show jobs in the correct expected dispatch order. Jobs may be dispatched in the order the queues are configured in `lsb.queues`. You should avoid configuring queues with the same priority.

Example `bjobs -aps` output

The following example uses this configuration;

- The APS only considers the job priority and queue priority for jobs from normal queue (priority 30) and short queue (priority 20)
 - `APS_PRIORITY = WEIGHT [[QRIORITY, 10] [JPRIORITY, 1]]`
 - `QUEUE_GROUP = short`
- Priority queue (40) and idle queue (15) do not use APS to order jobs
- `JOB_PRIORITY_OVER_TIME=5/10` in `lsb.params`
- `MAX_USER_PRIORITY=100` in `lsb.params`

`bjobs -aps` was run at 14:41:

bjobs -aps							
JOBID	USER	STAT	QUEUE	FROM_HOST	JOB_NAME	SUBMIT_TIME	APS
15	User2	PEND	priority	HostB	myjob	Dec 21 14:30	-
22	User1	PEND	Short	HostA	myjob	Dec 21 14:30	(60)
2	User1	PEND	Short	HostA	myjob	Dec 21 11:00	360
12	User2	PEND	normal	HostB	myjob	Dec 21 14:30	355
4	User1	PEND	Short	HostA	myjob	Dec 21 14:00	270
5	User1	PEND	Idle	HostA	myjob	Dec 21 14:01	-

For job 2, $APS = 10 * 20 + 1 * (50 + 220 * 5 / 10) = 360$ For job 12, $APS = 10 * 30 + 1 * (50 + 10 * 5 / 10) = 355$ For job 4, $APS = 10 * 20 + 1 * (50 + 40 * 5 / 10) = 270$

View APS configuration for a queue

1. Run `bqueues -l normal` to see the current APS information for a queue:

```

bqueues -l normal
QUEUE: normal
-- No description provided. This is the default queue.

PARAMETERS/STATISTICS
PRIORITY NICE STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SSUSP  USUSP  RSV
500    20  Open: Active          -   -   -   -   0    0    0    0    0    0

SCHEDULING PARAMETERS
loadSched r15s r1m r15m ut      pg      io      ls      it      tmp      swp      mem
loadStop  -   -   -   -      -      -      -      -      -      -      -

SCHEDULING POLICIES: FAIRSHARE APS_PRIORITY
APS_PRIORITY:
WEIGHT FACTORS      LIMIT FACTORS      GRACE PERIOD
FAIRSHARE           10000.00           -               -
RESOURCE            101010.00           -               1010h
PROCESSORS          - 10.01             -               -
MEMORY              1000.00            20010.00         3h
SWAP                10111.00           -               -
WORK                1.00               -               -
JOB_PRIORITY        -999999.00          10000.00         4131s
QUEUE_PRIORITY      10000.00            10.00           -

USER_SHARES: [user1, 10]

```

```
SHARE_INFO_FOR: normal /
USER/GROUP    SHARES  PRIORITY  STARTED  RESERVED  CPU_TIME  RUN_TIME
user1         10      3.333    0        0         0.0      0
USERS: all
HOSTS: all
REQUEUE_EXIT_VALUES: 10
```

Job priority behavior

Fairshare

The default user-based fairshare can be a factor in APS calculation by adding the FS factor to `APS_PRIORITY` in the queue.

- APS cannot be used together with `DISPATCH_ORDER=QUEUE`.
- APS cannot be used together with cross-queue fairshare (`FAIRSHARE_QUEUES`). The `QUEUE_GROUP` parameter replaces `FAIRSHARE_QUEUES`, which is obsolete in LSF 7.0.
- APS cannot be used together with queue-level fairshare or host-partition fairshare.

FCFS

APS overrides the job sort result of FCFS.

SLA scheduling

APS cannot be used together with time-based SLAs with velocity, deadline, or throughput goals.

Job requeue

All requeued jobs are treated as newly submitted jobs for APS calculation. The job priority, system, and ADMIN APS factors are reset on requeue.

Rerun jobs

Rerun jobs are not treated the same as requeued jobs. A job typically reruns because the host failed, not through some user action (like job requeue), so the job priority is not reset for rerun jobs.

Job migration

Suspended (`bstop`) jobs and migrated jobs (`bmi g`) are always scheduled before pending jobs. For migrated jobs, LSF keeps the existing job priority information.

If `LSB_REQUEUE_TO_BOTTOM` and `LSB_MIG2PEND` are configured in `lsf.conf`, the migrated jobs keep their APS information. When `LSB_REQUEUE_TO_BOTTOM` and `LSB_MIG2PEND` are configured, the migrated jobs need to compete with other pending jobs based on the APS value. If you want to reset the APS value, the you should use `brequeue`, not `bmi g`.

Resource reservation

The resource reservation is based on queue policies. The APS value does not affect current resource reservation policy.

Preemption

The preemption is based on queue policies. The APS value does not affect the current preemption policy.

Chunk jobs

The first chunk job to be dispatched is picked based on the APS priority. Other jobs in the chunk is picked based on the APS priority and the default chunk job scheduling policies.

The following job properties must be the same for all chunk jobs:

- Submitting user
- Resource requirements
- Host requirements
- Queue or application profile
- Job priority

Backfill scheduling

Not affected.

Advance reservation

Not affected.

Resizable jobs

For new resizable job allocation requests, the resizable job inherits the APS value from the original job. The subsequent calculations use factors as follows:

Factor or sub-factor	Behavior
FAIRSHARE	<p>Resizable jobs submitting into fairshare queues or host partitions are subject to fairshare scheduling policies. The dynamic priority of the user who submitted the job is the most important criterion. LSF treats pending resize allocation requests as a regular job and enforces the fairshare user priority policy to schedule them.</p> <p>The dynamic priority of users depends on:</p> <ul style="list-style-type: none"> • Their share assignment • The slots their jobs are currently consuming • The resources their jobs consumed in the past • The adjustment made by the fairshare plugin (libfairshareadjust.*) <p>Resizable job allocation changes affect the user priority calculation if RUN_JOB_FACTOR is greater than zero (0). Resize add requests increase number of slots in use and decrease user priority. Resize release requests decrease number of slots in use, and increase user priority. The faster a resizable job grows, the lower the user priority is, the less likely a pending allocation request can get more slots.</p>
MEM	Use the value inherited from the original job
PROC	Use the MAX value of the resize request
SWAP	Use the value inherited from the original job

Factor or sub-factor	Behavior
JPRIORITY	<p>Use the value inherited from the original job. If the automatic job priority escalation is configured, the dynamic value is calculated.</p> <p>For a requeued and rerun resizable jobs, the JPRIORITY is reset, and the new APS value is calculated with the new JPRIORITY.</p> <p>For migrated resizable job, the JPRIORITY is carried forward, and the new APS value is calculated with the JPRIORITY continued from the original value.</p>
QPRIORITY	Use the value inherited from the original job
ADMIN	Use the value inherited from the original job

Job Requeue and Job Rerun

About job requeue

A networked computing environment is vulnerable to any failure or temporary conditions in network services or processor resources. For example, you might get NFS stale handle errors, disk full errors, process table full errors, or network connectivity problems. Your application can also be subject to external conditions such as a software license problems, or an occasional failure due to a bug in your application.

Such errors are temporary and probably happen at one time but not another, or on one host but not another. You might be upset to learn all your jobs exited due to temporary errors and you did not know about it until 12 hours later.

LSF provides a way to automatically recover from temporary errors. You can configure certain exit values such that in case a job exits with one of the values, the job is automatically requeued as if it had not yet been dispatched. This job is then be retried later. It is also possible for you to configure your queue such that a requeued job is not scheduled to hosts on which the job had previously failed to run.

Automatic job requeue

You can configure a queue to automatically requeue a job if it exits with a specified exit value.

- The job is requeued to the head of the queue from which it was dispatched, unless the `LSB_REQUEUE_TO_BOTTOM` parameter in `lsf.conf` is set.
- When a job is requeued, LSF does not save the output from the failed run.
- When a job is requeued, LSF does not notify the user by sending mail.
- A job terminated by a signal is not requeued.

The reserved keyword `all` specifies all exit codes. Exit codes are typically between 0 and 255. Use a tilde (~) to exclude specified exit codes from the list.

For example:

```
REQUEUE_EXIT_VALUES=all ~1 ~2 EXCLUDE(9)
```

Jobs exited with all exit codes except 1 and 2 are requeued. Jobs with exit code 9 are requeued so that the failed job is not rerun on the same host (exclusive job requeue).

Configure automatic job requeue

1. To configure automatic job requeue, set `REQUEUE_EXIT_VALUES` in the queue definition (`lsb.queues`) or in an application profile (`lsb.applications`) and specify the exit codes that cause the job to be requeued.

Application-level exit values override queue-level values. Job-level exit values (`bsub -Q`) override application-level and queue-level values.

```
Begin Queue
...
REQUEUE_EXIT_VALUES = 99 100
...
End Queue
```

This configuration enables jobs that exit with 99 or 100 to be requeued.

Control how many times a job can be requeued

By default, if a job fails and its exit value falls into `REQUEUE_EXIT_VALUES`, LSF requeues the job automatically. Jobs that fail repeatedly are requeued five times by default.

1. To limit the number of times a failed job is requeued, set `MAX_JOB_REQUEUE` cluster wide (l sb. params), in the queue definition (l sb. queues), or in an application profile (l sb. appl i cat i ons).

Specify an integer greater than zero (0).

`MAX_JOB_REQUEUE` in l sb. appl i cat i ons overrides l sb. queues, and l sb. queues overrides l sb. params configuration. Specifying a job-level exit value using `bsub -Q` overrides all `MAX_JOB_REQUEUE` settings.

When `MAX_JOB_REQUEUE` is set, if a job fails and its exit value falls into `REQUEUE_EXIT_VALUES`, the number of times the job has been requeued is increased by 1 and the job is requeued. When the requeue limit is reached, the job is suspended with `PSUSP` status. If a job fails and its exit value is not specified in `REQUEUE_EXIT_VALUES`, the job is not requeued.

View the requeue retry limit

1. Run `bj obs -l` to display the job exit code and reason if the job requeue limit is exceeded.
2. Run `bhi st -l` to display the exit code and reason for finished jobs if the job requeue limit is exceeded.

The job requeue limit is recovered when LSF is restarted and reconfigured. LSF replays the job requeue limit from the `JOB_STATUS` event and its pending reason in l sb. events.

Job-level automatic requeue

1. Use `bsub -Q` to submit a job that is automatically requeued if it exits with the specified exit values.

Use spaces to separate multiple exit codes. The reserved keyword `all` specifies all exit codes. Exit codes are typically between 0 and 255. Use a tilde (~) to exclude specified exit codes from the list.

Job-level requeue exit values override application-level and queue-level configuration of the parameter `REQUEUE_EXIT_VALUES`, if defined.

Jobs running with the specified exit code share the same application and queue with other jobs.

For example:

```
bsub -Q "all ~1 ~2 EXCLUDE(9)" myjob
```

Jobs exited with all exit codes except 1 and 2 are requeued. Jobs with exit code 9 are requeued so that the failed job is not rerun on the same host (exclusive job requeue).

Enable exclusive job requeue

1. Define an exit code as `EXCLUDE(exit_code)` to enable exclusive job requeue.

Exclusive job requeue does not work for parallel jobs.

Note:

If `mbat chd` is restarted, it does not remember the previous hosts from which the job exited with an exclusive requeue exit code. In this situation, it is possible for a job to be dispatched to hosts on which the job has previously exited with an exclusive exit code.

Modify requeue exit values

1. Use `bmod -Q` to modify or cancel job-level requeue exit values.

`bmod -Q` does not affect running jobs. For rerunnable and requeue jobs, `bmod -Q` affects the next run.

MultiCluster Job forwarding model	For jobs sent to a remote cluster, arguments of <code>bsub -Q</code> take effect on remote clusters.
-----------------------------------	--

MultiCluster Lease model	The arguments of <code>bsub -Q</code> apply to jobs running on remote leased hosts as if they are running on local hosts.
--------------------------	---

Configure reverse requeue

By default, if you use automatic job requeue, jobs are requeued to the head of a queue. You can have jobs requeued to the bottom of a queue instead. The job priority does not change.

You must already use automatic job requeue (`REQUEUE_EXIT_VALUES` in `lsb.queues`).

To configure reverse requeue:

1. Set `LSB_REQUEUE_TO_BOTTOM` in `lsf.conf` to 1.
2. Reconfigure the cluster:
 - a) **lsadmin reconfig**
 - b) **badmin mbdrestart**

Exclusive job requeue

You can configure automatic job requeue so that a failed job is not rerun on the same host.

Limitations

- If `mbat chd` is restarted, this feature might not work properly, since LSF forgets which hosts have been excluded. If a job ran on a host and exited with an exclusive exit code before `mbat chd` was restarted, the job could be dispatched to the same host again after `mbat chd` is restarted.
- Exclusive job requeue does not work for MultiCluster jobs or parallel jobs
- A job terminated by a signal is not requeued

Configure exclusive job requeue

1. Set `REQUEUE_EXIT_VALUES` in the queue definition (`lsb.queues`) and define the exit code using parentheses and the keyword `EXCLUDE`:

EXCLUDE(*exit_code*...)

exit_code has the following form:

"[all] [~number ...] | [number ...]"

The reserved keyword `all` specifies all exit codes. Exit codes are typically between 0 and 255. Use a tilde (~) to exclude specified exit codes from the list.

Jobs are requeued to the head of the queue. The output from the failed run is not saved, and the user is not notified by LSF.

When a job exits with any of the specified exit codes, it is requeued, but it is not dispatched to the same host again.

```
Begin Queue
...
REQUEUE_EXIT_VALUES=30 EXCLUDE(20) HOSTS=hostA hostB hostC
```

```
...
End Queue
```

A job in this queue can be dispatched to host A, host B or host C.

If a job running on host A exits with value 30 and is requeued, it can be dispatched to host A, host B, or host C. However, if a job running on host A exits with value 20 and is requeued, it can only be dispatched to host B or host C.

If the job runs on host B and exits with a value of 20 again, it can only be dispatched on host C. Finally, if the job runs on host C and exits with a value of 20, it cannot be dispatched to any of the hosts, so it is pending forever.

Requeue a job

You can use `brequeue` to kill a job and requeue it. When the job is requeued, it is assigned the PEND status and the job's new position in the queue is after other jobs of the same priority.

1. To requeue one job, use `brequeue`.
 - You can only use `brequeue` on running (RUN), user-suspended (USUSP), or system-suspended (SSUSP) jobs.
 - Users can only requeue their own jobs. Only root and LSF administrator can requeue jobs submitted by other users.
 - You cannot use `brequeue` on interactive batch jobs

brequeue 109

LSF kills the job with job ID 109, and requeues it in the PEND state. If job 109 has a priority of 4, it is placed after all the other jobs with the same priority.

brequeue -u User5 45 67 90

LSF kills and requeues 3 jobs belonging to User5. The jobs have the job IDs 45, 67, and 90.

Automatic job rerun

Job requeue vs. job rerun

Automatic job requeue occurs when a job finishes and has a specified exit code (usually indicating some type of failure).

Automatic job rerun occurs when the execution host becomes unavailable while a job is running. It does not occur if the job itself fails.

About job rerun

When a job is rerun or restarted, it is first returned to the queue from which it was dispatched with the same options as the original job. The priority of the job is set sufficiently high to ensure the job gets dispatched before other jobs in the queue. The job uses the same job ID number. It is executed when a suitable host is available, and an email message is sent to the job owner informing the user of the restart.

Automatic job rerun can be enabled at the job level, by the user, or at the queue level, by the LSF administrator. If automatic job rerun is enabled, the following conditions cause LSF to rerun the job:

- The execution host becomes unavailable while a job is running
- The system fails while a job is running

When LSF reruns a job, it returns the job to the submission queue, with the same job ID. LSF dispatches the job as if it was a new submission, even if the job has been checkpointed.

Once job is rerun, LSF schedules resizable jobs based on their initial allocation request.

Execution host fails

If the execution host fails, LSF dispatches the job to another host. You receive a mail message informing you of the host failure and the requeuing of the job.

LSF system fails

If the LSF system fails, LSF requeues the job when the system restarts.

Configure queue-level job rerun

1. To enable automatic job rerun at the queue level, set `RERUNNABLE` in `lsb. queues` to `yes`.

Submit a rerunnable job

1. To enable automatic job rerun at the job level, use `bsub -r`.
Interactive batch jobs (`bsub -I`) cannot be rerunnable.

Submit a job as not rerunnable

1. To disable automatic job rerun at the job level, use `bsub -rn`.

Disable post-execution for rerunnable jobs

Running of post-execution commands upon restart of a rerunnable job may not always be desirable; for example, if the `post-exec` removes certain files, or does other cleanup that should only happen if the job finishes successfully.

1. Use `LSB_DISABLE_RERUN_POST_EXEC=Y` in `lsf.conf` to prevent the post-exec from running when a job is rerun.

40

Job Migration

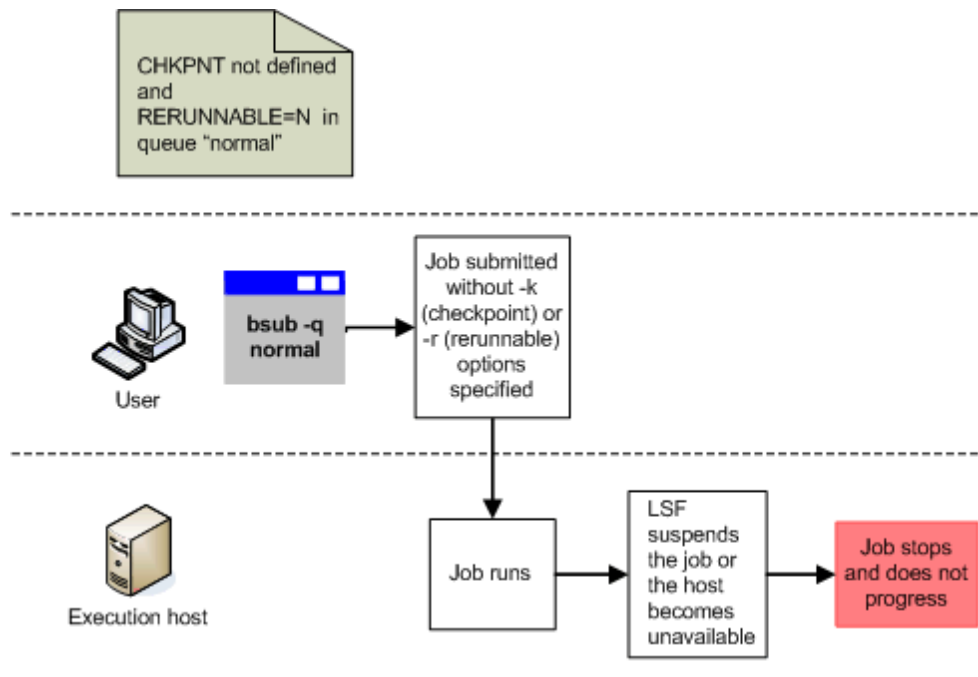
The job migration feature enables you to move checkpointable and rerunnable jobs from one host to another. Job migration makes use of job checkpoint and restart so that a migrated checkpointable job restarts on the new host from the point at which the job stopped on the original host.

About job migration

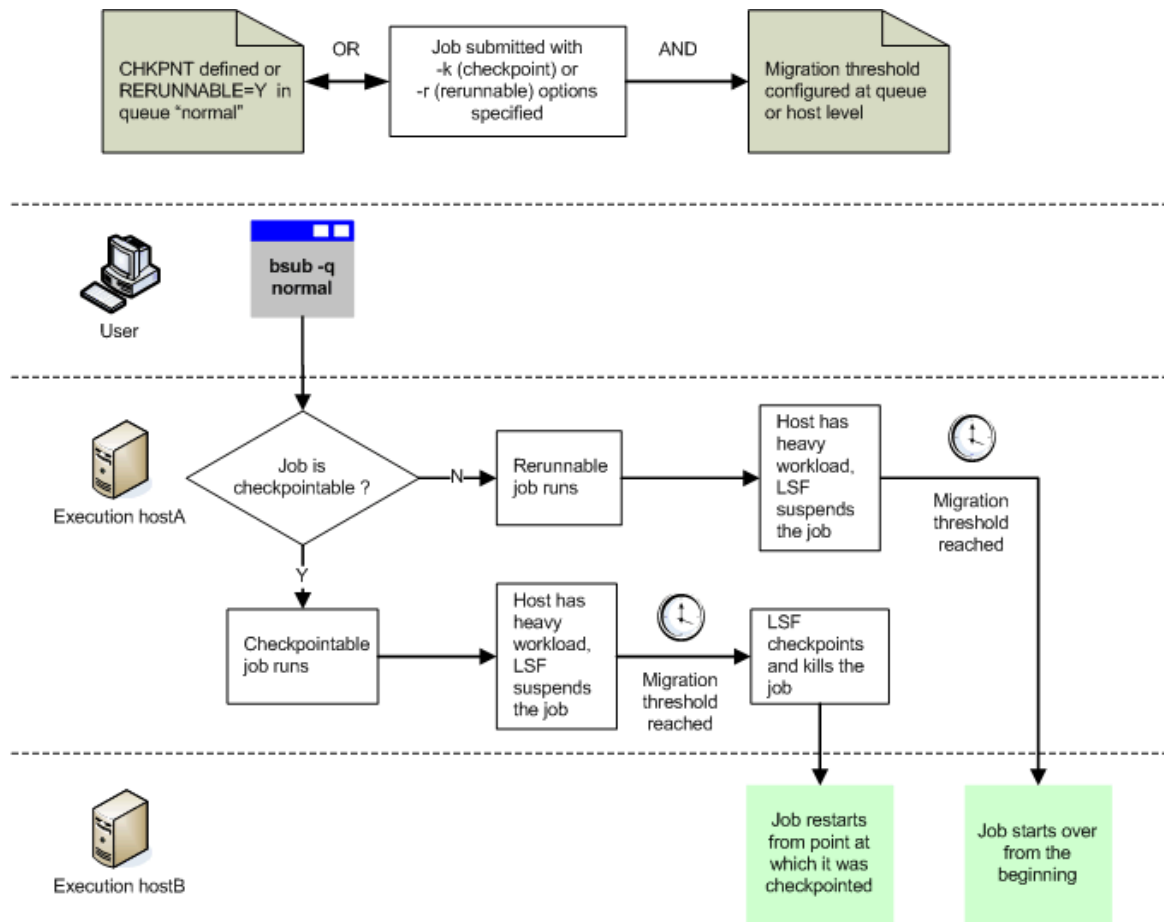
Job migration refers to the process of moving a checkpointable or rerunnable job from one host to another. This facilitates load balancing by moving jobs from a heavily-loaded host to a lightly-loaded host.

You can initiate job migration on demand (bmi g) or automatically. To initiate job migration automatically, you configure a migration threshold at job submission, or at the host, queue, or application level.

Default behavior (job migration not enabled)



With automatic job migration enabled



Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> • UNIX • Linux • Windows
Job types	<ul style="list-style-type: none"> • Non-interactive batch jobs submitted with bsub or bmod, including chunk jobs

Applicability	Details
Dependencies	<ul style="list-style-type: none">• UNIX and Windows user accounts must be valid on all hosts in the cluster, or the correct type of account mapping must be enabled:<ul style="list-style-type: none">• For a mixed UNIX/Windows cluster, UNIX/Windows user account mapping must be enabled• For a cluster with a non-uniform user name space, between-host account mapping must be enabled• For a MultiCluster environment with a non-uniform user name space, cross-cluster user account mapping must be enabled• Both the original and the new hosts must:<ul style="list-style-type: none">• Be binary compatible• Run the same dot version of the operating system for predictable results• Have network connectivity and read/execute permissions to the checkpoint and restart executables (in LSF_SERVERDIR by default)• Have network connectivity and read/write permissions to the checkpoint directory and the checkpoint file• Have access to all files open during job execution so that LSF can locate them using an absolute path name

Configuration to enable job migration

The job migration feature requires that a job be made checkpointable or rerunnable at the job, application, or queue level. An LSF user can make a job

- Checkpointable, using `bsub -k` and specifying a checkpoint directory and checkpoint period, and an optional initial checkpoint period
- Rerunnable, using `bsub -r`

Configuration file	Parameter and syntax	Behavior
lsb. queues	<code>CHKPNT=chkpnt_dir [chkpnt_period]</code>	<ul style="list-style-type: none"> • All jobs submitted to the queue are checkpointable. • The specified checkpoint directory must already exist. LSF will not create the checkpoint directory. • The user account that submits the job must have read and write permissions for the checkpoint directory. • For the job to restart on another execution host, both the original and new hosts must have network connectivity to the checkpoint directory. • If the queue administrator specifies a checkpoint period, in minutes, LSF creates a checkpoint file every <code>chkpnt_period</code> during job execution. • If a user specifies a checkpoint directory and checkpoint period at the job level with <code>bsub -k</code>, the job-level values override the queue-level values.
	<code>RERUNNABLE=Y</code>	<ul style="list-style-type: none"> • If the execution host becomes unavailable, LSF reruns the job from the beginning on a different host.

Configuration file	Parameter and syntax	Behavior
l sb. appl i cat i ons	CHKPNT_DIR= <i>chkpnt_dir</i>	<ul style="list-style-type: none"> Specifies the checkpoint directory for automatic checkpointing for the application. To enable automatic checkpoint for the application profile, administrators must specify a checkpoint directory in the configuration of the application profile. If CHKPNT_PERIOD, CHKPNT_INITPERIOD or CHKPNT_METHOD was set in an application profile but CHKPNT_DIR was not set, a warning message is issued and those settings are ignored. The checkpoint directory is the directory where the checkpoint files are created. Specify an absolute path or a path relative to the current working directory for the job. Do not use environment variables in the directory path. If checkpoint-related configuration is specified in both the queue and an application profile, the application profile setting overrides queue level configuration.
	CHKPNT_INITPERIOD= <i>init_chkpnt_period</i>	
	CHKPNT_PERIOD= <i>chkpnt_period</i>	
	CHKPNT_METHOD= <i>chkpnt_method</i>	

Configuration to enable automatic job migration

Automatic job migration assumes that if a job is system-suspended (SSUSP) for an extended period of time, the execution host is probably heavily loaded. Configuring a queue-level or host-level migration threshold lets the job to resume on another less loaded host, and reduces the load on the original host. You can use `bmi g` at any time to override a configured migration threshold.

Configuration file	Parameter and syntax	Behavior
l sb. queues	MIG= <i>minutes</i>	
l sb. appl i cat i ons		<ul style="list-style-type: none"> LSF automatically migrates jobs that have been in the SSUSP state for more than the specified number of minutes Specify a value of 0 to migrate jobs immediately upon suspension Applies to all jobs submitted to the queue Job-level command line migration threshold (<code>bsub -mi g</code>) overrides threshold configuration in application profile and queue. Application profile configuration overrides queue level configuration.

Configuration file	Parameter and syntax		Behavior
lsb. hosts	HOST_NAME <i>host_name</i>	MIG <i>minutes</i>	<ul style="list-style-type: none">LSF automatically migrates jobs that have been in the SSUSP state for more than the specified number of minutesSpecify a value of 0 to migrate jobs immediately upon suspensionApplies to all jobs running on the host

Note:

When a host migration threshold is specified, and is lower than the value for the job, the queue, or the application, the host value is used.

Job migration behavior

LSF migrates a job by performing the following actions:

1. Stops the job if it is running
2. Checkpoints the job if the job is checkpointable
3. Kills the job on the current host
4. Restarts or reruns the job on the first available host, bypassing all pending jobs

Configuration to modify job migration

You can configure LSF to requeue a migrating job rather than restart or rerun the job.

Configuration file	Parameter and syntax	Behavior
lsf.conf	LSB_MIG2PEND=1	<ul style="list-style-type: none"> LSF requeues a migrating job rather than restarting or rerunning the job LSF requeues the job as pending in order of the original submission time and priority In a MultiCluster environment, LSF ignores this parameter
	LSB_REQUEUE_TO_BOTTOM=1	<ul style="list-style-type: none"> When LSB_MIG2PEND=1, LSF requeues a migrating job to the bottom of the queue, regardless of the original submission time and priority If the queue defines APS scheduling, migrated jobs keep their APS information and compete with other pending jobs based on the APS value

Checkpointing resizable jobs

After a checkpointable resizable job restarts (brestart), LSF restores the original job allocation request. LSF also restores job-level autoresizable attribute and notification command if they are specified at job submission.

Example

The following example shows a queue configured for periodic checkpointing in lsb queues:

```
Begin Queue
...
QUEUE_NAME=checkpoint
CHKPNT=mydir 240
DESCRIPTION=Automatically checkpoints jobs every 4 hours to mydir
...
End Queue
```

Note:

The bqueues command displays the checkpoint period in seconds; the lsb queues CHKPNT parameter defines the checkpoint period in minutes.

If the command `bchkpnt -k 123` is used to checkpoint and kill job 123, you can restart the job using the `brestart` command as shown in the following example:

brestart -q priority mydir 123

Job <456> is submitted to queue <priority>

LSF assigns a new job ID of 456, submits the job to the queue named "priority," and restarts the job.

Once job 456 is running, you can change the checkpoint period using the `bchkpnt` command:

bchkpnt -p 360 456

Job <456> is being checkpointed

Job migration commands

Commands for submission

Job migration applies to checkpointable or rerunnable jobs submitted with a migration threshold, or that have already started and are either running or suspended.

Command	Description
<code>bsub -mi g migration_threshold</code>	<ul style="list-style-type: none"> Submits the job with the specified migration threshold for checkpointable or rerunnable jobs. Enables automatic job migration and specifies the migration threshold, in minutes. A value of 0 (zero) specifies that a suspended job should be migrated immediately. Command-level job migration threshold overrides application profile and queue-level settings. Where a host migration threshold is also specified, and is lower than the job value, the host value is used.

Commands to monitor

Command	Description
<code>bhi st -l</code>	<ul style="list-style-type: none"> Displays the actions that LSF took on a completed job, including migration to another host
<code>bj obs -l</code>	<ul style="list-style-type: none"> Displays information about pending, running, and suspended jobs

Commands to control

Command	Description
<code>bmi g</code>	<ul style="list-style-type: none"> Migrates one or more running jobs from one host to another. The jobs must be checkpointable or rerunnable Checkpoints, kills, and restarts one or more checkpointable jobs—<code>bmi g</code> combines the functionality of the <code>bchkpnt</code> and <code>brstart</code> commands into a single command Migrates the job on demand even if you have configured queue-level or host-level migration thresholds When absolute job priority scheduling (APS) is configured in the queue, LSF schedules migrated jobs before pending jobs—for migrated jobs, LSF maintains the existing job priority
<code>bmod -mi g migration_threshold -mi gn</code>	<ul style="list-style-type: none"> Modifies or cancels the migration threshold specified at job submission for checkpointable or rerunnable jobs. Enables or disables automatic job migration and specifies the migration threshold, in minutes. A value of 0 (zero) specifies that a suspended job should be migrated immediately. Command-level job migration threshold overrides application profile and queue-level settings. Where a host migration threshold is also specified, and is lower than the job value, the host value is used.

Commands to display configuration

Command	Description
<code>bhosts -l</code>	<ul style="list-style-type: none"> Displays information about hosts configured in <code>lsb.hosts</code>, including the values defined for migration thresholds in minutes
<code>bqueues -l</code>	<ul style="list-style-type: none"> Displays information about queues configured in <code>lsb.queues</code>, including the values defined for migration thresholds <p>Note:</p> <p>The <code>bqueues</code> command displays the migration threshold in seconds—the <code>lsb.queues MIG</code> parameter defines the migration threshold in minutes.</p>
<code>badmi n showconf</code>	<ul style="list-style-type: none"> Displays all configured parameters and their values set in <code>lsf.conf</code> or <code>ego.conf</code> that affect <code>mbatchd</code> and <code>sbatchd</code>. <p>Use a text editor to view other parameters in the <code>lsf.conf</code> or <code>ego.conf</code> configuration files.</p> <ul style="list-style-type: none"> In a MultiCluster environment, displays the parameters of daemons on the local cluster.

Job Checkpoint and Restart

The job checkpoint and restart feature enables you to stop jobs and then restart them from the point at which they stopped, which optimizes resource usage. LSF can periodically capture the state of a running job and the data required to restart it. This feature provides fault tolerance and allows LSF administrators and users to migrate jobs from one host to another to achieve load balancing.

About job checkpoint and restart

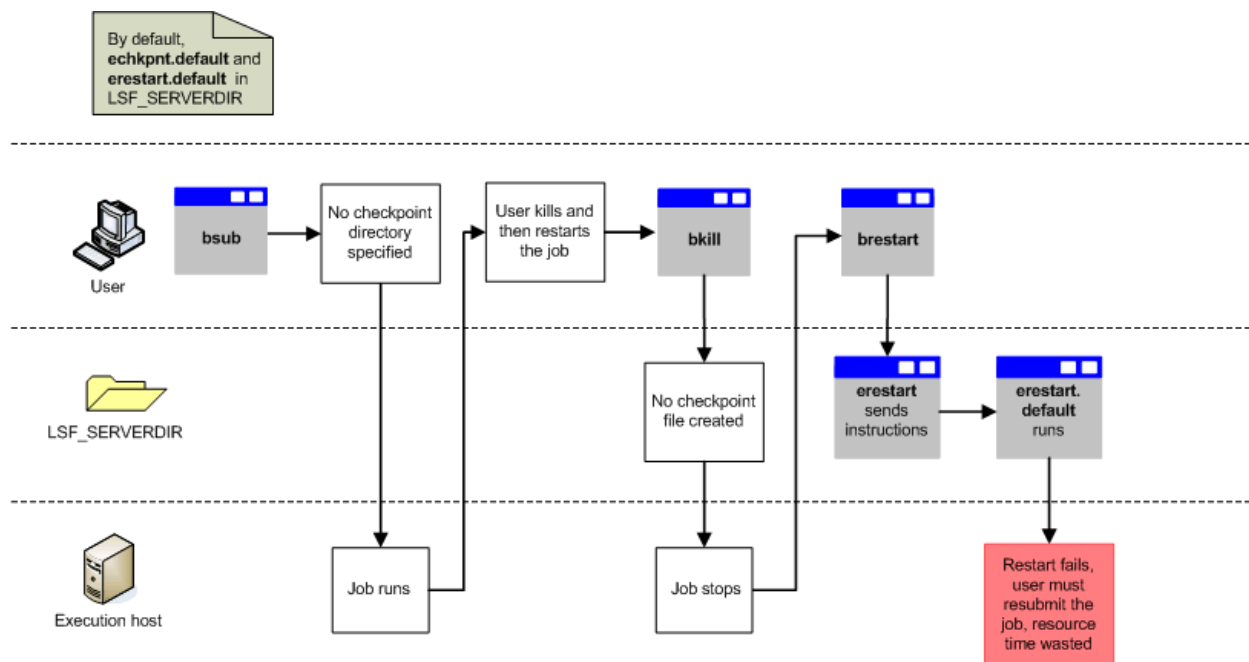
Checkpointing enables LSF users to restart a job on the same execution host or to migrate a job to a different execution host. LSF controls checkpointing and restart by means of interfaces named `echkpt` and `erestart`. By default, when a user specifies a checkpoint directory using `bsub -k` or `bmod -k` or submits a job to a queue that has a checkpoint directory specified, `echkpt` sends checkpoint instructions to an executable named `echkpt.default`.

When LSF checkpoints a job, the `echkpt` interface creates a checkpoint file in the directory `checkpoint_dir/job_ID`, and then checkpoints and resumes the job. The job continues to run, even if checkpointing fails.

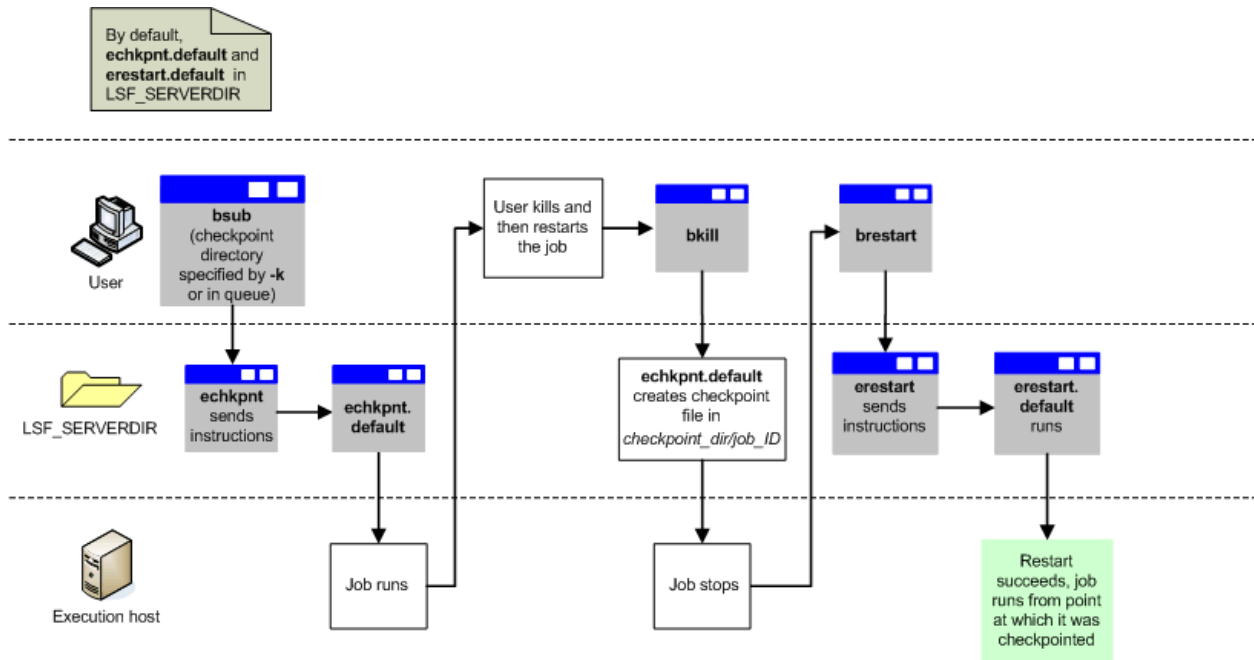
When LSF restarts a stopped job, the `erestart` interface recovers job state information from the checkpoint file, including information about the execution environment, and restarts the job from the point at which the job stopped. At job restart, LSF

1. Resubmits the job to its original queue and assigns a new job ID
2. Dispatches the job when a suitable host becomes available (not necessarily the original execution host)
3. Re-creates the execution environment based on information from the checkpoint file
4. Restarts the job from its most recent checkpoint

Default behavior (job checkpoint and restart not enabled)



With job checkpoint and restart enabled



Kernel-level checkpoint and restart

The operating system provides checkpoint and restart functionality that is transparent to your applications and enabled by default. To implement job checkpoint and restart at the kernel level, the LSF `echkpnt` and `erestart` executables invoke operating system-specific calls.

LSF uses the default executables `echkpnt.default` and `erestart.default` for kernel-level checkpoint and restart.

User-level checkpoint and restart

For systems that do not support kernel-level checkpoint and restart, LSF provides a job checkpoint and restart implementation that is transparent to your applications and does not require you to rewrite code. User-level job checkpoint and restart is enabled by linking your application files to the LSF checkpoint libraries in `LSF_LIBDIR`. LSF uses the default executables `echkpnt.default` and `erestart.default` for user-level checkpoint and restart.

Application-level checkpoint and restart

Different applications have different checkpointing implementations that require the use of customized external executables (`echkpnt.application` and `erestart.application`). Application-level checkpoint and restart enables you to configure LSF to use specific `echkpnt.application` and `erestart.application` executables for a job, queue, or cluster. You can write customized checkpoint and restart executables for each application that you use.

LSF uses a combination of corresponding checkpoint and restart executables. For example, if you use `echkpnt.fluent` to checkpoint a particular job, LSF will use `erestart.fluent` to restart the checkpointed job. You cannot override this behavior or configure LSF to use a specific restart executable.

Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> Kernel-level checkpoint and restart using the LSF checkpoint libraries works only with supported operating system versions and architecture for: <ul style="list-style-type: none"> SGI IRIX 6.4 and later SGI Altix ProPack 3 and later
Job types	<ul style="list-style-type: none"> Non-interactive batch jobs submitted with <code>bsub</code> or <code>bmod</code> Non-interactive batch jobs, including chunk jobs, checkpointed with <code>bchkpnt</code> Non-interactive batch jobs migrated with <code>bmi g</code> Non-interactive batch jobs restarted with <code>brestart</code>
Dependencies	<ul style="list-style-type: none"> UNIX and Windows user accounts must be valid on all hosts in the cluster, or the correct type of account mapping must be enabled. <ul style="list-style-type: none"> For a mixed UNIX/Windows cluster, UNIX/Windows user account mapping must be enabled. For a cluster with a non-uniform user name space, between-host account mapping must be enabled. For a MultiCluster environment with a non-uniform user name space, cross-cluster user account mapping must be enabled. The checkpoint and restart executables run under the user account of the user who submits the job. User accounts must have the correct permissions to <ul style="list-style-type: none"> Successfully run executables located in <code>LSF_SERVERDIR</code> or <code>LSB_ECHKPNT_METHOD_DIR</code> Write to the checkpoint directory The <code>erestart.application</code> executable must have access to the original command line used to submit the job. For user-level checkpoint and restart, you must have access to your application object (<code>.o</code>) files. To allow restart of a checkpointed job on a different host than the host on which the job originally ran, both the original and the new hosts must: <ul style="list-style-type: none"> Be binary compatible Run the same dot version of the operating system for predictable results Have network connectivity and read/execute permissions to the checkpoint and restart executables (in <code>LSF_SERVERDIR</code> by default) Have network connectivity and read/write permissions to the checkpoint directory and the checkpoint file Have access to all files open during job execution so that LSF can locate them using an absolute path name
Limitations	<ul style="list-style-type: none"> <code>bmod</code> cannot change the <code>echkpnt</code> and <code>erestart</code> executables associated with a job. Linux 32, AIX, and HP platforms with NFS (network file systems), checkpoint directories (including path and file name) must be shorter than 1000 characters. Linux 64 with NFS (network file systems), checkpoint directories (including path and file name) must be shorter than 2000 characters.

Configuration to enable job checkpoint and restart

The job checkpoint and restart feature requires that a job be made checkpointable at the job or queue level. LSF users can make jobs checkpointable by submitting jobs using `bsub -k` and specifying a checkpoint directory. Queue administrators can make all jobs in a queue checkpointable by specifying a checkpoint directory for the queue.

Configuration file	Parameter and syntax	Behavior
lsb.queues	<code>CHKPNT=chkpnt_dir[chkpnt_period]</code>	<ul style="list-style-type: none"> All jobs submitted to the queue are checkpointable. LSF writes the checkpoint files, which contain job state information, to the checkpoint directory. The checkpoint directory can contain checkpoint files for multiple jobs. The specified checkpoint directory must already exist. LSF will not create the checkpoint directory. The user account that submits the job must have read and write permissions for the checkpoint directory. For the job to restart on another execution host, both the original and new hosts must have network connectivity to the checkpoint directory. If the queue administrator specifies a checkpoint period, in minutes, LSF creates a checkpoint file every <i>chkpnt_period</i> during job execution. <p>Note:</p> <p>There is no default value for checkpoint period. You must specify a checkpoint period if you want to enable periodic checkpointing.</p> <ul style="list-style-type: none"> If a user specifies a checkpoint directory and checkpoint period at the job level with <code>bsub -k</code>, the job-level values override the queue-level values. The file path of the checkpoint directory can contain up to 4000 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.
lsb.appl icat ion s		

Configuration to enable kernel-level checkpoint and restart

Kernel-level checkpoint and restart is enabled by default. LSF users make a job checkpointable by either submitting a job using `bsub -k` and specifying a checkpoint directory or by submitting a job to a queue that defines a checkpoint directory for the `CHKPNT` parameter.

Configuration to enable user-level checkpoint and restart

To enable user-level checkpoint and restart, you must link your application object files to the LSF checkpoint libraries provided in `LSF_LIBDIR`. You do not have to change any code within your application. For instructions on how to link application files, see the *Platform LSF Programmer's Guide*.

Configuration to enable application-level checkpoint and restart

Application-level checkpointing requires the presence of at least one `echkpkt. application` executable in the directory specified by the parameter `LSF_SERVERDIR` in `lsf.conf`. Each `echkpkt. application` must have a corresponding `erestart. application`.

Important:

The `erestart. application` executable must:

- Have access to the command line used to submit or modify the job
- Exit with a return value without running an application; the `erestart` interface runs the application to restart the job

Executable file	UNIX naming convention	Windows naming convention
echkpkt	LSF_SERVERDIR/echkpkt. application	LSF_SERVERDIR\echkpkt. application.exe LSF_SERVERDIR\echkpkt. application.bat
erestart	LSF_SERVERDIR/erestart. application	LSF_SERVERDIR\erestart. application.exe LSF_SERVERDIR\erestart. application.bat

Restriction:

The names `echkpkt.default` and `erestart.default` are reserved. Do not use these names for application-level checkpoint and restart executables.

Valid file names contain only alphanumeric characters, underscores (`_`), and hyphens (`-`).

For application-level checkpoint and restart, once the `LSF_SERVERDIR` contains one or more checkpoint and restart executables, users can specify the external checkpoint executable associated with each checkpointable job they submit. At restart, LSF invokes the corresponding external restart executable.

Requirements for application-level checkpoint and restart executables

- The executables must be written in C or Fortran.
- The directory/name combinations must be unique within the cluster. For example, you can write two different checkpoint executables with the name `echkpkt.fluent` and save them as `LSF_SERVERDIR/echkpkt.fluent` and `my_execs/echkpkt.fluent`. To run checkpoint and restart executables from a directory other than `LSF_SERVERDIR`, you must configure the parameter `LSB_ECHKPNT_METHOD_DIR` in `lsf.conf`.

- Your executables must return the following values.
 - An `echkpnt.application` must return a value of 0 when checkpointing succeeds and a non-zero value when checkpointing fails.
 - The `erestart` interface provided with LSF restarts the job using a restart command that `erestart.application` writes to a file. The return value indicates whether `erestart.application` successfully writes the parameter definition `LSB_RESTART_CMD=restart_command` to the file `checkpoint_dir/job_ID/.restart_cmd`.
 - A non-zero value indicates that `erestart.application` failed to write to the `.restart_cmd` file.
 - A return value of 0 indicates that `erestart.application` successfully wrote to the `.restart_cmd` file, or that the executable intentionally did not write to the file.
- Your executables must recognize the syntax used by the `echkpnt` and `erestart` interfaces, which communicate with your executables by means of a common syntax.

- `echkpnt.application` syntax:

```
echkpnt [-c] [-f] [-k | -s] [-d checkpoint_dir] [-x] process_group_ID
```

Restriction:

The `-k` and `-s` options are mutually exclusive.

- `erestart.application` syntax:

```
erestart [-c] [-f] checkpoint_dir
```

Option or variable	Description	Operating systems
<code>-c</code>	Copies all files in use by the checkpointed process to the checkpoint directory.	Some, such as SGI systems running IRIX and Altix
<code>-f</code>	Forces a job to be checkpointed even under non-checkpointable conditions, which are specific to the checkpoint implementation used. This option could create checkpoint files that do not provide for successful restart.	Some, such as SGI systems running IRIX and Altix
<code>-k</code>	Kills a job after successful checkpointing. If checkpoint fails, the job continues to run.	All operating systems that LSF supports
<code>-s</code>	Stops a job after successful checkpointing. If checkpoint fails, the job continues to run.	Some, such as SGI systems running IRIX and Altix
<code>-d checkpoint_dir</code>	Specifies the checkpoint directory as a relative or absolute path.	All operating systems that LSF supports
<code>-x</code>	Identifies the cpr (checkpoint and restart) process as type HID. This identifies the set of processes to checkpoint as a process hierarchy (tree) rooted at the current PID.	Some, such as SGI systems running IRIX and Altix
<code>process_group_ID</code>	ID of the process or process group to checkpoint.	All operating systems that LSF supports

Job checkpoint and restart behavior

LSF invokes the `echkpnt` interface when a job is

- Automatically checkpointed based on a configured checkpoint period
- Manually checkpointed with `bchkpnt`
- Migrated to a new host with `bmi g`

After checkpointing, LSF invokes the `erestart` interface to restart the job. LSF also invokes the `erestart` interface when a user

- Manually restarts a job using `brestart`
- Migrates the job to a new host using `bmi g`

All checkpoint and restart executables run under the user account of the user who submits the job.

Note:

By default, LSF redirects standard error and standard output to `/dev/null` and discards the data.

Checkpoint directory and files

LSF identifies checkpoint files by the checkpoint directory and job ID. For example:

```
bsub -k my_dir
Job <123> is submitted to default queue <default>
```

LSF writes the checkpoint file to `my_dir/123`.

LSF maintains all of the checkpoint files for a single job in one location. When a job restarts, LSF creates both a new subdirectory based on the new job ID and a symbolic link from the old to the new directory. For example, when job 123 restarts on a new host as job 456, LSF creates `my_dir/456` and a symbolic link from `my_dir/123` to `my_dir/456`.

The file path of the checkpoint directory can contain up to 4000 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.

Precedence of job, queue, application, and cluster-level checkpoint values

LSF handles checkpoint and restart values as follows:

1. *Checkpoint directory and checkpoint period*—values specified at the job level override values for the queue. Values specified in an application profile setting overrides queue level configuration.
If checkpoint-related configuration is specified in the queue, application profile, and at job level:
 - Application-level and job-level parameters are merged. If the same parameter is defined at both job-level and in the application profile, the job-level value overrides the application profile value.
 - The merged result of job-level and application profile settings override queue-level configuration.
2. *Checkpoint and restart executables*—the value for `checkpoint_method` specified at the job level overrides the application-level `CHKPNT_METHOD`, and the cluster-level value for `LSB_ECHKPNT_METHOD` specified in `lsf.conf` or as an environment variable.
3. *Configuration parameters and environment variables*—values specified as environment variables override the values specified in `lsf.conf`

If the command line is...	And...	Then...
bsub -k "my_dir 240"	In l sb. queues, CHKPNT=other_dir 360	<ul style="list-style-type: none"> LSF saves the checkpoint file to <i>my_dir/job_ID</i> every 240 minutes
bsub -k "my_dir fluent"	In l sf. conf, LSB_ECHKPNT_METHOD=myapp	<ul style="list-style-type: none"> LSF invokes echkpnt.fluent at job checkpoint and erestart.fluent at job restart
bsub -k "my_dir"	In l sb. applications, CHKPNT_PERIOD=360	<ul style="list-style-type: none"> LSF saves the checkpoint file to <i>my_dir/job_ID</i> every 360 minutes
bsub -k "240"	In l sb. applications, CHKPNT_DIR=app_dir CHKPNT_PERIOD=360 In l sb. queues, CHKPNT=other_dir	<ul style="list-style-type: none"> LSF saves the checkpoint file to <i>app_dir/job_ID</i> every 240 minutes

Configuration to modify job checkpoint and restart

There are configuration parameters that modify various aspects of job checkpoint and restart behavior by:

- Specifying mandatory application-level checkpoint and restart executables that apply to all checkpointable batch jobs in the cluster
- Specifying the directory that contains customized application-level checkpoint and restart executables
- Saving standard output and standard error to files in the checkpoint directory
- Automatically checkpointing jobs before suspending or terminating them
- For Cray systems only, copying all open job files to the checkpoint directory

Configuration to specify mandatory application-level executables

You can specify mandatory checkpoint and restart executables by defining the parameter `LSB_ECHKPNT_METHOD` in `lsf.conf` or as an environment variable.

Configuration file	Parameter and syntax	Behavior
<code>lsf.conf</code>	<code>LSB_ECHKPNT_METHOD=" <i>echkpnt_application</i> "</code>	<ul style="list-style-type: none"> • The specified <code>echkpnt</code> runs for all batch jobs submitted to the cluster. At restart, the corresponding <code>erestart</code> runs. • For example, if <code>LSB_ECHKPNT_METHOD=fluent</code>, at checkpoint, LSF runs <code>echkpnt.fluent</code> and at restart, LSF runs <code>erestart.fluent</code>. • If an LSF user specifies a different <i>echkpnt_application</i> at the job level using <code>bsub -k</code> or <code>bmod -k</code>, the job level value overrides the value in <code>lsf.conf</code>.

Configuration to specify the directory for application-level executables

By default, LSF looks for application-level checkpoint and restart executables in `LSF_SERVERDIR`. You can modify this behavior by specifying a different directory as an environment variable or in `lsf.conf`.

Configuration file	Parameter and syntax	Behavior
<code>lsf.conf</code>	<code>LSB_ECHKPNT_METHOD_DIR=<i>path</i></code>	<ul style="list-style-type: none"> • Specifies the absolute path to the directory that contains the <code>echkpnt.application</code> and <code>erestart.application</code> executables • User accounts that run these executables must have the correct permissions for the <code>LSB_ECHKPNT_METHOD_DIR</code> directory.

Configuration to save standard output and standard error

By default, LSF redirects the standard output and standard error from checkpoint and restart executables to `/dev/null` and discards the data. You can modify this behavior by defining the parameter `LSB_ECHKPNT_KEEP_OUTPUT` as an environment variable or in `lsf.conf`.

Configuration file	Parameter and syntax	Behavior
<code>lsf.conf</code>	<code>LSB_ECHKPNT_KEEP_OUTPUT=Y y</code>	<ul style="list-style-type: none"> The <code>stdout</code> and <code>stderr</code> for <code>echkpnt.application</code> or <code>echkpnt.default</code> are redirected to <code>checkpoint_dir/job_ID/</code> <ul style="list-style-type: none"> <code>echkpnt.out</code> <code>echkpnt.err</code> The <code>stdout</code> and <code>stderr</code> for <code>erestart.application</code> or <code>erestart.default</code> are redirected to <code>checkpoint_dir/job_ID/</code> <ul style="list-style-type: none"> <code>erestart.out</code> <code>erestart.err</code>

Configuration to checkpoint jobs before suspending or terminating them

LSF administrators can configure LSF at the queue level to checkpoint jobs before suspending or terminating them.

Configuration file	Parameter and syntax	Behavior
<code>lsb.queue</code> s	<code>JOB_CONTROLS=SUSPEND CHKPNT TERMINATE</code>	<ul style="list-style-type: none"> LSF checkpoints jobs before suspending or terminating them When suspending a job, LSF checkpoints the job and then stops it by sending the <code>SIGSTOP</code> signal When terminating a job, LSF checkpoints the job and then kills it

Configuration to copy open job files to the checkpoint directory

For hosts that use the Cray operating system, LSF administrators can configure LSF at the host level to copy all open job files to the checkpoint directory every time the job is checkpointed.

Configuration file	Parameter and syntax	Behavior
<code>lsb.host</code> s	<code>HOST_NAME CHKPNT</code> <code>host_name C</code>	<ul style="list-style-type: none"> LSF copies all open job files to the checkpoint directory when a job is checkpointed

Job checkpoint and restart commands

Commands for submission

Command	Description
<code>bsub -k "checkpoint_dir [checkpoint_period] [method=echkpt_application]"</code>	<ul style="list-style-type: none"> Specifies a relative or absolute path for the checkpoint directory and makes the job checkpointable. If the specified checkpoint directory does not already exist, LSF creates the checkpoint directory. If a user specifies a checkpoint period (in minutes), LSF creates a checkpoint file every <i>chkpt_period</i> during job execution. The command-line values for the checkpoint directory and checkpoint period override the values specified for the queue. If a user specifies an <i>echkpt_application</i>, LSF runs the corresponding restart executable when the job restarts. For example, for <code>bsub -k "my_dir method=fluent"</code> LSF runs <code>echkpt.fluent</code> at job checkpoint and <code>erestart.fluent</code> at job restart. The command-line value for <i>echkpt_application</i> overrides the value specified by <code>LSB_ECHKPNT_METHOD</code> in <code>lsf.conf</code> or as an environment variable. Users can override <code>LSB_ECHKPNT_METHOD</code> and use the default checkpoint and restart executables by defining method=default.

Commands to monitor

Command	Description
<code>bacct -l</code>	<ul style="list-style-type: none"> Displays accounting statistics for finished jobs, including termination reasons. <code>TERM_CHKPNT</code> indicates that a job was checkpointed and killed. If <code>JOB_CONTROL</code> is defined for a queue, LSF does not display the result of the action.
<code>bhist -l</code>	<ul style="list-style-type: none"> Displays the actions that LSF took on a completed job, including job checkpoint, restart, and migration to another host.
<code>bjobs -l</code>	<ul style="list-style-type: none"> Displays information about pending, running, and suspended jobs, including the checkpoint directory, the checkpoint period, and the checkpoint method (either <i>application</i> or <i>default</i>).

Commands to control

Command	Description
<code>bmod -k "checkpoint_dir [checkpoint_period] [method=echkpt_application]"</code>	<ul style="list-style-type: none"> Resubmits a job and changes the checkpoint directory, checkpoint period, and the checkpoint and restart executables associated with the job.

Command	Description
<code>bmod -kn</code>	<ul style="list-style-type: none"> Dissociates the checkpoint directory from a job, which makes the job no longer checkpointable.
<code>bchkpnt</code>	<ul style="list-style-type: none"> Checkpoint the most recently submitted checkpointable job. Users can specify particular jobs to checkpoint by including various <code>bchkpnt</code> options.
<code>bchkpnt -p <i>checkpoint_period</i> <i>job_ID</i></code>	<ul style="list-style-type: none"> Checkpoint a job immediately and changes the checkpoint period for the job.
<code>bchkpnt -k <i>job_ID</i></code>	<ul style="list-style-type: none"> Checkpoint a job immediately and kills the job.
<code>bchkpnt -p 0 <i>job_ID</i></code>	<ul style="list-style-type: none"> Checkpoint a job immediately and disables periodic checkpointing.
<code>brestart</code>	<ul style="list-style-type: none"> Restarts a checkpointed job on the first available host.
<code>brestart -m</code>	<ul style="list-style-type: none"> Restarts a checkpointed job on the specified host or host group.
<code>bmi g</code>	<ul style="list-style-type: none"> Migrates one or more running jobs from one host to another. The jobs must be checkpointable or rerunnable. Checkpoints, kills, and restarts one or more checkpointable jobs.

Commands to display configuration

Command	Description
<code>bqueues -l</code>	<ul style="list-style-type: none"> Displays information about queues configured in <code>l sb. queues</code>, including the values defined for checkpoint directory and checkpoint period. <p>Note:</p> <p>The <code>bqueues</code> command displays the checkpoint period in seconds; the <code>l sb. queues CHPNT</code> parameter defines the checkpoint period in minutes.</p>
<code>badmi n showconf</code>	<ul style="list-style-type: none"> Displays all configured parameters and their values set in <code>l sf. conf</code> or <code>ego. conf</code> that affect <code>mbatchd</code> and <code>sbatchd</code>. Use a text editor to view other parameters in the <code>l sf. conf</code> or <code>ego. conf</code> configuration files. In a MultiCluster environment, displays the parameters of daemons on the local cluster.

Resizable Jobs

Enabling resizable jobs allows jobs to dynamically use the number of slots available at any given time or release slots that are no longer needed.

About resizable jobs

Resizable job

To optimize resource utilization, LSF allows job allocation to shrink and grow during the job run time.

Use resizable jobs for long-tailed jobs, jobs that use a large number of processors for a period, but then toward the end of the job use a smaller number of processors.

Without resizable jobs, a job's slot allocation is static from the time the job is dispatched until it finishes. This means resources are wasted, even if you use reservation and backfill (estimated runtimes can be inaccurate). With resizable jobs, jobs can have additional slots added when needed, during the job's runtime.

Autoresizable job

An autoresizable job is a resizable job with a minimum and maximum slot request, where LSF automatically schedules and allocates additional resources to satisfy the job maximum request as the job runs.

Use autoresizable jobs for jobs in which tasks are easily parallelized: Each step or task can be made to run on a separate processor to achieve a faster result. The more resources the job gets, the faster the job can run. Session Scheduler jobs are very good candidates.

For autoresizable jobs, LSF automatically recalculates the pending allocation requests. The maximum pending allocation request is calculated based on the maximum number of requested slots minus the number of allocated slots. Because the job is running and its previous minimum request is already satisfied, LSF is able to allocate additional slots to the running job. For instance, if job requests a minimum of 4 and a maximum of 32, if LSF allocates 20 slots to the job initially, its active pending allocation request is for another 12 slots. After LSF assigns another 4 slots, the pending allocation request is now 8 slots.

Default behavior (feature not enabled)

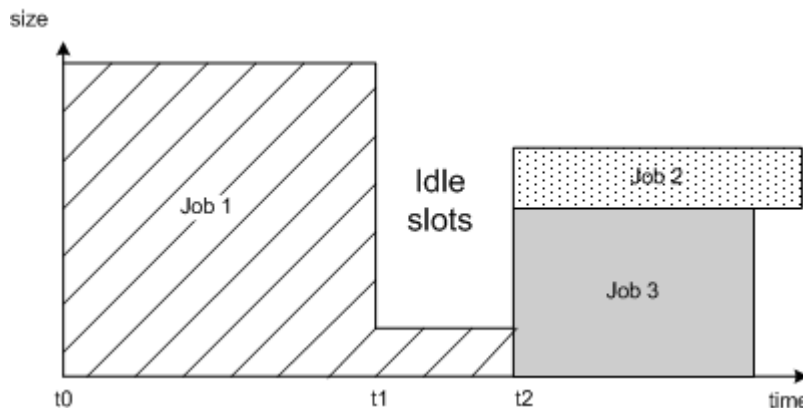


Figure 17: Long-tailed: wasted slots

With resizable jobs enabled

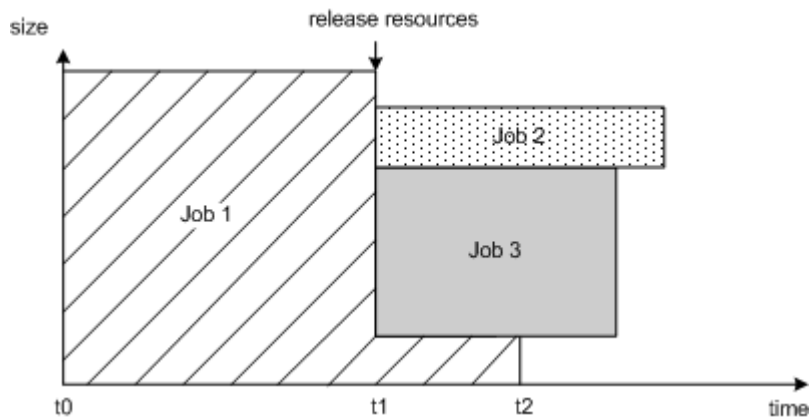


Figure 18: Long-tailed: releasing resources (shrink)

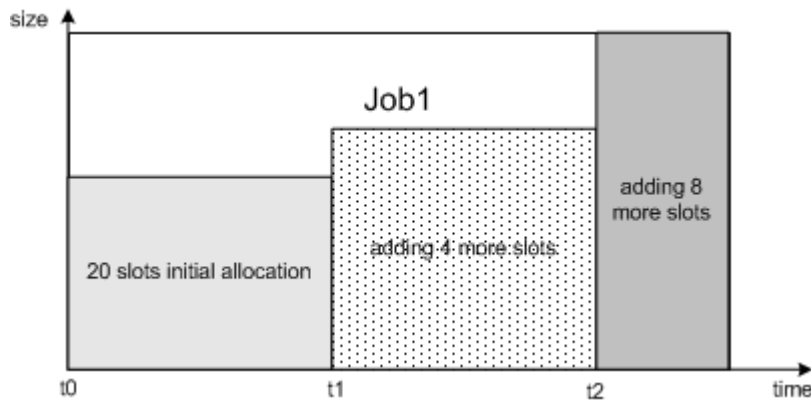


Figure 19: Adding resources (grow)

Pending allocation request

A pending allocation request is an additional resource request attached to a resizable job. Only running jobs can have pending allocation requests. At any given time, a job only has one allocation request.

LSF creates a new pending allocation request and schedules it after a job physically starts on the remote host (after LSF receives the `JOB_EXECUTE` event from `sbatchd`) or resize notification command successfully completes.

Resize notification command

A resize notification command is an executable that is invoked on the first execution host of a job in response to an allocation (grow or shrink) event. It can be used to inform the running application for allocation change. Due to the variety of implementations of applications, each resizable application may have its own notification command provided by the application developer.

The notification command runs under the same user ID environment, home, and working directory as the actual job. The standard input, output, and error of the program are redirected to the NULL device. If the notification command is not in the user's normal execution path (the `SPATH` variable), the full path name of the command must be specified.

A notification command exits with one of the following values:

```
LSB_RESIZE_NOTIFY_OK=0
```

```
LSB_RESIZE_NOTIFY_FAIL=1
```

LSF sets these environment variables in the notification command environment.

`LSB_RESIZE_NOTIFY_OK` indicates notification succeeds. For allocation grow and shrink events, LSF updates the job allocation to reflect the new allocation.

`LSB_RESIZE_NOTIFY_FAIL` indicates notification failure. For allocation "grow" event, LSF reschedules the pending allocation request. For allocation "shrink" event, LSF fails the allocation release request.

For a list of other environment variables that apply to the resize notification command, see the environment variables reference documentation in this guide.

Configuration to enable resizable jobs

The resizable jobs feature is enabled by defining an application profile using the `RESIZABLE_JOBS` parameter in `lsb. appl i cat i ons`.

Configuration file	Parameter and syntax	Behavior
lsb. appl i cat i ons	<code>RESIZABLE_JOBS=Y N auto</code>	<ul style="list-style-type: none"> When <code>RESIZABLE_JOBS=Y</code> jobs submitted to the application profile are resizable. When <code>RESIZABLE_JOBS=auto</code> jobs submitted to the application profile are automatically resizable. To enable cluster-wide resizable behavior by default, define <code>RESIZABLE_JOBS=Y</code> in the default application profile.
	<code>RESIZE_NOTIFY_CMD=notify_cmd</code>	<p><code>RESIZE_NOTIFY_CMD</code> specifies an application-level resize notification command. The resize notification command is invoked on the first execution host of a running resizable job when a resize event occurs.</p> <p>LSF sets appropriate environment variables to indicate the event type before running the notification command.</p>

Configuration to modify resizable job behavior

There is no configuration to modify resizable job behavior.

Resizable job commands

Commands for submission

Command	Description
<code>bsub -app application_profile_name</code>	Submits the job to the specified application profile configured for resizable jobs
<code>bsub -app application_profile_name -rnc resize_notification_command</code>	Submits the job to the specified application profile configured for resizable jobs, with the specified resize notification command. The job-level resize notification command overrides the application-level <code>RESIZE_NOTIFY_CMD</code> setting.
<code>bsub -ar -app application_profile_name</code>	Submits the job to the specified application profile configured for resizable jobs, as an autoresizable job. The job-level <code>-ar</code> option overrides the application-level <code>RESIZABLE_JOBS</code> setting. For example, if the application profile is not autoresizable, job level <code>bsub -ar</code> will make the job autoresizable.

Commands to monitor

Command	Description
<code>bacct -l</code>	<ul style="list-style-type: none"> Displays resize notification command. Displays resize allocation changes.
<code>bhist -l</code>	<ul style="list-style-type: none"> Displays resize notification command. Displays resize allocation changes. Displays the job-level autoresizable attribute.
<code>bjobs -l</code>	<ul style="list-style-type: none"> Displays resize notification command. Displays resize allocation changes. Displays the job-level autoresizable attribute. Displays pending resize allocation requests.

Commands to control

Command	Description
<code>bmod -ar -arn</code>	Add or remove the job-level autoresizable attribute. <code>bmod</code> only updates the autoresizable attribute for pending jobs.
<code>bmod -rnc resize_notification_cmd -rncn</code>	Modify or remove resize notification command for submitted job.

Command	Description
<code>bresize release</code>	<p>Release allocated resources from a running resizable job.</p> <ul style="list-style-type: none"> • Release all slots except one slot from the first execution node. • Release all hosts except the first execution node. • Release a list of hosts, with the option to specify slots to release on each host. • Specify a resize notification command to be invoked on the first execution host of the job. <p>Example:</p> <p>bresize release "1*hostA 2*hostB hostC" 221</p> <p>To release resources from a running job, the job must be submitted to an application profile configured as resizable.</p> <ul style="list-style-type: none"> • By default, only cluster administrators, queue administrators, root and the job owner are allowed to run <code>bresize</code> to change job allocations. • User group administrators are allowed to run <code>bresize</code> to change the allocation of jobs within their user groups.
<code>bresize cancel</code>	<p>Cancel a pending allocation request. If job does not have active pending request, the command fails with an error message.</p>
<code>bresize release -rnc resize_notification_cmd</code>	<p>Specify or remove a resize notification command. The resize notification is invoked on the job first execution node. The resize notification command only applies to this release request and overrides the corresponding resize notification parameters defined in either the application profile (<code>RESIZE_NOTIFY_CMD</code> in <code>lsb. applications</code>) and job level (<code>bsub -rnc notify_cmd</code>), only for this resize request.</p> <p>If the resize notification command completes successfully, LSF considers the allocation release done and updates the job allocation. If the resize notification command fails, LSF does not update the job allocation.</p> <p>The <code>resize_notification_cmd</code> specifies the name of the executable to be invoked on the first execution host when the job's allocation has been modified.</p> <p>The resize notification command runs under the user account that submitted the job.</p> <p><code>-rncn</code> overrides the resize notification command in both job-level and application-level for this <code>bresize</code> request.</p>
<code>bresize release -c</code>	<p>By default, if the job has an active pending allocation request, LSF does not allow users to release resource. Use the <code>bresize release -c</code> command to cancel the active pending resource request when releasing slots from existing allocation. By default, the command only releases slots.</p> <p>If a job still has an active pending allocation request, but you do not want to allocate more resources to the job, use the <code>bresize cancel</code> command to cancel allocation request.</p> <p>Only the job owner, cluster administrators, queue administrators, user group administrators, and root are allowed to cancel pending resource allocation requests.</p>

Commands to display configuration

Command	Description
bapp	Displays the value of parameters defined in lsb. applications.

Autoresizable job management

Autoresizable jobs can have resources released or added.

Submit an autoresizable job

1. Run **bsub -n 4,10 -ar**.

LSF dispatches the job (as long as the minimum slot request is satisfied).

After the job successfully starts, LSF continues to schedule and allocate additional resources to satisfy the maximum slot request for the job.

2. (Optional, as required) Release resources that are no longer needed.

bresize release *released_host_specification job_ID*

where *released_host_specification* is the specification (list or range of hosts and number of slots) of resources to be released.

Example: **bresize release "1*hostA 2*hostB hostC" 221**

LSF releases 1 slot on hostA, 2 slots on hostB, and all slots on hostC for job221.

Result: The resize notification command runs on the first execution host.

Check pending resize requests

A resize request pends until the job's maximum slot request has been allocated or the job finishes (or the resize request is canceled).

1. Run **bjobs -l *job_id***.

Cancel an active pending request

Only the job owner, cluster administrators, queue administrators, user group administrators, and root can cancel pending resource allocation requests.

An allocation request must be pending.

If a job still has an active pending resize request, but you do not want to allocate more resources to the job, you can cancel it.

By default, if the job has an active pending resize request, you cannot release the resources. You must cancel the request first.

1. Run **bresize cancel**.

Specify a resize notification command manually

You can specify a resize notification command on job submission, other than one that is set up for the application profile

1. On job submission, run **`bsub -rnc resize_notification_cmd`**.

The job submission command overrides the application profile setting.

2. Ensure the resize notification command checks any environment variables for resizing.

For example, `LSB_RESIZE_EVENT` indicates why the notification command was called (grow or shrink) and `LSB_RESIZE_HOSTS` lists slots and hosts. Use `LSB_JOBID` to determine which job is affected.

The command you specified runs on the first execution host of the resized job.

LSF monitors the exit code from the command and takes appropriate action when the command returns an exit code corresponding to resize failure.

Script for resizing

```
#!/bin/sh
# The purpose of this script is to inform
# an application of a resize event.
#
# You can identify the application by:
#
#   1. LSF job ID ($LSB_JOBID), or
#   2. pid ($LSB_JOBPID).

# handle the 'grow' event
if [ $LSB_RESIZE_EVENT = "grow" ]; then

    # Inform the application that it can use
    # additional slots as specified in
    # $LSB_RESIZE_HOSTS.
    #
    # Exit with $LSB_RESIZE_NOTIFY_FAIL if
    # the application fails to resize.
    #
    # If the application cannot use any
    # additional resources, you may want
    # to run 'bresize cancel $LSB_JOBID'
    # before exit.

    exit $LSB_RESIZE_NOTIFY_OK
fi

# handle the 'shrink' event
if [ $LSB_RESIZE_EVENT = "shrink" ]; then

    # Instruct the application to release the
    # slots specified in $LSB_RESIZE_HOSTS.
    #
    # Exit with $LSB_RESIZE_NOTIFY_FAIL if
    # the resources cannot be released.

    exit $LSB_RESIZE_NOTIFY_OK
fi

# unknown event -- should not happen
exit $LSB_RESIZE_NOTIFY_FAIL
```


How resizable jobs works with other Platform LSF features

Resource usage	<p>When a job grows or shrinks, its resource reservation (for example memory or shared resources) changes proportionately.</p> <ul style="list-style-type: none"> • Job-based resource usage does not change in grow or shrink operations. • Host-based resource usage changes only when the job gains slots on a new host or releases all slots on a host. • Slot-based resource usage changes whenever the job grows or shrinks.
Limits	<p>Slots are only added to a job's allocation when resize occurs if the job does not violate any resource limits placed on it.</p>
Job scheduling and dispatch	<p>The JOB_ACCEPT_INTERVAL parameter in l sb. params or l sb. queues controls the number of seconds to wait after dispatching a job to a host before dispatching a second job to the same host. The parameter applies to all allocated hosts of a parallel job. For resizable job allocation requests, JOB_ACCEPT_INTERVAL applies to newly allocated hosts.</p>
Chunk jobs	<p>Because candidate jobs for the chunk job feature are short-running sequential jobs, the resizable job feature does not support job chunking:</p> <ul style="list-style-type: none"> • Autoresizable jobs in a chunk queue or application profile cannot be chunked together • bresi ze commands to resize job allocations do not apply to running chunk job members
brequeue	<p>Jobs requeued with brequeue start from the beginning. After requeue, LSF restores the original allocation request for the job.</p>
blaunch	<p>Parallel tasks running through bl aunch can be resizable.</p>
bswitch	<p>bswi tch can switch resizable jobs between queues regardless of job state (including job's resizing state). Once the job is switched, the parameters in new queue apply, including threshold configuration, run limit, CPU limit, queue-level resource requirements, etc.</p>
User group administrators	<p>User group administrators are allowed to issue bresi ze commands to release a part of resources from job allocation (bresi ze rel ease) or cancel active pending resize request (bresi ze cancel).</p>
Requeue exit values	<p>If job-level, application-level or queue-level REQUEUE_EXIT_VALUES are defined, and as long as job exits with a defined exit code, LSF puts the requeued job back to PEND status. For resizable jobs, LSF schedules the job according to the initial allocation request regardless of any job allocation size change.</p>
Automatic job rerun	<p>A rerunnable job is rescheduled after the first running host becomes unreachable. Once job is rerun, LSF schedules resizable jobs based on their initial allocation request.</p>
Compute units	<p>Autoresizable jobs cannot have compute unit requirements.</p>

Compound
resource
requirements

Resizable jobs cannot have compound resource requirements.

Chunk Jobs and Job Arrays

Job chunking

LSF supports *job chunking*, where jobs with similar resource requirements submitted by the same user are grouped together for dispatch. The `CHUNK_JOB_SIZE` parameter in `lsb.queues` and `lsb.applications` specifies the maximum number of jobs allowed to be dispatched together in a *chunk job*.

Job chunking can have the following advantages:

- Reduces communication between `sbatchd` and `mbatchd`, and scheduling overhead in `mbatchd`
- Increases job throughput in `mbatchd` and more balanced CPU utilization on the execution hosts

All of the jobs in the chunk are dispatched as a unit rather than individually. Job execution is sequential, but each chunk job member is not necessarily executed in the order it was submitted.

Restriction:

You cannot auto-migrate a suspended chunk job member.

Job arrays

LSF provides a structure called a *job array* that allows a sequence of jobs that share the same executable and resource requirements, but have different input files, to be submitted, controlled, and monitored as a single unit. Using the standard LSF commands, you can also control and monitor individual jobs and groups of jobs submitted from a job array.

After the job array is submitted, LSF independently schedules and dispatches the individual jobs.

Job packs

If your jobs are not related and do not have similar resource requirements, but you still want to submit a large group of jobs quickly and reduce system overhead, you can use the job packs feature instead of job arrays or job chunking.

Chunk job dispatch

Jobs with the following characteristics are typical candidates for job chunking:

- Take between 1 and 2 minutes to run
- All require the same resource (for example a software license or a specific amount of memory)
- Do not specify a beginning time (`bsub -b`) or termination time (`bsub -t`)

Running jobs with these characteristics without chunking can under utilize resources because LSF spends more time scheduling and dispatching the jobs than actually running them.

Configuring a special high-priority queue for short jobs is not desirable because users may be tempted to send all of their jobs to this queue, knowing that it has high priority.

Note:

Throughput can deteriorate if the chunk job size is too big. Performance may decrease on queues with `CHUNK_JOB_SIZE` greater than 30. You should evaluate the chunk job size on your own systems for best performance.

Restrictions on chunk jobs

`CHUNK_JOB_SIZE` is ignored and jobs are not chunked under the following conditions:

- Interactive queues (`INTERACTIVE = ONLY` parameter)
- CPU limit greater than 30 minutes (`CPULIMIT` parameter in `l sb. queues` or `l sb. appl i cat i ons`). If `CHUNK_JOB_DURATION` is set in `l sb. params`, the job is chunked only if it is submitted with a CPU limit that is less than or equal to the value of `CHUNK_JOB_DURATION` (`bsub -c`)
- Run limit greater than 30 minutes (`RUNLIMIT` parameter in `l sb. queues` or `l sb. appl i cat i ons`). If `CHUNK_JOB_DURATION` is set in `l sb. params`, the job is chunked only if it is submitted with a run limit that is less than or equal to the value of `CHUNK_JOB_DURATION` (`bsub -W`)
- Run time estimate greater than 30 minutes (`RUNTIME` parameter in `l sb. appl i cat i ons`)

Jobs submitted with the following `bsub` options are not chunked; they are dispatched individually:

- `-I` (interactive jobs)
- `-c` (jobs with CPU limit greater than 30)
- `-W` (jobs with run limit greater than 30 minutes)
- `-app` (jobs associated with an application profile that specifies a run time estimate or run time limit greater than 30 minutes, or a CPU limit greater than 30). `CHUNK_JOB_SIZE` is either not specified in the application, or `CHUNK_JOB_SIZE=1`, which disables chunk job dispatch configured in the queue.
- `-R "cu[]"` (jobs with a compute unit resource requirement).

Configure queue-level job chunking

By default, `CHUNK_JOB_SIZE` is not enabled.

1. To configure a queue to dispatch chunk jobs, specify the `CHUNK_JOB_SIZE` parameter in the queue definition in `l sb. queues`.

For example, the following configures a queue named `chunk`, which dispatches up to 4 jobs in a chunk:

```
Begin Queue
```

```

QUEUE_NAME      = chunk
PRIORITY        = 50
CHUNK_JOB_SIZE  = 4
End Queue

```

After adding `CHUNK_JOB_SIZE` to `l sb. queues`, use `badmi n reconfi g` to reconfigure your cluster.

Configure application-level job chunking

By default, `CHUNK_JOB_SIZE` is not enabled. Enabling application-level job chunking overrides queue-level job chunking.

1. To configure an application profile to chunk jobs together, specify the `CHUNK_JOB_SIZE` parameter in the application profile definition in `l sb. appl i cat i ons`.

Specify `CHUNK_JOB_SIZE=1` to disable job chunking for the application. This value overrides chunk job dispatch configured in the queue.

After adding `CHUNK_JOB_SIZE` to `l sb. appl i cat i ons`, use `badmi n reconfi g` to reconfigure your cluster.

Configure limited job chunking

If `CHUNK_JOB_DURATION` is defined in the file `l sb. params`, a job submitted to a chunk job queue is chunked under the following conditions:

- A job-level CPU limit or run time limit is specified (`bsub -c` or `-W`), or
- An application-level CPU limit, run time limit, or run time estimate is specified (`CPULIMIT`, `RUNLIMIT`, or `RUNTIME` in `l sb. appl i cat i ons`), or
- A queue-level CPU limit or run time limit is specified (`CPULIMIT` or `RUNLIMIT` in `l sb. queues`),

and the values of the CPU limit, run time limit, and run time estimate are all less than or equal to the `CHUNK_JOB_DURATION`.

Jobs are not chunked if:

- The CPU limit, run time limit, or run time estimate is greater than the value of `CHUNK_JOB_DURATION`, or
- No CPU limit, no run time limit, and no run time estimate are specified.

The value of `CHUNK_JOB_DURATION` is displayed by `bparams -l`.

1. After adding `CHUNK_JOB_DURATION` to `l sb. params`, use `badmi n reconfi g` to reconfigure your cluster.

By default, `CHUNK_JOB_DURATION` is not enabled.

How Platform LSF submits and controls chunk jobs

When a job is submitted to a queue or application profile configured with the `CHUNK_JOB_SIZE` parameter, LSF attempts to place the job in an existing chunk. A job is added to an existing chunk if it has the same characteristics as the first job in the chunk:

- Submitting user
- Resource requirements
- Host requirements
- Queue or application profile
- Job priority

If a suitable host is found to run the job, but there is no chunk available with the same characteristics, LSF creates a new chunk.

Resources reserved for any member of the chunk are reserved at the time the chunk is dispatched and held until the whole chunk finishes running. Other jobs requiring the same resources are not dispatched until the chunk job is done.

For example, if all jobs in the chunk require a software license, the license is checked out and each chunk job member uses it in turn. The license is not released until the last chunk job member is finished running.

WAIT status

When `sbat chd` receives a chunk job, it does not start all member jobs at once. A chunk job occupies a single job slot. Even if other slots are available, the chunk job members must run one at a time in the job slot they occupy. The remaining jobs in the chunk that are waiting to run are displayed as `WAIT` by `bj obs`. Any jobs in `WAIT` status are included in the count of pending jobs by `bqueues` and `busers`. The `bhost s` command shows the single job slot occupied by the entire chunk job in the number of jobs shown in the `NJOBS` column.

The `bhist -l` command shows jobs in `WAIT` status as `Waiting . . .`

The `bj obs -l` command does not display a `WAIT` reason in the list of pending jobs.

Control chunk jobs

Job controls affect the state of the members of a chunk job. You can perform the following actions on jobs in a chunk job:

Action (Command)	Job State	Effect on Job (State)
Suspend (<code>bst op</code>)	<code>PEND</code>	Removed from chunk (<code>PSUSP</code>)
	<code>RUN</code>	All jobs in the chunk are suspended (<code>NRUN -1, NSUSP +1</code>)
	<code>USUSP</code>	No change
	<code>WAIT</code>	Removed from chunk (<code>PSUSP</code>)
Kill (<code>bki ll</code>)	<code>PEND</code>	Removed from chunk (<code>NJOBS -1, PEND -1</code>)
	<code>RUN</code>	Job finishes, next job in the chunk starts if one exists (<code>NJOBS -1, PEND -1</code>)
	<code>USUSP</code>	Job finishes, next job in the chunk starts if one exists (<code>NJOBS -1, PEND -1, SUSP -1, RUN +1</code>)
	<code>WAIT</code>	Job finishes (<code>NJOBS-1, PEND -1</code>)
Resume (<code>bresume</code>)	<code>USUSP</code>	Entire chunk is resumed (<code>RUN +1, USUSP -1</code>)
Migrate (<code>bmi g</code>)	<code>WAIT</code>	Removed from chunk
Switch queue (<code>bswi tch</code>)	<code>RUN</code>	Job is removed from the chunk and switched; all other <code>WAIT</code> jobs are queued to <code>PEND</code>
	<code>WAIT</code>	Only the <code>WAIT</code> job is removed from the chunk and switched, and queued to <code>PEND</code>
Checkpoint (<code>bchkpnt</code>)	<code>RUN</code>	Job is checkpointed normally

Action (Command)	Job State	Effect on Job (State)
Modify (bmod)	PEND	Removed from the chunk to be scheduled later

Migrating jobs with `bmi g` changes the dispatch sequence of the chunk job members. They are not redispached in the order they were originally submitted.

Rerunnable chunk jobs

If the execution host becomes unavailable, rerunnable chunk job members are removed from the queue and dispatched to a different execution host.

Checkpoint chunk jobs

Only running chunk jobs can be checkpointed. If `bchkpnt -k` is used, the job is also killed after the checkpoint file has been created. If chunk job in WAIT state is checkpointed, `mbatchd` rejects the checkpoint request.

Fairshare policies and chunk jobs

Fairshare queues can use job chunking. Jobs are accumulated in the chunk job so that priority is assigned to jobs correctly according to the fairshare policy that applies to each user. Jobs belonging to other users are dispatched in other chunks.

TERMINATE_WHEN job control action

If the `TERMINATE_WHEN` job control action is applied to a chunk job, `sbatchd` kills the chunk job element that is running and puts the rest of the waiting elements into pending state to be rescheduled later.

Enforce resource usage limits on chunk jobs

By default, resource usage limits are not enforced for chunk jobs because chunk jobs are typically too short to allow LSF to collect resource usage.

1. To enforce resource limits for chunk jobs, define `LSB_CHUNK_RUSAGE=Y` in `lsf.conf`. Limits may not be enforced for chunk jobs that take less than a minute to run.

Job arrays

Job arrays are groups of jobs with the same executable and resource requirements, but different input files. Job arrays can be submitted, controlled, and monitored as a single unit or as individual jobs or groups of jobs.

Each job submitted from a job array shares the same job ID as the job array and are uniquely referenced using an array index. The dimension and structure of a job array is defined when the job array is created.

Syntax

The `bsub` syntax used to create a job array follows:

```
bsub -J "arrayName[indexList, ...]" myJob
```

Where:

-J "arrayName [indexList, ...]" Names and creates the job array. The square brackets, [], around `indexList` must be entered exactly as shown and the job array name specification must be enclosed in quotes. Commas (,) are used to separate multiple `indexList` entries. The maximum length of this specification is 255 characters.

arrayName User specified string used to identify the job array. Valid values are any combination of the following characters:

```
a-z | A-Z | 0-9 | . | - | _
```

indexList = start[-end[:step]] Specifies the size and dimension of the job array, where:

start Specifies the start of a range of indices. Can also be used to specify an individual index. Valid values are unique positive integers. For example, [1-5] and [1, 2, 3, 4, 5] specify 5 jobs with indices 1 through 5.

end Specifies the end of a range of indices. Valid values are unique positive integers.

step Specifies the value to increment the indices in a range. Indices begin at `start`, increment by the value of `step`, and do not increment past the value of `end`. The default value is 1. Valid values are positive integers. For example, [1-10: 2] specifies a range of 1-10 with step value 2 creating indices 1, 3, 5, 7, and 9.

After the job array is created (submitted), individual jobs are referenced using the job array name or job ID and an index value. For example, both of the following series of job array statements refer to jobs submitted from a job array named `myArray` which is made up of 1000 jobs and has a job ID of 123:

```
myArray[1], myArray[2], myArray[3], ..., myArray[1000] 123[1], 123[2], 123[3], ..., 123[1000]
```

Create a job array

1. Create a job array at job submission time.

For example, the following command creates a job array named `myArray` made up of 1000 jobs.

```
bsub -J "myArray[1-1000]" myJob  
Job <123> is submitted to default queue <normal>.
```


Change the maximum size of a job array

A large job array allows a user to submit a large number of jobs to the system with a single job submission.

By default, the maximum number of jobs in a job array is 1000, which means the maximum size of a job array cannot exceed 1000 jobs.

1. Set `MAX_JOB_ARRAY_SIZE` in `lsb.params` to any positive integer between 1 and 2147483646.

The maximum number of jobs in a job array cannot exceed the value set by `MAX_JOB_ARRAY_SIZE`.

Handle input and output files

LSF provides methods for coordinating individual input and output files for the multiple jobs created when submitting a job array. These methods require your input files to be prepared uniformly. To accommodate an executable that uses standard input and standard output, LSF provides runtime variables (`%I` and `%J`) that are expanded at runtime. To accommodate an executable that reads command line arguments, LSF provides an environment variable (`LSB_JOBINDEX`) that is set in the execution environment.

Prepare input files

LSF needs all the input files for the jobs in your job array to be located in the same directory. By default LSF assumes the current working directory (CWD); the directory from where `bsub` was issued.

1. To override CWD, specify an absolute path when submitting the job array.

Each file name consists of two parts, a consistent name string and a variable integer that corresponds directly to an array index. For example, the following file names are valid input file names for a job array. They are made up of the consistent name `input.` and integers that correspond to job array indices from 1 to 1000:

```
input.1, input.2, input.3, ..., input.1000
```

Redirect standard input

The variables `%I` and `%J` are used as substitution strings to support file redirection for jobs submitted from a job array. At execution time, `%I` is expanded to provide the job array index value of the current job, and `%J` is expanded to provide the job ID of the job array.

1. Use the `-i` option of `bsub` and the `%I` variable when your executable reads from standard input.

To use `%I`, all the input files must be named consistently with a variable part that corresponds to the indices of the job array. For example:

```
input.1, input.2, input.3, ..., input.N
```

For example, the following command submits a job array of 1000 jobs whose input files are named `input.1`, `input.2`, `input.3`, ..., `input.1000` and located in the current working directory:

```
bsub -J "myArray[1-1000]" -i "input.%I" myJob
```

Redirect standard output and error

1. Use the `-o` option of `bsub` and the `%I` and `%J` variables when your executable writes to standard output and error.
 - a) To create an output file that corresponds to each job submitted from a job array, specify `%I` as part of the output file name.

For example, the following command submits a job array of 1000 jobs whose output files are put in CWD and named out put . 1, out put . 2, out put . 3, ..., out put . 1000:

```
bsub -J "myArray[1-1000]" -o "output.%I" myJob
```

- b) To create output files that include the job array job ID as part of the file name specify %J.

For example, the following command submits a job array of 1000 jobs whose output files are put in CWD and named out put . 123. 1, out put . 123. 2, out put . 123. 3, ..., out put . 123. 1000. The job ID of the job array is 123.

```
bsub -J "myArray[1-1000]" -o "output.%J.%I" myJob
```

Pass arguments on the command line

The environment variable LSB_JOBINDEX is used as a substitution string to support passing job array indices on the command line. When the job is dispatched, LSF sets LSB_JOBINDEX in the execution environment to the job array index of the current job. LSB_JOBINDEX is set for all jobs. For non-array jobs, LSB_JOBINDEX is set to zero (0).

To use LSB_JOBINDEX, all the input files must be named consistently and with a variable part that corresponds to the indices of the job array. For example:

```
i nput . 1, i nput . 2, i nput . 3, . . . , i nput . N
```

You must escape LSB_JOBINDEX with a backslash, \, to prevent the shell interpreting bsub from expanding the variable. For example, the following command submits a job array of 1000 jobs whose input files are named i nput . 1, i nput . 2, i nput . 3, ..., i nput . 1000 and located in the current working directory. The executable is being passed an argument that specifies the name of the input files:

```
bsub -J "myArray[1-1000]" myJob -f input.\$LSB_JOBINDEX
```

Set a whole array dependency

Like all jobs in LSF, a job array can be dependent on the completion or partial completion of a job or another job array. A number of job-array-specific dependency conditions are provided by LSF.

1. To make a job array dependent on the completion of a job or another job array use the -w "dependency_condition" option of bsub.

For example, to have an array dependent on the completion of a job or job array with job ID 123, use the following command:

```
bsub -w "done(123)" -J "myArray2[1-1000]" myJob
```

Set a partial array dependency

1. To make a job or job array dependent on an existing job array, use one of the following dependency conditions.

Condition	Description
numrun(jobArrayJobId, op num)	Evaluate the number of jobs in RUN state
numpend(jobArrayJobId, op num)	Evaluate the number of jobs in PEND state
numdone(jobArrayJobId, op num)	Evaluate the number of jobs in DONE state
numexit(jobArrayJobId, op num)	Evaluate the number of jobs in EXIT state
numended(jobArrayJobId, op num)	Evaluate the number of jobs in DONE and EXIT state

Condition	Description
numhold(jobArrayJobId, op num)	Evaluate the number of jobs in PSUSP state
numstart(jobArrayJobId, op num)	Evaluate the number of jobs in RUN and SSUSP and USUSP state

2. Use one the following operators (op) combined with a positive integer (num) to build a condition:

```
== | > | < | >= | <= | !=
```

Optionally, an asterisk (*) can be used in place of num to mean all jobs submitted from the job array.

For example, to start a job named myJob when 100 or more elements in a job array with job ID 123 have completed successfully:

```
bsub -w "numdone(123, >= 100)" myJob
```

Monitor job arrays

Use `bj obs` and `bhi st` to monitor the current and past status of job arrays.

Display job array status

1. To display summary information about the currently running jobs submitted from a job array, use the `-A` option of `bj obs`.

For example, a job array of 10 jobs with job ID 123:

```
bjobs -A 123
```

JOBID	ARRAY_SPEC	OWNER	NJOBS	PEND	DONE	RUN	EXIT	SSUSP	USUSP	PSUSP	
123	myArra[1- 10]	user1		10	3	3	4	0	0	0	0

Display job array dependencies

1. To display information for any job dependency information for an array, use the `bj depinfo` command.

For example, a job array (with job ID 456) where you want to view the dependencies on the third element of the array:

```
bjdepinfo -c "456[3]"
```

JOBID	CHILD	CHILD_STATUS	CHILD_NAME	LEVEL
456[3]	300	PEND	job300	1

Display current job status

1. To display the status of the individual jobs submitted from a job array, specify the job array job ID with `bj obs`. For jobs submitted from a job array, `JOBID` displays the job array job ID, and `JOBNAME` displays the job array name and the index value of each job.

For example, to view a job array with job ID 123:

```
bjobs 123
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT TIME
123	user1	DONE	default	hostA	hostC	myArray[1]	Feb 29 12: 34
123	user1	DONE	default	hostA	hostQ	myArray[2]	Feb 29 12: 34
123	user1	DONE	default	hostA	hostB	myArray[3]	Feb 29 12: 34
123	user1	RUN	default	hostA	hostC	myArray[4]	Feb 29 12: 34
123	user1	RUN	default	hostA	hostL	myArray[5]	Feb 29 12: 34
123	user1	RUN	default	hostA	hostB	myArray[6]	Feb 29 12: 34
123	user1	RUN	default	hostA	hostQ	myArray[7]	Feb 29 12: 34
123	user1	PEND	default	hostA		myArray[8]	Feb 29 12: 34
123	user1	PEND	default	hostA		myArray[9]	Feb 29 12: 34

```
123      user1  PEND    default  hostA      myArray[10] Feb 29 12:34
```

Display past job status

1. To display the past status of the individual jobs submitted from a job array, specify the job array job ID with `bhist`.

For example, to view the history of a job array with job ID 456:

bhist 456

Summary of time in seconds spent in various states:

JOBID	USER	JOB_NAME	PEND	PSUSP	RUN	USUSP	SSUSP	UNKWN	TOTAL
456[1]	user1	*rray[1]	14	0	65	0	0	0	79
456[2]	user1	*rray[2]	74	0	25	0	0	0	99
456[3]	user1	*rray[3]	121	0	26	0	0	0	147
456[4]	user1	*rray[4]	167	0	30	0	0	0	197
456[5]	user1	*rray[5]	214	0	29	0	0	0	243
456[6]	user1	*rray[6]	250	0	35	0	0	0	285
456[7]	user1	*rray[7]	295	0	33	0	0	0	328
456[8]	user1	*rray[8]	339	0	29	0	0	0	368
456[9]	user1	*rray[9]	356	0	26	0	0	0	382
456[10]	user1	*rray[10]	375	0	24	0	0	0	399

Display the current status of a specific job

1. To display the current status of a specific job submitted from a job array, specify in quotes, the job array job ID and an index value with `bjobs`.

For example, the status of the 5th job in a job array with job ID 123:

bjobs "123[5]"

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
123	user1	RUN	default	hostA	hostL	myArray[5]	Feb 29 12:34

Display the past status of a specific job

1. To display the past status of a specific job submitted from a job array, specify, in quotes, the job array job ID and an index value with `bhist`.

For example, the status of the 5th job in a job array with job ID 456:

bhist "456[5]"

Summary of time in seconds spent in various states:

JOBID	USER	JOB_NAME	PEND	PSUSP	RUN	USUSP	SSUSP	UNKWN	TOTAL
456[5]	user1	*rray[5]	214	0	29	0	0	0	243

Performance metric information

If you enable performance metric collection, every job submitted in a job array is counted individually, except for the `Job submission requests` metric. The entire job array counts as just one job submission request.

Control job arrays

You can control the whole array, all the jobs submitted from the job array, with a single command. LSF also provides the ability to control individual jobs and groups of jobs submitted from a job array. When issuing commands against a job array, use the job array job ID instead of the job array name. Job names are not unique in LSF, and issuing a command using a job array name may result in unpredictable behavior.

Most LSF commands allow operation on both the whole job array, individual jobs, and groups of jobs. These commands include `bkill`, `bstop`, `brresume`, and `bmod`.

Some commands only allow operation on individual jobs submitted from a job array. These commands include `btop`, `bbot`, and `bswitch`.

- Control a whole array
- Control individual jobs
- Control groups of jobs

Control a whole array

1. To control the whole job array, specify the command as you would for a single job using only the job ID.

For example, to kill a job array with job ID 123:

```
bkill 123
```

Control individual jobs

1. To control an individual job submitted from a job array, specify the command using the job ID of the job array and the index value of the corresponding job. The job ID and index value must be enclosed in quotes.

For example, to kill the 5th job in a job array with job ID 123:

```
bkill "123[5]"
```

Control groups of jobs

1. To control a group of jobs submitted from a job array, specify the command as you would for an individual job and use `indexList` syntax to indicate the jobs.

For example, to kill jobs 1-5, 239, and 487 in a job array with job ID 123:

```
bkill "123[1-5, 239, 487]"
```

Job array chunking

Job arrays in most queues can be chunked across an array boundary (not all jobs must belong to the same array). However, if the queue is preemptable or preemptive, the jobs are chunked when they belong to the same array.

For example:

```
job1[1], job1[2], job2[1], job2[2]  
in a preemption queue with CHUNK_JOB_SIZE=3
```

Then

- `job1[1]` and `job1[2]` are chunked.
- `job2[1]` and `job2[2]` are chunked.

Requeue jobs in DONE state

Use `brequeue` to requeue a job array. When the job is requeued, it is assigned the `PEND` status and the job's new position in the queue is after other jobs of the same priority.

1. To requeue `DONE` jobs use the `-d` option of `brequeue`.

For example, the command `brequeue -J "myarray[1-10]" -d 123` requeues jobs with job ID 123 and `DONE` status.

Note:

brequeue is not supported across clusters.

Requeue Jobs in EXIT state

1. To requeue EXIT jobs use the -e option of brequeue.

For example, the command `brequeue -J "myarray[1- 10] " -e 123` requeues jobs with job ID 123 and EXIT status.

Requeue all jobs in an array regardless of job state

1. A submitted job array can have jobs that have different job states. To requeue all the jobs in an array regardless of any job's state, use the -a option of brequeue.

For example, the command `brequeue -J "myarray[1- 10] " -a 123` requeues all jobs in a job array with job ID 123 regardless of their job state.

Requeue RUN jobs to PSUSP state

1. To requeue RUN jobs to PSUSP state, use the -H option of brequeue.

For example, the command `brequeue -J "myarray[1- 10] " -H 123` requeues to PSUSP RUN status jobs with job ID 123.

Requeue jobs in RUN state

1. To requeue RUN jobs use the -r option of brequeue.

For example, the command `brequeue -J "myarray[1- 10] " -r 123` requeues jobs with job ID 123 and RUN status.

Job array job slot limit

The job array job slot limit is used to specify the maximum number of jobs submitted from a job array that are allowed to run at any one time. A job array allows a large number of jobs to be submitted with one command, potentially flooding a system, and job slot limits provide a way to limit the impact a job array may have on a system. Job array job slot limits are specified using the following syntax:

```
bsub -J "job_array_name[index_list]%job_slot_limit" myJob
```

where:

%job_slot_limit

Specifies the maximum number of jobs allowed to run at any one time. The percent sign (%) must be entered exactly as shown. Valid values are positive integers less than the maximum index value of the job array.

Set a job array slot limit at submission

1. Use the bsub command to set a job slot limit at the time of submission.

To set a job array job slot limit of 100 jobs for a job array of 1000 jobs:

```
bsub -J "job_array_name[1000]%100" myJob
```

Set a job array slot limit after submission

1. Use the `bmod` command to set a job slot limit after submission.

For example, to set a job array job slot limit of 100 jobs for an array with job ID 123:

```
bmod -J "%100" 123
```

Change a job array job slot limit

Changing a job array job slot limit is the same as setting it after submission.

1. Use the `bmod` command to change a job slot limit after submission.

For example, to change a job array job slot limit to 250 for a job array with job ID 123:

```
bmod -J "%250" 123
```

View a job array job slot limit

1. To view job array job slot limits use the `-A` and `-l` options of `bjobs`. The job array job slot limit is displayed in the Job Name field in the same format in which it was set.

For example, the following output displays the job array job slot limit of 100 for a job array with job ID 123:

```
bjobs -A -l 123
```

```
Job <123>, Job Name <myArray[1-1000]%100>, User <user1>, Project <default>, Sta
tus <PEND>, Queue <normal>, Job Priority <20>, Command <my
Job>
```

```
Wed Feb 29 12:34:56 2010: Submitted from host <hostA>, CWD <$HOME>;
```

```
COUNTERS:
```

```
NJOBS  PEND  DONE  RUN  EXIT  SSUSP  USUSP  PSUSP
  10     9    0     1    0     0      0      0
```


Job Packs

Job packs overview

The purpose of this feature is to speed up the submission of a large number of jobs. When the feature is enabled, you can submit jobs by submitting a single file containing multiple job requests.

This feature supports all `bsub` options in the job submission file except for:

`-I -Ip -Is -IS -ISp -ISs -IX -XF -K -jsdl -h -V -pack`

About job packs

Enable / disable	Job packs are disabled by default. You must enable the feature before you can run <code>bsub -pack</code> .
Job submission rate	When you use the job packs feature to submit multiple jobs to <code>mbat chd</code> at once, instead of submitting the jobs individually, it minimizes system overhead and improves the overall job submission rate dramatically.
Job submission file	When you use this feature, you create a job submission file that defines each job request. You specify all the <code>bsub</code> options individually for each job, so unlike chunk jobs and job arrays, there is no need for jobs in this file to have anything in common. To submit the jobs to LSF, you simply submit the file using the <code>bsub -pack</code> option.
Job pack	<p>LSF parses the file contents and submits the job requests to <code>mbat chd</code>, sending multiple requests at one time. Each group of jobs submitted to <code>mbat chd</code> together is called a job pack. The job submission file can contain any number of job requests, and LSF will group them into job packs automatically. The reason to group jobs into packs is to maintain proper <code>mbat chd</code> performance: while <code>mbat chd</code> is processing a job pack, <code>mbat chd</code> is blocked from processing other requests, so limiting the number of jobs in each pack ensures a reasonable <code>mbat chd</code> response time for other job submissions. Job pack size is configurable.</p> <p>If the cluster configuration is not consistent, and <code>mbat chd</code> receives a job pack that exceeds the job pack size defined in <code>lsf.conf</code>, it will be rejected.</p>
Job request	Once the pack is submitted to <code>mbat chd</code> , each job request in the pack is handled by LSF as if it was submitted individually with the <code>bsub</code> command.

For example:

- If `BSUB_CHK_RESREQ` is enabled, LSF checks the syntax of the resource requirement string, instead of scheduling the job.
- If `-is` or `-Zs` is specified, LSF copies the command file to the spool directory, and this may affect the job submission rate.
- The job request cannot be submitted to `mbatchd` if the pending job threshold has been reached (`MAX_PEND_JOBS` in `lsb.params`).
- If `BSUB_QUIET` is enabled, LSF will not print information about successful job submission.

Job submission errors

By default, if any job request in a file cannot be submitted to `mbatchd`, LSF assumes the job submission file has become corrupt, and does not process any more requests from the file (the jobs already submitted to `mbatchd` successfully do continue to run). Optionally, you can modify the configuration and change this. If you do, LSF processes every request in the file and attempts to submit all the jobs, even if some previous job submissions have failed.

For example, the job submission file may contain job requests from many users, but the default behavior is that LSF stops processing requests after one job fails because the pending job threshold for the user has been reached. If you change the configuration, processing of the job submission file can continue, and job requests from other users can run.

mesub

By default, LSF runs `mesub` as usual for all jobs in the file. Optionally, you can modify configuration and change this. If you do, LSF processes the jobs in the file without running any `mesub`, even if there are `esubs` configured at the application level (`-a` option of `bsub`), or using `LSB_ESUB_METHOD` in `lsb.conf`, or through a named `esub` executable under `LSF_SERVERDIR`.

The `esub` is never executed.

Enable and configure job packs

1. Edit `lsb.conf`.

These parameters will be ignored if defined in the environment instead of the `lsb.conf` file.

2. Define the parameter `LSB_MAX_PACK_JOBS=100`.

Do this to enable the feature and set the job pack size. We recommend 100 as the initial pack size.

If the value is 1, jobs from the file are submitted individually, as if submitted directly using the `bsub` command.

If the value is 0, job packs are disabled.

3. Optionally, define the parameter `LSB_PACK_MESUB=N`.

Do this if you want to further increase the job submission rate by preventing the execution of any `mesub` during job submission.

This parameter only affects the jobs submitted using job packs, it does not affect jobs submitted in the usual way.

4. Optionally, define the parameter `LSB_PACK_SKIP_ERROR=Y`.

Do this if you want LSF to process all requests in a job submission file, and continue even if some requests have errors.

- Restart `mbatchd` to make your changes take effect.

Submit job packs

- Prepare the job submission file.

Prepare a text file containing all the jobs you want to submit. Each line in the file is one job request. For each request, the syntax is identical to the `bsub` command line (without the word "bsub").

For example:

```
#This file contains 2 job requests.
-R "select[mem>200] rusage[mem=100]" job1.sh
-R "select[swap>400] rusage[swap=200]" job2.sh
#end
```

The job submission file has the following limitations:

- The following `bsub` options are not supported:
 - `-I -Ip -Is -IS -ISp -ISs -IX -XF -K -j sdl -h -V -pack`
- Terminal Services jobs are not supported.
- I/O redirection is not supported.
- Blank lines and comment lines (beginning with #) are ignored. Comments at the end of a line are not supported.
- Backslash (\) is NOT considered a special character to join two lines.
- Shell scripting characters are treated as plain text, they will not be interpreted.
- Matched pairs of single and double quotations are supported, but they must have space before and after. For example, `-J "job1"` is supported, `-J"job1"` is not, and `-J "job" 1` is not.

For job dependencies, job name is recommended instead of job ID to specify the dependency condition. A job request will be rejected if the job name or job ID of the job it depends on does not already exist.

- Submit the file.

Use the `bsub -pack` option to submit all the jobs in a file. Run:

```
bsub -pack job_submission_file
```

where *job_submission_file* is the full path to the job submission file. Do not put any other `bsub` options in the command line, they must be included in each individual job request in the file.

The `-pack` option is not supported in a job script.

Performance metrics

If you enable performance metric collection, every job submitted in a job pack is counted individually, except for the `Job submission requests` metric. Each job pack counts as just one job submission request.

45

Running Parallel Jobs

How Platform LSF runs parallel jobs

When LSF runs a job, the `LSB_HOSTS` variable is set to the names of the hosts running the batch job. For a parallel batch job, `LSB_HOSTS` contains the complete list of hosts that LSF has allocated to that job.

LSF starts one controlling process for the parallel batch job on the first host in the host list. It is up to your parallel application to read the `LSB_HOSTS` environment variable to get the list of hosts, and start the parallel job components on all the other allocated hosts.

LSF provides a generic interface to parallel programming packages so that any parallel package can be supported by writing shell scripts or wrapper programs.

Preparing your environment to submit parallel jobs to Platform LSF

Getting the host list

Some applications can take this list of hosts directly as a command line parameter. For other applications, you may need to process the host list.

Example

The following example shows a `/bin/sh` script that processes all the hosts in the host list, including identifying the host where the job script is executing.

```
#!/bin/sh
# Process the list of host names in LSB_HOSTS
for host in $LSB_HOSTS ; do
  handle_host $host
done
```

Parallel job scripts

Each parallel programming package has different requirements for specifying and communicating with all the hosts used by a parallel job. LSF is not tailored to work with a specific parallel programming package. Instead, LSF provides a generic interface so that any parallel package can be supported by writing shell scripts or wrapper programs.

You can modify these scripts to support more parallel packages.

Use a job starter

You can configure the script into your queue as a job starter, and then all users can submit parallel jobs without having to type the script name.

1. To see if your queue already has a job starter defined, run `bqueues -l`.

Submit a parallel job

LSF can allocate more than one slot to run a job and automatically keeps track of the job status, while a parallel job is running.

When submitting a parallel job that requires multiple slots, you can specify the exact number of slots to use.

1. To submit a parallel job, use `bsub -n` and specify the number of slots the job requires.
2. To submit jobs based on the number of available job slots instead of the number of CPUs, use `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`.

For example:

```
bsub -n 4 myjob
```

The job `myjob` submits as a parallel job. The job is started when four job slots are available.

Note:

When `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the resource requirement string keyword `ncpus` refers to the number of slots instead of the number of CPUs however `lshosts` output will continue to show `ncpus` as defined by `EGO_DEFINE_NCPUS` in `lsf.conf`.

Start parallel tasks with Platform LSF utilities

For simple parallel jobs you can use LSF utilities to start parts of the job on other hosts. Because LSF utilities handle signals transparently, LSF can suspend and resume all components of your job without additional programming.

Run parallel tasks with lsrgrun

The simplest parallel job runs an identical copy of the executable on every host. The `lsrgrun` command takes a list of host names and runs the specified task on each host. The `lsrgrun -p` command specifies that the task should be run in parallel on each host.

Example

This example submits a job that uses `lsrgrun` to run `myjob` on all the selected hosts in parallel:

```
bsub -n 10 'lsrgrun -p -m "$LSB_HOSTS" myjob'  
Job <3856> is submitted to default queue <normal>.
```

For more complicated jobs, you can write a shell script that runs `lsrgrun` in the background to start each component.

Run parallel tasks with the blaunch distributed application framework

Most MPI implementations and many distributed applications use `rsh` and `ssh` as their task launching mechanism. The `blaunch` command provides a drop-in replacement for `rsh` and `ssh` as a transparent method for launching parallel and distributed applications within LSF.

Similar to the `lsrgrun` command, `blaunch` transparently connects directly to the RES/SBD on the remote host, and subsequently creates and tracks the remote tasks, and provides the connection back to LSF. There is no need to insert `pam` or `taskstarter` into the `rsh` or `ssh` calling sequence, or configure any wrapper scripts.

Important:

You cannot run `blaunch` directly from the command line.

`blaunch` only works within an LSF job; it can only be used to launch tasks on remote hosts that are part of a job allocation. It cannot be used as a standalone command. On success `blaunch` exits with 0.

Windows: `blaunch` is supported on Windows 2000 or later with the following exceptions:

- Only the following signals are supported: SIGKILL, SIGSTOP, SIGCONT.
- The `-n` option is not supported.
- `CMD.EXE /C <user command line>` is used as intermediate command shell when: `-no-shell` is not specified
- `CMD.EXE /C` is not used when `-no-shell` is specified.
- Windows Vista User Account Control must be configured correctly to run jobs.

Submit jobs with blaunch

Use `bsub` to call `blaunch`, or to invoke a job script that calls `blaunch`. The `blaunch` command assumes that `bsub -n` implies one remote task per job slot.

- **Submit a parallel job:**

```
bsub -n 4 blaunch myjob
```

- **Submit a parallel job to launch tasks on a specific host:**

```
bsub -n 4 blaunch hostA myjob
```

- **Submit a job with a host list:**

```
bsub -n 4 blaunch -z "hostA hostB" myjob
```

- **Submit a job with a host file:**

```
bsub -n 4 blaunch -u ./hostfile myjob
```

- **Submit a job to an application profile**

```
bsub -n 4 -app pjob blaunch myjob
```

Job slot limits for parallel jobs

A job slot is the basic unit of processor allocation in LSF. A sequential job uses one job slot. A parallel job that has N components (tasks) uses N job slots, which can span multiple hosts.

By default, running and suspended jobs count against the job slot limits for queues, users, hosts, and processors that they are associated with.

With processor reservation, job slots reserved by pending jobs also count against all job slot limits.

When backfilling occurs, the job slots used by backfill jobs count against the job slot limits for the queues and users, but not hosts or processors. This means when a pending job and a running job occupy the same physical job slot on a host, both jobs count towards the queue limit, but only the pending job counts towards host limit.

Specify a minimum and maximum number of processors

By default, when scheduling a parallel job, the number of slots allocated on each host will not exceed the number of CPUs on that host even though host MXJ is set greater than number of CPUs. When submitting a parallel job, you can also specify a minimum number and a maximum number of processors.

If you specify a maximum and minimum number of processors, the job starts as soon as the minimum number of processors is available, but it uses up to the maximum number of processors, depending on how many processors are available at the time. Once the job starts running, no more processors are allocated to it even though more may be available later on.

Jobs that request fewer processors than the minimum PROCLIMIT defined for the queue or application profile to which the job is submitted, or more processors than the maximum PROCLIMIT are rejected. If the job requests minimum and maximum processors, the maximum requested cannot be less than the minimum PROCLIMIT, and the minimum requested cannot be more than the maximum PROCLIMIT.

If `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the job specifies a maximum and minimum number of job slots instead of processors. LSF ignores the number of CPUs constraint during parallel job scheduling and only schedules based on slots.

If `PARALLEL_SCHED_BY_SLOT` is not defined for a resizable job, individual allocation requests are constrained by the number of CPUs during scheduling. However, the final resizable job allocation may not agree. For example, if an autoresizable job requests 1 to 4 slots, on a 2 CPUs 4 slots box, an autoresizable job eventually will use up to 4 slots.

Syntax

```
bsub -n min_proc [,max_proc]
```

Example

```
bsub -n 4,16 myjob
```

At most, 16 processors can be allocated to this job. If there are less than 16 processors eligible to run the job, this job can still be started as long as the number of eligible processors is greater than or equal to 4.

About specifying a first execution host

In general, the first execution host satisfies certain resource requirements that might not be present on other available hosts.

By default, LSF selects the first execution host dynamically according to the resource availability and host load for a parallel job. Alternatively, you can specify one or more first execution host candidates so that LSF selects one of the candidates as the first execution host.

When a first execution host is specified to run the first task of a parallel application, LSF does not include the first execution host or host group in a job resize allocation request.

Specify a first execution host

1. To specify one or more hosts, host groups, or compute units as first execution host candidates, add the (!) symbol after the host name.

You can specify first execution host candidates at job submission, or in the queue definition.

Job level

1. Use the `-m` option of `bsub`:

```
bsub -n 32 -m "hostA! hostB hostgroup1! hostC" myjob
```

The scheduler selects either hostA or a host defined in hostgroup1 as the first execution host, based on the job's resource requirements and host availability.

2. In a MultiCluster environment, insert the (!) symbol after the cluster name, as shown in the following example:

```
bsub -n 2 -m "host2@cluster2! host3@cluster2" my_parallel_job
```

Queue level

The queue-level specification of first execution host candidates applies to all jobs submitted to the queue.

1. Specify the first execution host candidates in the list of hosts in the `HOSTS` parameter in `lsb.queues`:

```
HOSTS = hostA! hostB hostgroup1! hostC
```

Rules

Follow these guidelines when you specify first execution host candidates:

- If you specify a host group or compute unit, you must first define the host group or compute unit in the file `lsb.hosts`.
- Do not specify a dynamic host group as a first execution host.
- Do not specify "all," "allremote," or "others," or a host partition as a first execution host.
- Do not specify a preference (+) for a host identified by (!) as a first execution host candidate.
- For each parallel job, specify enough regular hosts to satisfy the CPU requirement for the job. Once LSF selects a first execution host for the current job, the other first execution host candidates
 - Become unavailable to the current job
 - Remain available to other jobs as either regular or first execution hosts
- You cannot specify first execution host candidates when you use the `brun` command.

If the first execution host is incorrect at job submission, the job is rejected. If incorrect configurations exist on the queue level, warning messages are logged and displayed when LSF starts, restarts or is reconfigured.

Job chunking

Specifying first execution host candidates affects job chunking. For example, the following jobs have different job requirements, and is not placed in the same job chunk:

```
bsub -n 2 -m "hostA! hostB hostC" myjob
bsub -n 2 -m "hostA hostB hostC" myjob
bsub -n 2 -m "hostA hostB! hostC" myjob
```

The requirements of each job in this example are:

- Job 1 must start on hostA
- Job 2 can start and run on hostA, hostB, or hostC
- Job 3 must start on hostB

For job chunking, all jobs must request the same hosts *and* the same first execution hosts (if specified). Jobs that specify a host preference must all specify the same preference.

Resource reservation

If you specify first execution host candidates at the job or queue level, LSF tries to reserve a job slot on the first execution host. If LSF cannot reserve a first execution host job slot, it does not reserve slots on any other hosts.

Compute units

If compute units resource requirements are used, the compute unit containing the first execution host is given priority:

```
bsub -n 64 -m "hg! cu1 cu2 cu3 cu4" -R "cu[pref=config]" myjob
```

In this example the first execution host is selected from the host group hg. Next in the job's allocation list are any appropriate hosts from the same compute unit as the first execution host. Finally remaining hosts are grouped by compute unit, with compute unit groups appearing in the same order as in the ComputeUnit section of lsb. hosts.

Compound resource requirements

If compound resource requirements are being used, the resource requirements specific to the first execution host should appear first:

```
bsub -m "hostA! hg12" -R "1*{select[type==X86_64]rusage[licA=1]} + {select[type==any]}" myjob
```

In this example the first execution host must satisfy: `select[type==X86_64]rusage[licA=1]`

Control job locality using compute units

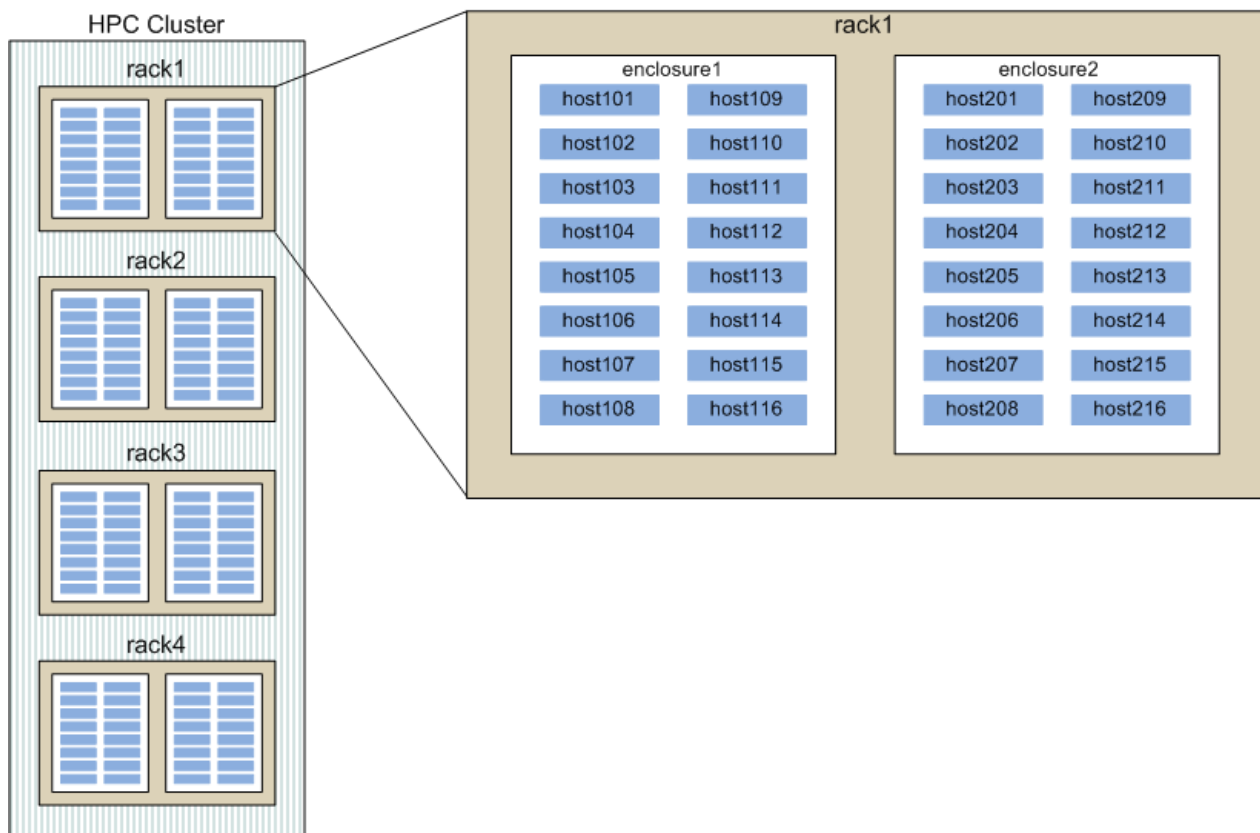
Compute units are groups of hosts laid out by the LSF administrator and configured to mimic the network architecture, minimizing communications overhead for optimal placement of parallel jobs. Different granularities of compute units provide the flexibility to configure an extensive cluster accurately and run larger jobs over larger compute units.

Resource requirement keywords within the compute unit section can be used to allocate resources throughout compute units in manner analogous to host resource allocation. Compute units then replace hosts as the basic unit of allocation for a job.

High performance computing clusters running large parallel jobs spread over many hosts benefit from using compute units. Communications bottlenecks within the network architecture of a large cluster can be isolated through careful configuration of compute units. Using compute units instead of hosts as the basic allocation unit, scheduling policies can be applied on a large scale.

Note:

Configure each individual host as a compute unit to use the compute unit features for host level job allocation.



As indicated in the picture, two types of compute units have been defined in the parameter `COMPUTE_UNIT_TYPES` in `lsb.params`:

COMPUTE_UNIT_TYPES= enclosure! rack

! indicates the default compute unit type. The first type listed (enclosure) is the finest granularity and the only type of compute unit containing hosts and host groups. Coarser granularity rack compute units can only contain enclosures.

The hosts have been grouped into compute units in the ComputeUnit section of lsb.hosts as follows (some lines omitted):

NAME	MEMBER	CONDENSED	TYPE
enclosure1	(host1[01-16])	Y	enclosure
...			
enclosure8	(host8[01-16])	Y	enclosure
rack1	(enclosure[1-2])	Y	rack
rack2	(enclosure[3-4])	Y	rack
rack3	(enclosure[5-6])	Y	rack
rack4	(enclosure[7-8])	Y	rack

This example defines 12 compute units, all of which have condensed output:

- enclosure1 through enclosure8 are the finest granularity, and each contain 16 hosts.
- rack1, rack2, rack3, and rack4 are the coarsest granularity, and each contain 2 enclosures.

Syntax

The cu string supports the following syntax:

cu[balance]

All compute units used for this job should contribute the same number of slots (to within one slot). Provides a balanced allocation over the fewest possible compute units.

cu[pref=config]

Compute units for this job are considered in the order they appear in the lsb.hosts configuration file. This is the default value.

cu[pref=minavail]

Compute units with the fewest available slots are considered first for this job. Useful for smaller jobs (both sequential and parallel) since this reduces fragmentation of compute units, leaving whole compute units free for larger jobs.

cu[pref=maxavail]

Compute units with the most available slots are considered first for this job.

cu[maxcus=number]

Maximum number of compute units the job can run across.

cu[usablecuslots=number]

All compute units used for this job should contribute the same minimum *number* of slots. At most the final allocated compute unit can contribute fewer than *number* slots.

cu[type=cu_type]

Type of compute unit being used, where *cu_type* is one of the types defined by COMPUTE_UNIT_TYPES in lsb.params. The default is the compute unit type listed first in lsb.params.

cu[excl]

Compute units used exclusively for the job. Must be enabled by EXCLUSIVE in lsb. queues.

Continuing with the example shown above, assume lsb. queues contains the parameter definition **EXCLUSIVE=CU[rack]** and that the slots available for each compute unit are shown under MAX in the condensed display from bhost.s, where HOST_NAME refers to the compute unit:

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
encl osure1	ok	-	64	34	34	0	0	0
encl osure2	ok	-	64	54	54	0	0	0
encl osure3	ok	-	64	46	46	0	0	0
encl osure4	ok	-	64	44	44	0	0	0
encl osure5	ok	-	64	45	45	0	0	0
encl osure6	ok	-	64	44	44	0	0	0
encl osure7	ok	-	32	0	0	0	0	0
encl osure8	ok	-	64	0	0	0	0	0
rack1	ok	-	128	88	88	0	0	0
rack2	ok	-	128	90	90	0	0	0
rack3	ok	-	128	89	89	0	0	0
rack4	ok	-	128	0	0	0	0	0

Based on the 12 configured compute units, jobs can be submitted with a variety of compute unit requirements.

Use compute units

1. **bsub -R "cu[]" -n 64 ./app**

This job is restricted to compute units of the default type encl osure. The default pref=config applies, with compute units considered in configuration order. The job runs on 30 slots in encl osure1, 10 slots in encl osure2, 8 slots in encl osure3, and 16 slots in encl osure4 for a total of 64 slots.

2. Compute units can be considered in order of most free slots or fewest free slots, where free slots include any slots available and not occupied by a running job.

bsub -R "cu[pref=minavail]" -n 32 ./app

This job is restricted to compute units of the default type encl osure in the order pref=minavail. Compute units with the fewest free slots are considered first. The job runs on 10 slots in encl osure2, 18 slots in encl osure3 and 3 slots in encl osure5 for a total of 32 slots.

3. **bsub -R "cu[type=rack:pref=maxavail]" -n 64 ./app**

This job is restricted to compute units of the default type encl osure in the order pref=maxavail. Compute units with the most free slots are considered first. The job runs on 64 slots in encl osure8.

Localized allocations

Jobs can be run over a limited number of compute units using the maxcus keyword.

1. **bsub -R "cu[pref=maxavail:maxcus=1]" ./app**

This job is restricted to a single enclosure, and compute units with the most free slots are considered first. The job requirements are satisfied by encl osure8 which has 64 free slots.

2. **bsub -n 64 -R "cu[maxcus=3]" ./app**

This job requires a total of 64 slots over 3 enclosures or less. Compute units are considered in configuration order. The job requirements are satisfied by the following allocation:

compute unit	free slots
enclosure1	30
enclosure3	18
enclosure4	16

Balanced slot allocations

Balanced allocations split jobs evenly between compute units, which increases the efficiency of some applications.

1. **bsub -n 80 -R "cu[balance:maxcus=4]" ./app**

This job requires a balanced allocation over the fewest possible compute units of type enclosure (the default type), with a total of 80 slots. Since none of the configured enclosures have 80 slots, 2 compute units with 40 slots each are used, satisfying the maxcus requirement to use 4 compute units or less.

The keyword pref is not included so the default order of pref=config is used. The job requirements are satisfied by 40 slots on both enclosure7 and enclosure8 for a total of 80 slots.

2. **bsub -n 64 -R "cu[balance:type=rack:pref=maxavail]" ./app**

This job requires a balanced allocation over the fewest possible compute units of type rack, with a total of 64 slots. Compute units with the most free slots are considered first, in the order rack4, rack1, rack3, rack2. The job requirements are satisfied by rack4.

3. **bsub -n "40,80" -R "cu[balance:pref=minavail]" ./app**

This job requires a balanced allocation over compute units of type rack, with a range of 40 to 80 slots. Only the minimum number of slots is considered when a range is specified along with keyword balance, so the job needs 40 slots. Compute units with the fewest free slots are considered first.

Because balance uses the fewest possible compute units, racks with 40 or more slots are considered first, namely rack1 and rack4. The rack with the fewest available slots is then selected, and all job requirements are satisfied by rack1.

Balanced host allocations

Using balance and ptile together within the requirement string results in a balanced host allocation over compute units, and the same number of slots from each host. The final host may provide fewer slots if required.

• **bsub -n 64 -R "cu[balance] span[ptile=4]" ./app**

This job requires a balanced allocation over the fewest possible compute units of type enclosure, with a total of 64 slots. Each host used must provide 4 slots. Since enclosure8 has 64 slots available over 16 hosts (4 slots per host), it satisfies the job requirements.

Had enclosure8 not satisfied the requirements, other possible allocations in order of consideration (fewest compute units first) include:

number of compute units	number of hosts
2	8+8
3	5+5+6

number of compute units	number of hosts
4	4+4+4+4
5	3+3+3+3+4

Minimum slot allocations

Minimum slot allocations result in jobs spreading over fewer compute units, and ignoring compute units with few hosts available.

1. `bsub -n 45 -R "cu[usablecuslots=10:pref=minavail]" ./app`

This job requires an allocation of at least 10 slots in each enclosure, except possibly the last one. Compute units with the fewest free slots are considered first. The requirements are satisfied by a slot allocation of:

compute unit	number of slots
enclosure2	10
enclosure5	19
enclosure4	16

2. `bsub -n "1,140" -R "cu[usablecuslots=20]" ./app`

This job requires an allocation of at least 20 slots in each enclosure, except possibly the last one. Compute units are considered in configuration order and as close to 140 slots are allocated as possible. The requirements are satisfied by an allocation of 140 slots, where only the last compute unit has fewer than 20 slots allocated as follows:

compute unit	number of slots
enclosure1	30
enclosure4	20
enclosure6	20
enclosure7	64
enclosure2	6

Exclusive compute unit jobs

Because **EXCLUSIVE=CU[rack]** in lsb. queues, jobs may use compute units of type rack or finer granularity type enclosure exclusively. Exclusive jobs lock all compute units they run in, even if not all slots are being used by the job. Running compute unit exclusive jobs minimizes communications slowdowns resulting from shared network bandwidth.

1. `bsub -R "cu[excl:type=enclosure]" ./app`

This job requires exclusive use of an enclosure with compute units considered in configuration order. The first enclosure not running any jobs is enclosure7.

2. Using `excl` with `usablecuslots`, the job avoids compute units where a large portion of the hosts are unavailable.

`bsub -n 90 -R "cu[excl:usablecuslots=12:type=enclosure]" ./app`

This job requires exclusive use of compute units, and will not use a compute unit if fewer than 12 slots are available. Compute units are considered in configuration order. In this case the job requirements are satisfied by 64 slots in `encl osure7` and 26 slots in `encl osure8`.

3. **`bsub -R "cu[excl]" ./app`**

This job requires exclusive use of a rack with compute units considered in configuration order. The only rack not running any jobs is `rack4`.

Reservation

Compute unit constraints such as keywords `maxcus`, `balance`, and `excl` can result in inaccurately predicted start times from default LSF resource reservation. Time-based resource reservation provides a more accurate pending job predicted start time. When calculating job a time-based predicted start time, LSF considers job scheduling constraints and requirements, including job topology and resource limits, for example.

Host-level compute units

Configuring each individual host as a compute unit allows you to use the compute unit features for host level job allocation. Consider an example where one type of compute units has been defined in the parameter `COMPUTE_UNIT_TYPES` in `lsb.params`:

`COMPUTE_UNIT_TYPES= host!`

The hosts have been grouped into compute hosts in the `ComputeUnit` section of `lsb.hosts` as follows:

```
Begin ComputeUnit
NAME  MEMBER  TYPE
h1    host 1  host
h2    host 2  host
...
h50   host 50 host
End ComputeUnit
```

Each configured compute unit of default type `host` contains a single host.

Order host allocations

Using the compute unit keyword `pref`, hosts can be considered in order of most free slots or fewest free slots, where free slots include any slots available and not occupied by a running job:

1. **`bsub -R "cu[]" ./app`**

Compute units of default type `host`, each containing a single host, are considered in configuration order.

2. **`bsub -R "cu[pref=minavail]" ./app`**

Compute units of default type `host` each contain a single host. Compute units with the fewest free slots are considered first.

3. **`bsub -n 20 -R "cu[pref=maxavail]" ./app`**

Compute units of default type `host` each contain a single host. Compute units with the most free slots are considered first. A total of 20 slots are allocated for this job.

Limit hosts in allocations

Using the compute unit keyword `maxcus`, the maximum number of hosts allocated to a job can be set:

- **`bsub -n 12 -R "cu[pref=maxavail:maxcus=3]" ./app`**

Compute units of default type host each contain a single host. Compute units with the most free slots are considered first. This job requires an allocation of 12 slots over at most 3 hosts.

Balanced slot allocations

Using the compute unit keyword `balance`, jobs can be evenly distributed over hosts:

1. `bsub -n 9 -R "cu[balance]" ./app`

Compute units of default type host, each containing a single host, are considered in configuration order. Possible balanced allocations are:

compute units	hosts	number of slots per host
1	1	9
2	2	4, 5
3	3	3, 3, 3
4	4	2, 2, 2, 3
5	5	2, 2, 2, 2, 1
6	6	2, 2, 2, 1, 1, 1
7	7	2, 2, 1, 1, 1, 1, 1
8	8	2, 1, 1, 1, 1, 1, 1, 1
9	9	1, 1, 1, 1, 1, 1, 1, 1, 1

2. `bsub -n 9 -R "cu[balance:maxcus=3]" ./app`

Compute units of default type host, each containing a single host, are considered in configuration order. Possible balanced allocations are 1 host with 9 slots, 2 hosts with 4 and 5 slots, or 3 hosts with 3 slots each.

Minimum slot allocations

Using the compute unit keyword `usableslots`, hosts are only considered if they have a minimum number of slots free and usable for this job:

1. `bsub -n 16 -R "cu[usableslots=4]" ./app`

Compute units of default type host, each containing a single host, are considered in configuration order. Only hosts with 4 or more slots available and not occupied by a running job are considered. Each host (except possibly the last host allocated) must contribute at least 4 slots to the job.

2. `bsub -n 16 -R "rusage[mem=1000] cu[usableslots=4]" ./app`

Compute units of default type host, each containing a single host, are considered in configuration order. Only hosts with 4 or more slots available, not occupied by a running job, and with 1000 memory units are considered. A host with 10 slots and 2000 units of memory, for example, will only have 2 slots free that satisfy the memory requirements of this job.

Control processor allocation across hosts

Sometimes you need to control how the selected processors for a parallel job are distributed across the hosts in the cluster.

You can control this at the job level or at the queue level. The queue specification is ignored if your job specifies its own locality.

Specify parallel job locality at the job level

By default, LSF does allocate the required processors for the job from the available set of processors.

A parallel job may span multiple hosts, with a specifiable number of processes allocated to each host. A job may be scheduled on to a single multiprocessor host to take advantage of its efficient shared memory, or spread out on to multiple hosts to take advantage of their aggregate memory and swap space. Flexible spanning may also be used to achieve parallel I/O.

You are able to specify “select all the processors for this parallel batch job on the same host”, or “do not choose more than *n* processors on one host” by using the `span` section in the resource requirement string (`bsub -R` or `RES_REQ` in the queue definition in `lsb.queues`).

If `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the `span` string is used to control the number of job slots instead of processors.

Syntax

The `span` string supports the following syntax:

`span[hosts=1]`

Indicates that all the processors allocated to this job must be on the same host.

`span[ptile=value]`

Indicates the number of processors on each host that should be allocated to the job, where *value* is one of the following:

- Default `ptile` value, specified by *n* processors. In the following example, the job requests 4 processors on each available host, regardless of how many processors the host has:

```
span[ptile=4]
```

- Predefined `ptile` value, specified by '!'. The following example uses the predefined maximum job slot limit `lsb.hosts` (MXJ per host type/model) as its value:

```
span[ptile='!']
```

Tip:

If the host or host type/model does not define MXJ, the default predefined `ptile` value is 1.

- Predefined `ptile` value with optional multiple `ptile` values, per host type or host model:
 - For host type, you must specify `same[type]` in the resource requirement. In the following example, the job requests 8 processors on a host of type HP or SGI, and

2 processors on a host of type LINUX, and the predefined maximum job slot limit in `lsb.hosts (MXJ)` for other host types:

```
span[ptile='!', HP: 8, SGI: 8, LINUX: 2] same[type]
```

- For host model, you must specify `same[model]` in the resource requirement. In the following example, the job requests 4 processors on hosts of model PC1133, and 2 processors on hosts of model PC233, and the predefined maximum job slot limit in `lsb.hosts (MXJ)` for other host models:

```
span[ptile='!', PC1133: 4, PC233: 2] same[model]
```

span[hosts=-1]

Disables span setting in the queue. LSF allocates the required processors for the job from the available set of processors.

Specify multiple ptile values

In a span string with multiple `ptile` values, you must specify a predefined default value (`ptile='!'`) and either host model or host type.

You can specify both type and model in the same section in the resource requirement string, but the `ptile` values must be the same type.

If you specify `same[type: model]`, you *cannot* specify a predefined `ptile` value (!) in the span section.

Restriction:

Under bash 3.0, the exclamation mark (!) is not interpreted correctly by the shell. To use predefined ptile value (`ptile='!'`), use the `+H` option to disable '!' style history substitution in bash (`sh +H`).

The following span strings are valid:

```
same[type:model] span[ptile=LINUX:2,SGI:4]
```

LINUX and SGI are both host types and can appear in the same span string.

```
same[type:model] span[ptile=PC233:2,PC1133:4]
```

PC233 and PC1133 are both host models and can appear in the same span string.

You cannot mix host model and host type in the same span string. The following span strings are *not* correct:

```
span[ptile='!',LINUX:2,PC1133:4] same[model]
span[ptile='!',LINUX:2,PC1133:4] same[type]
```

The LINUX host type and PC1133 host model cannot appear in the same span string.

Multiple ptile values for a host type

For host type, you must specify `same[type]` in the resource requirement. For example:

```
span[ptile='!', HP: 8, SGI: 8, LINUX: 2] same[type]
```

The job requests 8 processors on a host of type HP or SGI, and 2 processors on a host of type LINUX, and the predefined maximum job slot limit in `lsb.hosts (MXJ)` for other host types.

Multiple ptile values for a host model

For host model, you must specify `same[model]` in the resource requirement. For example:

```
span[ptile='!', PC1133: 4, PC233: 2] same[model]
```

The job requests 4 processors on hosts of model PC1133, and 2 processors on hosts of model PC233, and the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host models.

Examples

```
bsub -n 4 -R "span[hosts=1]" myjob
```

Runs the job on a host that has at least 4 processors currently eligible to run the 4 components of this job.

```
bsub -n 4 -R "span[ptile=2]" myjob
```

Runs the job on 2 hosts, using 2 processors on each host. Each host may have more than 2 processors available.

```
bsub -n 4 -R "span[ptile=3]" myjob
```

Runs the job on 2 hosts, using 3 processors on the first host and 1 processor on the second host.

```
bsub -n 4 -R "span[ptile=1]" myjob
```

Runs the job on 4 hosts, even though some of the 4 hosts may have more than one processor currently available.

```
bsub -n 4 -R "type==any same[type] span[ptile='!',LINUX:2,SGI:4]" myjob
```

Submits myjob to request 4 processors running on 2 hosts of type LINUX (2 processors per host), or a single host of type SGI, or for other host types, the predefined maximum job slot limit in `lsb.hosts` (MXJ).

```
bsub -n 16 -R "type==any same[type] span[ptile='!',HP:8,SGI:8,LINUX:2]" myjob
```

Submits myjob to request 16 processors on 2 hosts of type HP or SGI (8 processors per hosts), or on 8 hosts of type LINUX (2 processors per host), or the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host types.

```
bsub -n 4 -R "same[model] span[ptile='!',PC1133:4,PC233:2]" myjob
```

Submits myjob to request a single host of model PC1133 (4 processors), or 2 hosts of model PC233 (2 processors per host), or the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host models.

Specify parallel job locality at the queue level

The queue may also define the locality for parallel jobs using the `RES_REQ` parameter.

Run parallel processes on homogeneous hosts

Parallel jobs run on multiple hosts. If your cluster has heterogeneous hosts some processes from a parallel job may for example, run on Solaris and some on SGI IRIX. However, for performance reasons you may want all processes of a job to run on the same type of host instead of having some processes run on one type of host and others on another type of host.

You can use the same section in the resource requirement string to indicate to LSF that processes are to run on one type or model of host. You can also use a custom resource to define the criteria for homogeneous hosts.

Run all parallel processes on the same host type

```
bsub -n 4 -R"select[type==SGI6 || type==SOL7] same[type]" myjob
```

Allocate 4 processors on the same host type—either SGI IRIX, or Solaris 7, but not both.

Run all parallel processes on the same host type and model

```
bsub -n 6 -R"select[type==any] same[type:model]" myjob
```

Allocate 6 processors on any host type or model as long as all the processors are on the same host type and model.

Run all parallel processes on hosts in the same high-speed connection group

```
bsub -n 12 -R "select[type==any && (hgconnect==hg1 || hgconnect==hg2 || hgconnect==hg3)] same [hgconnect:type]" myjob
```

For performance reasons, you want to have LSF allocate 12 processors on hosts in high-speed connection group hg1, hg2, or hg3, but not across hosts in hg1, hg2 or hg3 at the same time. You also want hosts that are chosen to be of the same host type.

This example reflects a network in which network connections among hosts in the same group are high-speed, and network connections between host groups are low-speed.

In order to specify this, you create a custom resource hgconnect in lsf.shared.

```
Begin Resource
RESOURCENAME  TYPE      INTERVAL  INCREASING  RELEASE  DESCRIPTION
hgconnect      STRING    ()         ()           ()       (OS release)
...
End Resource
```

In the lsf.cluster. *cluster_name* file, identify groups of hosts that share high-speed connections.

```
Begin ResourceMap
RESOURCENAME  LOCATION
hgconnect      (hg1@[hostA hostB] hg2@[hostD hostE] hg3@[hostF hostG hostX])
End ResourceMap
```

If you want to specify the same resource requirement at the queue level, define a custom resource in lsf.shared as in the previous example, map hosts to high-speed connection groups in lsf.cluster. *cluster_name*, and define the following queue in lsf.queues:

```
Begin Queue
QUEUE_NAME = My_test
PRIORITY = 30
NICE = 20 RES_REQ = "select[mem > 1000 && type==any && (hgconnect==hg1 || hgconnect==hg2 || hgconnect==hg3)]same[hgconnect:type]"
DESCRIPTION = either hg1 or hg2 or hg3
End Queue
```

This example allocates processors on hosts that:

- Have more than 1000 MB in memory
- Are of the same host type
- Are in high-speed connection group hg1 or hg2 or hg3

Limit the number of processors allocated

Use the PROCLIMIT parameter in l sb. queues or l sb. appl i cat i ons to limit the number of processors that can be allocated to a parallel job.

Syntax

PROCLIMIT = [*minimum_limit* [*default_limit*]] *maximum_limit*

All limits must be positive numbers greater than or equal to 1 that satisfy the following relationship:

$$1 \leq \textit{minimum} \leq \textit{default} \leq \textit{maximum}$$

You can specify up to three limits in the PROCLIMIT parameter:

If you specify ...	Then ...
One limit	It is the maximum processor limit. The minimum and default limits are set to 1.
Two limits	The first is the minimum processor limit, and the second is the maximum. The default is set equal to the minimum. The minimum must be less than or equal to the maximum.
Three limits	The first is the minimum processor limit, the second is the default processor limit, and the third is the maximum. The minimum must be less than the default and the maximum.

How PROCLIMIT affects submission of parallel jobs

The -n option of bsub specifies the number of processors to be used by a parallel job, subject to the processor limits of the queue or application profile.

Jobs that specify fewer processors than the minimum PROCLIMIT or more processors than the maximum PROCLIMIT are rejected.

If a default value for PROCLIMIT is specified, jobs submitted without specifying -n use the default number of processors. If the queue or application profile has only minimum and maximum values for PROCLIMIT, the number of processors is equal to the minimum value. If only a maximum value for PROCLIMIT is specified, or no PROCLIMIT is specified, the number of processors is equal to 1.

Incorrect processor limits are ignored, and a warning message is displayed when LSF is reconfigured or restarted. A warning message is also logged to the mbat chd log file when LSF is started.

Change PROCLIMIT

If you change the PROCLIMIT parameter, the new processor limit does not affect running jobs. Pending jobs with no processor requirements use the new default PROCLIMIT value. If the pending job does not satisfy the new processor limits, it remains in PEND state, and the pending reason changes to the following:

Job no longer satisfies PROCLIMIT configuration

If PROCLIMIT specification is incorrect (for example, too many parameters), a reconfiguration error message is issued. Reconfiguration proceeds and the incorrect PROCLIMIT is ignored.

MultiCluster

Jobs forwarded to a remote cluster are subject to the processor limits of the remote queues. Any processor limits specified on the local cluster are not applied to the remote job.

Resizable jobs

Resizable job allocation requests obey the PROCLIMIT definition in both application profiles and queues. When the maximum job slot request is greater than the maximum slot definition in PROCLIMIT, LSF chooses the minimum value of both. For example, if a job asks for `-n 1, 4`, but PROCLIMIT is defined as `2 2 3`, the maximum slot request for the job is 3 rather than 4.

Automatic queue selection

When you submit a parallel job without specifying a queue name, LSF automatically selects the most suitable queue from the queues listed in the `DEFAULT_QUEUE` parameter in `lsb.params` or the `LSB_DEFAULTQUEUE` environment variable. Automatic queue selection takes into account any maximum and minimum PROCLIMIT values for the queues available for automatic selection.

If you specify `-n min_proc,max_proc`, but do not specify a queue, the first queue that satisfies the processor requirements of the job is used. If no queue satisfies the processor requirements, the job is rejected.

For example, queues with the following PROCLIMIT values are defined in `lsb.queues`:

- queueA with PROCLIMIT=1 1 1
- queueB with PROCLIMIT=2 2 2
- queueC with PROCLIMIT=4 4 4
- queueD with PROCLIMIT=8 8 8
- queueE with PROCLIMIT=16 16 16

In `lsb.params`: `DEFAULT_QUEUE=queueA queueB queueC queueD queueE`

For the following jobs:

`bsub -n 8 myjob`

LSF automatically selects queueD to run myjob.

`bsub -n 5 myjob`

Job myjob fails because no default queue has the correct number of processors.

Maximum processor limit

PROCLIMIT is specified in the default queue in `lsb.queues` as:

```
PROCLIMIT = 3
```

The maximum number of processors that can be allocated for this queue is 3.

Example	Description
<code>bsub -n 2 myjob</code>	The job myjob runs on 2 processors.
<code>bsub -n 4 myjob</code>	The job myjob is rejected from the queue because it requires more than the maximum number of processors configured for the queue (3).
<code>bsub -n 2,3 myjob</code>	The job myjob runs on 2 or 3 processors.
<code>bsub -n 2,5 myjob</code>	The job myjob runs on 2 or 3 processors, depending on how many slots are currently available on the host.
<code>bsub myjob</code>	No default or minimum is configured, so the job myjob runs on 1 processor.

Minimum and maximum processor limits

PROCLIMIT is specified in l sb. queues as:

```
PROCLIMIT = 3 8
```

The minimum number of processors that can be allocated for this queue is 3 and the maximum number of processors that can be allocated for this queue is 8.

Example	Description
bsub -n 5 myjob	The job myj ob runs on 5 processors.
bsub -n 2 myjob	The job myj ob is rejected from the queue because the number of processors requested is less than the minimum number of processors configured for the queue (3).
bsub -n 4,5 myjob	The job myj ob runs on 4 or 5 processors.
bsub -n 2,6 myjob	The job myj ob runs on 3 to 6 processors.
bsub -n 4,9 myjob	The job myj ob runs on 4 to 8 processors.
bsub myjob	The default number of processors is equal to the minimum number (3). The job myj ob runs on 3 processors.

Minimum, default, and maximum processor limits

PROCLIMIT is specified in l sb. queues as:

```
PROCLIMIT = 4 6 9
```

- Minimum number of processors that can be allocated for this queue is 4
- Default number of processors for the queue is 6
- Maximum number of processors that can be allocated for this queue is 9

Example	Description
bsub myjob	Because a default number of processors is configured, the job myj ob runs on 6 processors.

Reserve processors

About processor reservation

When parallel jobs have to compete with sequential jobs for job slots, the slots that become available are likely to be taken immediately by a sequential job. Parallel jobs need multiple job slots to be available before they can be dispatched. If the cluster is always busy, a large parallel job could be pending indefinitely. The more processors a parallel job requires, the worse the problem is.

Processor reservation solves this problem by reserving job slots as they become available, until there are enough reserved job slots to run the parallel job.

You might want to configure processor reservation if your cluster has a lot of sequential jobs that compete for job slots with parallel jobs.

How processor reservation works

Processor reservation is disabled by default.

If processor reservation is enabled, and a parallel job cannot be dispatched because there are not enough job slots to satisfy its minimum processor requirements, the job slots that are currently available is reserved and accumulated.

A reserved job slot is unavailable to any other job. To avoid deadlock situations in which the system reserves job slots for multiple parallel jobs and none of them can acquire sufficient resources to start, a parallel job gives up all its reserved job slots if it has not accumulated enough to start within a specified time. The reservation time starts from the time the first slot is reserved. When the reservation time expires, the job cannot reserve any slots for one scheduling cycle, but then the reservation process can begin again.

If you specify first execution host candidates at the job or queue level, LSF tries to reserve a job slot on the first execution host. If LSF cannot reserve a first execution host job slot, it does not reserve slots on any other hosts.

Configure processor reservation

1. To enable processor reservation, set `SLOT_RESERVE` in `l sb. queues` and specify the reservation time.

A job cannot hold any reserved slots after its reservation time expires.

`SLOT_RESERVE=MAX_RESERVE_TIME[n]`.

where n is an integer by which to multiply `MBD_SLEEP_TIME`. `MBD_SLEEP_TIME` is defined in `l sb. params`; the default value is 60 seconds.

For example:

```
Begin Queue
.
PJOB_LIMIT=1
SLOT_RESERVE = MAX_RESERVE_TIME[ 5]
.
End Queue
```

In this example, if `MBD_SLEEP_TIME` is 60 seconds, a job can reserve job slots for 5 minutes. If `MBD_SLEEP_TIME` is 30 seconds, a job can reserve job slots for $5 * 30 = 150$ seconds, or 2.5 minutes.

View information about reserved job slots

1. Display reserved slots using `bj obs`.

The number of reserved slots can be displayed with the `bqueues`, `bhosts`, `bhpart`, and `busers` commands. Look in the `RSV` column.

Reserve memory for pending parallel jobs

By default, the `rusage` string reserves resources for running jobs. Because resources are not reserved for pending jobs, some memory-intensive jobs could be pending indefinitely because smaller jobs take the resources immediately before the larger jobs can start running. The more memory a job requires, the worse the problem is.

Memory reservation for pending jobs solves this problem by reserving memory as it becomes available, until the total required memory specified on the `rusage` string is accumulated and the job can start. Use memory reservation for pending jobs if memory-intensive jobs often compete for memory with smaller jobs in your cluster.

Unlike slot reservation, which only applies to parallel jobs, memory reservation applies to both sequential and parallel jobs.

Configure memory reservation for pending parallel jobs

You can reserve host memory for pending jobs.

1. Set the `RESOURCE_RESERVE` parameter in a queue defined in `lsb. queues`.

The `RESOURCE_RESERVE` parameter overrides the `SLOT_RESERVE` parameter. If both `RESOURCE_RESERVE` and `SLOT_RESERVE` are defined in the same queue, job slot reservation and memory reservation are enabled and an error is displayed when the cluster is reconfigured. `SLOT_RESERVE` is ignored. Backfill on memory may still take place.

The following queue enables both memory reservation and backfill in the same queue:

```
Begin Queue
QUEUE_NAME = reservation_backfill
DESCRIPTION = For resource reservation and backfill
PRIORITY = 40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
BACKFILL = Y
End Queue
```

Enable per-slot memory reservation

By default, memory is reserved for parallel jobs on a per-host basis. For example, by default, the command:
`bsub -n 4 -R "rusage[mem=500]" -q reservation myjob`

requires the job to reserve 500 MB on each host where the job runs.

1. To enable per-slot memory reservation, define `RESOURCE_RESERVE_PER_SLOT=y` in `lsb. params`. In this example, if per-slot reservation is enabled, the job must reserve 500 MB of memory for each job slot ($4 * 500 = 2$ GB) on the host in order to run.

Backfill scheduling

By default, a reserved job slot cannot be used by another job. To make better use of resources and improve performance of LSF, you can configure backfill scheduling.

About backfill scheduling

Backfill scheduling allows other jobs to use the reserved job slots, as long as the other jobs do not delay the start of another job. Backfilling, together with processor reservation, allows large parallel jobs to run while not underutilizing resources.

In a busy cluster, processor reservation helps to schedule large parallel jobs sooner. However, by default, reserved processors remain idle until the large job starts. This degrades the performance of LSF because the reserved resources are idle while jobs are waiting in the queue.

Backfill scheduling allows the reserved job slots to be used by small jobs that can run and finish before the large job starts. This improves the performance of LSF because it increases the utilization of resources.

How backfilling works

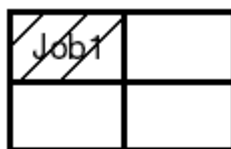
For backfill scheduling, LSF assumes that a job can run until its run limit expires. Backfill scheduling works most efficiently when all the jobs in the cluster have a run limit.

Since jobs with a shorter run limit have more chance of being scheduled as backfill jobs, users who specify appropriate run limits in a backfill queue is rewarded by improved turnaround time.

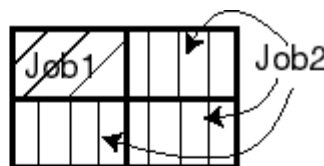
Once the big parallel job has reserved sufficient job slots, LSF calculates the start time of the big job, based on the run limits of the jobs currently running in the reserved slots. LSF cannot backfill if the big job is waiting for a job that has no run limit defined.

If LSF can backfill the idle job slots, only jobs with run limits that expire before the start time of the big job is allowed to use the reserved job slots. LSF cannot backfill with a job that has no run limit.

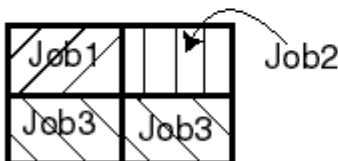
Example



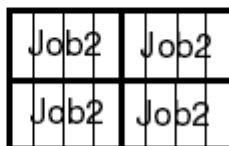
(a) Job1 started at 8:00 am.
Will finish at 10:00 am.



(b) Job2, submitted but can't start
since it needs 4 processors.
Remaining 3 reserved by Job2.



(c) At 8:30 am Job3 submitted.
Job3 backfills Job2.



(d) At 10:00 am, Job2 starts.

In this scenario, assume the cluster consists of a 4-CPU multiprocessor host.

1. A sequential job (job1) with a run limit of 2 hours is submitted and gets started at 8:00 am (figure a).

2. Shortly afterwards, a parallel job (j ob2) requiring all 4 CPUs is submitted. It cannot start right away because j ob1 is using one CPU, so it reserves the remaining 3 processors (figure b).
3. At 8:30 am, another parallel job (j ob3) is submitted requiring only two processors and with a run limit of 1 hour. Since j ob2 cannot start until 10:00am (when j ob1 finishes), its reserved processors can be backfilled by j ob3 (figure c). Therefore j ob3 can complete before j ob2's start time, making use of the idle processors.
4. Job3 finishes at 9:30am and j ob1 at 10:00am, allowing j ob2 to start shortly after 10:00am. In this example, if j ob3's run limit was 2 hours, it would not be able to backfill j ob2's reserved slots, and would have to run after j ob2 finishes.

Limitations

- A job does not have an estimated start time immediately after mbat chd is reconfigured.

Backfilling and job slot limits

A backfill job borrows a job slot that is already taken by another job. The backfill job does not run at the same time as the job that reserved the job slot first. Backfilling can take place even if the job slot limits for a host or processor have been reached. Backfilling cannot take place if the job slot limits for users or queues have been reached.

Job resize allocation requests

Pending job resize allocation requests are supported by backfill policies. However, the run time of pending resize request is equal to the remaining run time of the running resizable job. For example, if RUN LIMIT of a resizable job is 20 hours and 4 hours have already passed, the run time of pending resize request is 16 hours.

Configure backfill scheduling

Backfill scheduling is enabled at the queue level. Only jobs in a backfill queue can backfill reserved job slots. If the backfill queue also allows processor reservation, then backfilling can occur among jobs within the same queue.

Configure a backfill queue

1. To configure a backfill queue, define BACKFILL in l sb. queues.
2. Specify Y to enable backfilling. To disable backfilling, specify N or blank space.

BACKFILL=Y

Enforce run limits

Backfill scheduling requires all jobs to specify a duration. If you specify a run time limit using the command line bsub -W option or by defining the RUNLIMIT parameter in l sb. queues or l sb. appl i cat i ons, LSF uses that value as a hard limit and terminates jobs that exceed the specified duration. Alternatively, you can specify an estimated duration by defining the RUNTIME parameter in l sb. appl i cat i ons. LSF uses the RUNTIME estimate for scheduling purposes only, and does not terminate jobs that exceed the RUNTIME duration.

Backfill scheduling works most efficiently when all the jobs in a cluster have a run limit specified at the job level (bsub -W). You can use the external submission executable, esub, to make sure that all users specify a job-level run limit.

Otherwise, you can specify ceiling and default run limits at the queue level (RUNLIMIT in l sb. queues) or application level (RUNLIMIT in l sb. appl i cat i ons).

View information about job start time

1. Use `bj obs -l` to view the estimated start time of a job.

Use backfill on memory

If BACKFILL is configured in a queue, and a run limit is specified with `-W` on `bsub` or with `RUNLIMIT` in the queue, backfill jobs can use the accumulated memory reserved by the other jobs, as long as the backfill job can finish before the predicted start time of the jobs with the reservation.

Unlike slot reservation, which only applies to parallel jobs, backfill on memory applies to sequential and parallel jobs.

The following queue enables both memory reservation and backfill on memory in the same queue:

```
Begin Queue
QUEUE_NAME = reservation_backfill
DESCRIPTION = For resource reservation and backfill
PRIORITY = 40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
BACKFILL = Y
End Queue
```

Examples of memory reservation and backfill on memory

The following queues are defined in `lsb.queues`:

```
Begin Queue
QUEUE_NAME = reservation
DESCRIPTION = For resource reservation
PRIORITY=40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
End Queue
```

```
Begin Queue
QUEUE_NAME = backfill
DESCRIPTION = For backfill scheduling
PRIORITY = 30
BACKFILL = y
End Queue
```

lsb.params

Per-slot memory reservation is enabled by `RESOURCE_RESERVE_PER_SLOT=y` in `lsb.params`.

Assumptions

Assume one host in the cluster with 10 CPUs and 1 GB of free memory currently available.

Sequential jobs

Each of the following sequential jobs requires 400 MB of memory. The first three jobs run for 300 minutes.

Job 1:

```
bsub -W 300 -R "rusage[mem=400]" -q reservation myjob1
```

The job starts running, using 400M of memory and one job slot.

Job 2:

Submitting a second job with same requirements get the same result.

Job 3:

Submitting a third job with same requirements reserves one job slot, and reserve all free memory, if the amount of free memory is between 20 MB and 200 MB (some free memory may be used by the operating system or other software.)

Job 4:

```
bsub -W 400 -q backfill -R "rusage[mem=50]" myjob4
```

The job keeps pending, since memory is reserved by job 3 and it runs longer than job 1 and job 2.

Job 5:

```
bsub -W 100 -q backfill -R "rusage[mem=50]" myjob5
```

The job starts running. It uses one free slot and memory reserved by job 3. If the job does not finish in 100 minutes, it is killed by LSF automatically.

Job 6:

```
bsub -W 100 -q backfill -R "rusage[mem=300]" myjob6
```

The job keeps pending with no resource reservation because it cannot get enough memory from the memory reserved by job 3.

Job 7:

```
bsub -W 100 -q backfill myjob7
```

The job starts running. LSF assumes it does not require any memory and enough job slots are free.

Parallel jobs

Each process of a parallel job requires 100 MB memory, and each parallel job needs 4 cpus. The first two of the following parallel jobs run for 300 minutes.

Job 1:

```
bsub -W 300 -n 4 -R "rusage[mem=100]" -q reservation myJob1
```

The job starts running and use 4 slots and get 400MB memory.

Job 2:

Submitting a second job with same requirements gets the same result.

Job 3:

Submitting a third job with same requirements reserves 2 slots, and reserves all 200 MB of available memory, assuming no other applications are running outside of LSF.

Job 4:

```
bsub -W 400 -q backfill -R "rusage[mem=50]" myJob4
```

The job keeps pending since all available memory is already reserved by job 3. It runs longer than job 1 and job 2, so no backfill happens.

Job 5:

```
bsub -W 100 -q backfill -R "rusage[mem=50]" myJob5
```

This job starts running. It can backfill the slot and memory reserved by job 3. If the job does not finish in 100 minutes, it is killed by LSF automatically.

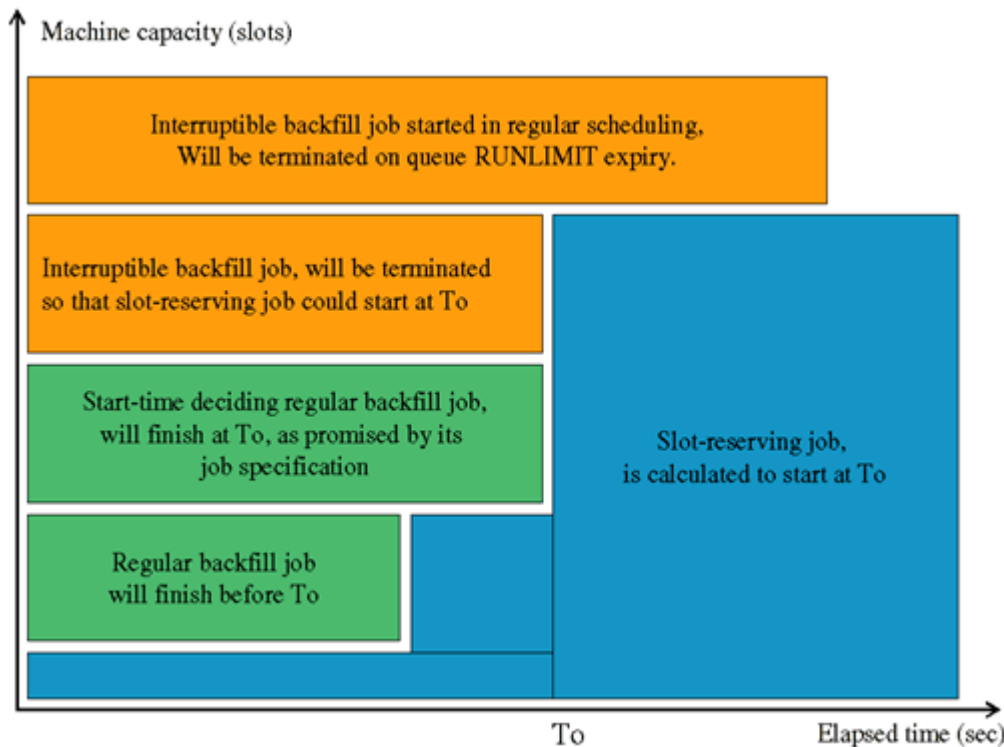
Use interruptible backfill

Interruptible backfill scheduling can improve cluster utilization by allowing reserved job slots to be used by low priority small jobs that are terminated when the higher priority large jobs are about to start.

An interruptible backfill job:

- Starts as a regular job and is killed when it exceeds the queue runtime limit, or
- Is started for backfill whenever there is a backfill time slice longer than the specified minimal time, and killed before the slot-reserving job is about to start. This applies to compute-intensive serial or single-node parallel jobs that can run a long time, yet be able to checkpoint or resume from an arbitrary computation point.

Resource allocation diagram



Job life cycle

1. Jobs are submitted to a queue configured for interruptible backfill. The job runtime requirement is ignored.
2. Job is scheduled as either regular job or backfill job.
3. The queue runtime limit is applied to the regularly scheduled job.
4. In backfill phase, the job is considered for run on any reserved resource, which duration is longer than the minimal time slice configured for the queue. The job runtime limit is set in such way, that the job releases the resource before it is needed by the slot reserving job.
5. The job runs in a regular manner. It is killed upon reaching its runtime limit, and requeued for the next run. Requeueing must be explicitly configured in the queue.

Assumptions and limitations

- The interruptible backfill job holds the slot-reserving job start until its calculated start time, in the same way as a regular backfill job. The interruptible backfill job is killed when its run limit expires.
- Killing other running jobs prematurely does not affect the calculated run limit of an interruptible backfill job. Slot-reserving jobs do not start sooner.

- While the queue is checked for the consistency of interruptible backfill, backfill and runtime specifications, the requeue exit value clause is not verified, nor executed automatically. Configure requeue exit values according to your site policies.
- In Platform MultiCluster, `bhist` does not display interruptible backfill information for remote clusters.
- A migrated job belonging to an interruptible backfill queue is migrated as if `LSB_MIG2PEND` is set.
- Interruptible backfill is disabled for resizable jobs. A resizable job can be submitted into interruptible backfill queue, but the job cannot be resized.

Configure an interruptible backfill queue

1. Configure `INTERRUPTIBLE_BACKFILL=seconds` in the lowest priority queue in the cluster. There can only be one interruptible backfill queue in the cluster.

Specify the minimum number of seconds for the job to be considered for backfilling. This minimal time slice depends on the specific job properties; it must be longer than at least one useful iteration of the job. Multiple queues may be created if a site has jobs of distinctively different classes.

For example:

```
Begin Queue
QUEUE_NAME    = background
# REQUEUE_EXIT_VALUES (set to whatever needed)
DESCRIPTION   = Interruptible Backfill queue
BACKFILL = Y
INTERRUPTIBLE_BACKFILL = 1
RUNLIMIT = 10
PRIORITY = 1
End Queue
```

Interruptible backfill is disabled if `BACKFILL` and `RUNLIMIT` are not configured in the queue.

The value of `INTERRUPTIBLE_BACKFILL` is the minimal time slice in seconds for a job to be considered for backfill. The value depends on the specific job properties; it must be longer than at least one useful iteration of the job. Multiple queues may be created for different classes of jobs.

`BACKFILL` and `RUNLIMIT` must be configured in the queue.

`RUNLIMIT` corresponds to a maximum time slice for backfill, and should be configured so that the wait period for the new jobs submitted to the queue is acceptable to users. 10 minutes of runtime is a common value.

You should configure `REQUEUE_EXIT_VALUES` for the queue so that resubmission is automatic. In order to terminate completely, jobs must have specific exit values:

- If jobs are checkpointable, use their checkpoint exit value.
- If jobs periodically save data on their own, use the `SIGTERM` exit value.

View the run limits for interruptible backfill jobs (bjobs and bhist)

1. Use `bjobs` to display the run limit calculated based on the configured queue-level run limit.

For example, the interruptible backfill queue `lazy` configures `RUNLIMIT=60`:

```
bjobs -l 135
Job <135>, User <user1>, Project <default>, Status <RUN>, Queue <lazy>, Command
<myjob>
Mon Nov 21 11:49:22 2009: Submitted from host <hostA>, CWD <$HOME/H
PC/job>;
  RUNLIMIT
  59.5 min of hostA
Mon Nov 21 11:49:26 2009: Started on <hostA>, Execution Home </home
```

```
/user1>, Execution CWD </home/user1/HPC/j obs>;
```

2. Use `bhi st` to display job-level run limit if specified.

For example, job 135 was submitted with a run limit of 3 hours:

```
bsub -n 1 -q lazy -W 3:0 myjob
```

```
Job <135> is submitted to queue <lazy>.
```

`bhi st` displays the job-level run limit:

bhist -l 135

```
Job <135>, User <user1>, Project <default>, Command <myjob>
Mon Nov 21 11:49:22 2009: Submitted from host <hostA>, to Queue <lazy>, CWD <${HOME}/HPC/j obs>;
  RUNLIMIT
    180.0 min of hostA
Mon Nov 21 11:49:26 2009: Dispatched to <hostA>;
Mon Nov 21 11:49:26 2009: Starting (Pid 2746);
Mon Nov 21 11:49:27 2009: Interruptible backfill runtime limit is 59.5 minutes;
Mon Nov 21 11:49:27 2009: Running with execution home </home/user1>, Execution CWD
...
```

Display available slots for backfill jobs

The `bsl ots` command displays slots reserved for parallel jobs and advance reservations. The available slots are not currently used for running jobs, and can be used for backfill jobs. The available slots displayed by `bsl ots` are only a snapshot of the slots currently not in use by parallel jobs or advance reservations. They are not guaranteed to be available at job submission.

By default, `bsl ots` displays all available slots, and the available run time for those slots. When no reserved slots are available for backfill, `bsl ots` displays "No reserved slots available."

The backfill window calculation is based on the snapshot information (current running jobs, slot reservations, advance reservations) obtained from `mbat chd`. The backfill window displayed can serve as reference for submitting backfillable jobs. However, if you have specified extra resource requirements or special submission options, it does not insure that submitted jobs are scheduled and dispatched successfully.

`bsl ots -R` only supports the `sel ect` resource requirement string. Other resource requirement selections are not supported.

If the available backfill window has no run time limit, its length is displayed as `UNLIMITED`.

Examples

Display all available slots for backfill jobs:

```
bslots
```

```
SLOTS RUNTIME
```

```
1  UNLIMITED
3  1 hour 30 minutes
5  1 hour 0 minutes
7  45 minutes
15 40 minutes
18 30 minutes
20 20 minutes
```

Display available slots for backfill jobs requiring 15 slots or more:

```
bslots -n 15
```

SLOTS RUNTIME

15 40 minutes

18 30 minutes

20 20 minutes

Display available slots for backfill jobs requiring a run time of 30 minutes or more:

bslots -W 30

SLOTS RUNTIME

3 1 hour 30 minutes

5 1 hour 0 minutes

7 45 minutes

15 40 minutes

18 30 minutes

bslots -W 2:45

No reserved slots available.

bslots -n 15 -W 30

SLOTS RUNTIME

15 40 minutes

18 30 minutes

Display available slots for backfill jobs requiring a host with more than 500 MB of memory:

bslots -R "mem>500"

SLOTS RUNTIME

7 45 minutes

15 40 minutes

Display the host names with available slots for backfill jobs:

bslots -l

SLOTS: 15

RUNTIME: 40 minutes

HOSTS: 1*hostB 1*hostE 3*hostC ...

3*hostZ

SLOTS: 15

RUNTIME: 30 minutes

HOSTS: 2*hostA 1*hostB 3*hostC ...

1*hostX

Submit backfill jobs according to available slots

1. Use `bsl ots` to display job slots available for backfill jobs.
2. Submit a job to a backfill queue. Specify a runtime limit and the number of processors required that are within the availability shown by `bsl ots`.

Submitting a job according to the backfill slot availability shown by `bsl ots` does not guarantee that the job is backfilled successfully. The slots may not be available by the time job is actually scheduled, or the job cannot be dispatched because other resource requirements are not satisfied.

Parallel fairshare

LSF can consider the number of CPUs when using fairshare scheduling with parallel jobs.

If the job is submitted with `bsub -n`, the following formula is used to calculate dynamic priority:

$$\text{dynamic priority} = \text{number_shares} / (\text{cpu_time} * \text{CPU_TIME_FACTOR} + \text{run_time} * \text{number_CPUs} * \text{RUN_TIME_FACTOR} + (1 + \text{job_slots}) * \text{RUN_JOB_FACTOR} + \text{fairshare_adjustment}(\text{struc} * \text{shareAdjustPair}) * \text{FAIRSHARE_ADJUSTMENT_FACTOR})$$

where *number_CPU*s is the number of CPUs used by the job.

Configure parallel fairshare

To configure parallel fairshare so that the number of CPUs is considered when calculating dynamic priority for queue-level user-based fairshare:

Note:

LSB_NCPU_ENFORCE does not apply to host-partition user-based fairshare. For host-partition user-based fairshare, the number of CPUs is automatically considered.

1. Configure fairshare at the queue level.
2. Enable parallel fairshare: `LSB_NCPU_ENFORCE=1` in `lsf.conf`.
3. Run the following commands to restart all LSF daemons:

```
# lsadmin reconfig
# lsadmin resrestart all
# badmin hrestart all
# badmin mbdrestart
```

How deadline constraint scheduling works for parallel jobs

Deadline constraint scheduling is enabled by default.

If deadline constraint scheduling is enabled and a parallel job has a CPU limit but no run limit, LSF considers the number of processors when calculating how long the job takes.

LSF assumes that the minimum number of processors are used, and that they are all the same speed as the candidate host. If the job cannot finish under these conditions, LSF does not place the job.

The formula is:

$(\text{deadline time} - \text{current time}) > (\text{CPU limit on candidate host} / \text{minimum number of processors})$

Optimized preemption of parallel jobs

You can configure preemption for parallel jobs to reduce the number of jobs suspended in order to run a large parallel job.

When a high-priority parallel job preempts multiple low-priority parallel jobs, sometimes LSF preempts more low-priority jobs than are necessary to release sufficient job slots to start the high-priority job.

The `PREEMPT_FOR` parameter in `l sb. params` with the `MINI_JOB` keyword enables the optimized preemption of parallel jobs, so LSF preempts fewer of the low-priority parallel jobs.

Enabling the feature only improves the efficiency in cases where both preemptive and preempted jobs are parallel jobs.

How optimized preemption works

When you run many parallel jobs in your cluster, and parallel jobs preempt other parallel jobs, you can enable a feature to optimize the preemption mechanism among parallel jobs.

By default, LSF can over-preempt parallel jobs. When a high-priority parallel job preempts multiple low-priority parallel jobs, sometimes LSF preempts more low-priority jobs than are necessary to release sufficient job slots to start the high-priority job. The optimized preemption mechanism reduces the number of jobs that are preempted.

Enabling the feature only improves the efficiency in cases where both preemptive and preempted jobs are parallel jobs. Enabling or disabling this feature has no effect on the scheduling of jobs that require only a single processor.

Configure optimized preemption

1. Use the `PREEMPT_FOR` parameter in `l sb. params` and specify the keyword `MINI_JOB` to configure optimized preemption at the cluster level.

If the parameter is already set, the `MINI_JOB` keyword can be used along with other keywords; the other keywords do not enable or disable the optimized preemption mechanism.

Processor binding for parallel jobs

By default, there is no processor binding.

For multi-host parallel jobs, LSF sets two environment variables (`$LSB_BIND_JOB` and `$LSB_BIND_CPU_LIST`) but does not attempt to bind the job to any host even if you enable the processor binding.

Resizable jobs

Adding slots to or removing slots from a resizable job triggers unbinding and rebinding of job processes. Rebinding does not guarantee that the processes can be bound to the same processors they were bound to previously.

If a multi-host parallel job becomes a single-host parallel job after resizing, LSF does not bind it.

If a single-host parallel job or sequential job becomes a multi-host parallel job after resizing, LSF does not bind it.

After unbinding and binding, the job CPU affinity is changed. LSF puts the new CPU list in the `LSB_BIND_CPU_LIST` environment variable and the binding method to `LSB_BIND_JOB` environment variable. And it is the responsibility of the notification command to tell the job that CPU binding has changed.

Job Execution and Interactive Jobs

Runtime Resource Usage Limits

About resource usage limits

Resource usage limits control how much resource can be consumed by running jobs. Jobs that use more than the specified amount of a resource are signalled or have their priority lowered.

Limits can be specified by the LSF administrator:

- At the queue level in `lsb.queues`
- In an application profile in `lsb.applications`
- At the job level when you submit a job

For example, by defining a high-priority short queue, you can allow short jobs to be scheduled earlier than long jobs. To prevent some users from submitting long jobs to this short queue, you can set CPU limit for the queue so that no jobs submitted from the queue can run for longer than that limit.

Limits specified at the queue level are *hard* limits, while those specified with job submission or in an application profile are *soft* limits. The hard limit acts as a ceiling for the soft limit. See `setrlimit(2)` man page for concepts of hard and soft limits.

Note:

This chapter describes queue-level and job-level resource usage limits. Priority of limits is different if limits are also configured in an application profile.

Resource usage limits and resource allocation limits

Resource usage limits are not the same as *resource allocation limits*, which are enforced during job scheduling and before jobs are dispatched. You set resource allocation limits to restrict the amount of a given resource that must be available during job scheduling for different classes of jobs to start, and which resource consumers the limits apply to. .

Resource usage limits and resource reservation limits

Resource usage limits are not the same as queue-based *resource reservation limits*, which are enforced during job submission. The parameter `RESRSV_LIMIT` (in `lsb.queues`) specifies allowed ranges of resource values, and jobs submitted with resource requests outside of this range are rejected.

Summary of resource usage limits

Limit	Job syntax (bsub)	Syntax (lsb.queues and lsb.applications)	Format/Default Units
Core file size limit	<code>-C core_limit</code>	<code>CORELIMIT=limit</code>	<i>integer</i> KB
CPU time limit	<code>-c cpu_limit</code>	<code>CPULIMIT=[default] maximum</code>	<code>[hours:]minutes[/host_name /host_model]</code>
Data segment size limit	<code>-D data_limit</code>	<code>DATALIMIT=[default] maximum</code>	<i>integer</i> KB
File size limit	<code>-F file_limit</code>	<code>FILELIMIT=limit</code>	<i>integer</i> KB
Memory limit	<code>-M mem_limit</code>	<code>MEMLIMIT=[default] maximum</code>	<i>integer</i> KB
Process limit	<code>-p process_limit</code>	<code>PROCESSLIMIT=[default] maximum</code>	<i>integer</i>

Limit	Job syntax (bsub)	Syntax (lsb.queues and lsb.applications)	Format/Default Units
Run time limit	-W <i>run_limit</i>	RUNLIMIT=[default] maximum	[hours:]minutes[/host_name /host_model]
Stack segment size limit	-S <i>stack_limit</i>	STACKLIMIT= <i>limit</i>	integer KB
Virtual memory limit	-v <i>swap_limit</i>	SWAPLIMIT= <i>limit</i>	integer KB
Thread limit	-T <i>thread_limit</i>	THREADLIMIT=[default] maximum	integer

Priority of resource usage limits

If no limit is specified at job submission, then the following apply to all jobs submitted to the queue:

If ...	Then ...
Both default and maximum limits are defined	The default is enforced
Only a maximum is defined	The maximum is enforced
No limit is specified in the queue or at job submission	No limits are enforced

Incorrect resource usage limits

Incorrect limits are ignored, and a warning message is displayed when the cluster is reconfigured or restarted. A warning message is also logged to the `mbatchd` log file when LSF is started.

If no limit is specified at job submission, then the following apply to all jobs submitted to the queue:

If ...	Then ...
The default limit is not correct	The default is ignored and the maximum limit is enforced
Both default and maximum limits are specified, and the maximum is not correct	The maximum is ignored and the resource has no maximum limit, only a default limit
Both default and maximum limits are not correct	The default and maximum are ignored and no limit is enforced

Resource usage limits specified at job submission must be less than the maximum specified in `lsb.queues`. The job submission is rejected if the user-specified limit is greater than the queue-level maximum, and the following message is issued:

```
Cannot exceed queue's hard limit(s). Job not submitted.
```

Enforce limits on chunk jobs

By default, resource usage limits are not enforced for chunk jobs because chunk jobs are typically too short to allow LSF to collect resource usage.

1. To enforce resource limits for chunk jobs, define `LSB_CHUNK_RUSAGE=Y` in `lsf.conf`. Limits may not be enforced for chunk jobs that take less than a minute to run.

Scaling the units for resource usage limits

The default unit for the following resource usage limits is KB:

- Core limit (-C and CORELIMIT)
- Memory limit (-M and MEMLIMIT)
- Stack limit (-S and STACKLIMIT)
- Swap limit (-v and SWAPLIMIT)

This default may be too small for some environments that make use of very large resource usage limits, for example, GB or TB.

LSF_UNIT_FOR_LIMITS in `lsf.conf` specifies larger units for the resource usage limits with default unit of KB.

The unit for the resource usage limit can be one of:

- KB (kilobytes)
- MB (megabytes)
- GB (gigabytes)
- TB (terabytes)
- PB (petabytes)
- EB (exabytes)

LSF_UNIT_FOR_LIMITS applies cluster-wide to limits at the job-level (bsub), queue-level (lsb. queues), and application level (lsb. applications).

The limit unit specified by LSF_UNIT_FOR_LIMITS also applies to limits modified with `bmod`, and the display of resource usage limits in query commands (`bacct`, `bapp`, `bhist`, `bhosts`, `bjobs`, `bqueues`, `lsload`, and `lshosts`).

Important:

Before changing the units of your resource usage limits, you should completely drain the cluster of all workload. There should be no running, pending, or finished jobs in the system.

In a MultiCluster environment, you should configure the same unit for all clusters.

After changing LSF_UNIT_FOR_LIMITS, you must restart your cluster.

How limit unit changes affect jobs

When LSF_UNIT_FOR_LIMITS is specified, the defined unit is used for the following commands. In command output, the larger unit appears as T, G, P, or E, depending on the job usage and the unit defined.

Command	Option/Output	Default unit
bsub/bmod	-C (core limit)	KB
	-M (memory limit)	KB
	-S (stack limit)	KB
	-v (swap limit)	KB

Command	Option/Output	Default unit
bjobs	rusage CORELIMIT, MEMLIMIT, STACKLIMIT, SWAPLIMIT	KB (may show MB depending on job rusage)
bqueues	CORELIMIT, MEMLIMIT, STACKLIMIT, SWAPLIMIT	KB (may show MB depending on job rusage)
	loadSched, loadStop	MB
bacct	Summary rusage	KB (may show MB depending on job rusage)
bapp	CORELIMIT, MEMLIMIT, STACKLIMIT, SWAPLIMIT	KB
bhist	History of limit change by bmod	KB
	MEM, SWAP	KB (may show MB depending on job rusage)
bhosts	loadSched, loadStop	MB
lsload	mem, swp	KB (may show MB depending on job rusage)
lshosts	maxmem, maxswp	KB (may show MB depending on job rusage)

Example

A job is submitted with `bsub -M 100` and `LSF_UNIT_FOR_LIMITS=MB`; the memory limit for the job is 100 MB rather than the default 100 KB.

Specify resource usage limits

Queues can enforce resource usage limits on running jobs. LSF supports most of the limits that the underlying operating system supports. In addition, LSF also supports a few limits that the underlying operating system does not support.

1. Specify queue-level resource usage limits using parameters in `l sb. queues`.

Specify queue-level resource usage limits

Limits configured in `l sb. queues` apply to all jobs submitted to the queue. Job-level resource usage limits specified at job submission override the queue definitions.

1. Specify only a maximum value for the resource.

For example, to specify a maximum run limit, use one value for the `RUNLIMIT` parameter in `l sb. queues`:

```
RUNLIMIT = 10
```

The maximum run limit for the queue is 10 minutes. Jobs cannot run for more than 10 minutes. Jobs in the `RUN` state for longer than 10 minutes are killed by LSF.

If only one run limit is specified, jobs that are submitted with `bsub -W` with a run limit that exceeds the maximum run limit is not allowed to run. Jobs submitted without `bsub -W` are allowed to run but are killed when they are in the `RUN` state for longer than the specified maximum run limit.

For example, in `l sb. queues`:

```
RUNLIMIT = 10
```

Default and maximum values

If you specify two limits, the first one is the default limit for jobs in the queue and the second one is the maximum (hard) limit. Both the default and the maximum limits must be positive integers. The default limit must be less than the maximum limit. The default limit is ignored if it is greater than the maximum limit.

Use the default limit to avoid having to specify resource usage limits in the `bsub` command.

For example, to specify a default and a maximum run limit, use two values for the `RUNLIMIT` parameter in `l sb. queues`:

```
RUNLIMIT = 10 15
```

- The first number is the default run limit applied to all jobs in the queue that are submitted without a job-specific run limit (without `bsub -W`).
- The second number is the maximum run limit applied to all jobs in the queue that are submitted with a job-specific run limit (with `bsub -W`). The default run limit must be less than the maximum run limit.

You can specify both default and maximum values for the following resource usage limits in `l sb. queues`:

- `CPULIMIT`
- `DATALIMIT`
- `MEMLIMIT`
- `PROCESSLIMIT`
- `RUNLIMIT`

- THREADLIMIT

Host specification with two limits

If default and maximum limits are specified for CPU time limits or run time limits, only one host specification is permitted. For example, the following CPU limits are correct (and have an identical effect):

```
CPULIMIT = 400/hostA 600
```

```
CPULIMIT = 400 600/hostA
```

The following CPU limit is not correct:

```
CPULIMIT = 400/hostA 600/hostB
```

The following run limits are correct (and have an identical effect):

```
RUNLIMIT = 10/hostA 15
```

```
RUNLIMIT = 10 15/hostA
```

The following run limit is not correct:

```
RUNLIMIT = 10/hostA 15/hostB
```

Default run limits for backfill scheduling

Default run limits are used for backfill scheduling of parallel jobs.

For example, in lsb. queues, you enter: `RUNLIMIT = 10 15`

- The first number is the default run limit applied to all jobs in the queue that are submitted without a job-specific run limit (without `bsub -W`).
- The second number is the maximum run limit applied to all jobs in the queue that are submitted with a job-specific run limit (with `bsub -W`). The default run limit cannot exceed the maximum run limit.

Automatically assigning a default run limit to all jobs in the queue means that backfill scheduling works efficiently.

For example, in lsb. queues, you enter:

```
RUNLIMIT = 10 15
```

The first number is the default run limit applied to all jobs in the queue that are submitted without a job-specific run limit. The second number is the maximum run limit.

If you submit a job to the queue without the `-W` option, the default run limit is used:

```
bsub myjob
```

The job `myjob` cannot run for more than 10 minutes as specified with the default run limit.

If you submit a job to the queue with the `-W` option, the maximum run limit is used:

```
bsub -W 12 myjob
```

The job `myjob` is allowed to run on the queue because the specified run limit (12) is less than the maximum run limit for the queue (15).

```
bsub -W 20 myjob
```

The job `myjob` is rejected from the queue because the specified run limit (20) is more than the maximum run limit for the queue (15).

Specify job-level resource usage limits

1. To specify resource usage limits at the job level, use one of the following `bsub` options:

- *-C core_limit*
- *-c cpu_limit*
- *-D data_limit*
- *-F file_limit*
- *-M mem_limit*
- *-p process_limit*
- *-W run_limit*
- *-S stack_limit*
- *-T thread_limit*
- *-v swap_limit*

Job-level resource usage limits specified at job submission override the queue definitions.

Supported resource usage limits and syntax

Core file size limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-C <i>core_limit</i>	CORELIMIT= <i>limit</i>	<i>integer</i> KB

Sets a per-process (soft) core file size limit for each process that belongs to this batch job.

By default, the limit is specified in KB. Use LSF_UNIT_FOR_LIMITS in `lsf.conf` to specify a larger unit for the limit (MB, GB, TB, PB, or EB).

On some systems, no core file is produced if the image for the process is larger than the core limit. On other systems only the first *core_limit* KB of the image are dumped. The default is no soft limit.

CPU time limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-c <i>cpu_limit</i>	CPULIMIT=[<i>default</i>] <i>maximum</i>	[<i>hours:</i>] <i>minutes</i> [/ <i>host_name</i> / <i>host_model</i>]

Sets the soft CPU time limit to *cpu_limit* for this batch job. The default is no limit. This option is useful for avoiding runaway jobs that use up too many resources. LSF keeps track of the CPU time used by all processes of the job.

When the job accumulates the specified amount of CPU time, a SIGXCPU signal is sent to all processes belonging to the job. If the job has no signal handler for SIGXCPU, the job is killed immediately. If the SIGXCPU signal is handled, blocked, or ignored by the application, then after the grace period expires, LSF sends SIGINT, SIGTERM, and SIGKILL to the job to kill it.

You can define whether the CPU limit is a per-process limit enforced by the OS or a per-job limit enforced by LSF with LSB_JOB_CPULIMIT in `lsf.conf`.

Jobs submitted to a chunk job queue are not chunked if the CPU limit is greater than 30 minutes.

Format

cpu_limit is in the form [*hour:*]*minute*, where *minute* can be greater than 59. 3.5 hours can either be specified as 3:30 or 210.

Normalized CPU time

The CPU time limit is *normalized* according to the CPU factor of the submission host and execution host. The CPU limit is scaled so that the job does approximately the same amount of processing for a given CPU limit, even if it is sent to a host with a faster or slower CPU.

For example, if a job is submitted from a host with a CPU factor of 2 and executed on a host with a CPU factor of 3, the CPU time limit is multiplied by 2/3 because the execution host can do the same amount of work as the submission host in 2/3 of the time.

If the optional host name or host model is not given, the CPU limit is scaled based on the DEFAULT_HOST_SPEC specified in the `lsb.params` file. (If DEFAULT_HOST_SPEC is not defined, the fastest batch host in the cluster is used as the default.) If host or host model is given, its CPU scaling factor is used to adjust the actual CPU time limit at the execution host.

The following example specifies that myj ob can run for 10 minutes on a DEC3000 host, or the corresponding time on any other host:

```
bsub -c 10/DEC3000 myjob
```

Data segment size limit

Job syntax (bsub)	Queue syntax (lsb.queueues)	Format/Default Units
-D <i>data_limit</i>	DATALIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i> KB

Sets a per-process (soft) data segment size limit in KB for each process that belongs to this batch job (see `getrlimit(2)`).

This option affects calls to `sbrk()` and `brk()`. An `sbrk()` or `malloc()` call to extend the data segment beyond the data limit returns an error.

Note:

Linux does not use `sbrk()` and `brk()` within its `calloc()` and `malloc()`. Instead, it uses `mmap()` to create memory. DATALIMIT cannot be enforced on Linux applications that call `sbrk()` and `malloc()`.

On AIX, if the XPG_SUS_ENV=ON environment variable is set in the user's environment before the process is executed and a process attempts to set the limit lower than current usage, the operation fails with `errno` set to `EINVAL`. If the XPG_SUS_ENV environment variable is not set, the operation fails with `errno` set to `EFAULT`.

The default is no soft limit.

File size limit

Job syntax (bsub)	Queue syntax (lsb.queueues)	Format/Default Units
-F <i>file_limit</i>	FILELIMIT= <i>limit</i>	<i>integer</i> KB

Sets a per-process (soft) file size limit in KB for each process that belongs to this batch job. If a process of this job attempts to write to a file such that the file size would increase beyond the file limit, the kernel sends that process a `SIGXFSZ` signal. This condition normally terminates the process, but may be caught. The default is no soft limit.

Memory limit

Job syntax (bsub)	Queue syntax (lsb.queueues)	Format/Default Units
-M <i>mem_limit</i>	MEMLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i> KB

Sets a per-process physical memory limit for all of the processes belonging to a job

By default, the limit is specified in KB. Use `LSF_UNIT_FOR_LIMITS` in `lsf.conf` to specify a larger unit for the the limit (MB, GB, TB, PB, or EB).

If `LSB_MEMLIMIT_ENFORCE=Y` or `LSB_JOB_MEMLIMIT=Y` are set in `lsf.conf`, LSF kills the job when it exceeds the memory limit. Otherwise, LSF passes the memory limit to the operating system. Some operating systems apply the memory limit to each process, and some do not enforce the memory limit at all.

Platform LSF memory limit enforcement

To enable LSF memory limit enforcement, set `LSB_MEMLIMIT_ENFORCE` in `lsf.conf` to `y`. LSF memory limit enforcement explicitly sends a signal to kill a running process once it has allocated memory past *mem_limit*.

You can also enable LSF memory limit enforcement by setting `LSB_JOB_MEMLIMIT` in `lsf.conf` to `y`. The difference between `LSB_JOB_MEMLIMIT` set to `y` and `LSB_MEMLIMIT_ENFORCE` set to `y` is that with `LSB_JOB_MEMLIMIT`, only the per-job memory limit enforced by LSF is enabled. The per-process memory limit enforced by the OS is disabled. With `LSB_MEMLIMIT_ENFORCE` set to `y`, both the per-job memory limit enforced by LSF and the per-process memory limit enforced by the OS are enabled.

`LSB_JOB_MEMLIMIT` disables per-process memory limit enforced by the OS and enables per-job memory limit enforced by LSF. When the total memory allocated to all processes in the job exceeds the memory limit, LSF sends the following signals to kill the job: `SIGINT` first, then `SIGTERM`, then `SIGKILL`.

On UNIX, the time interval between `SIGINT`, `SIGKILL`, `SIGTERM` can be configured with the parameter `JOB_TERMINATE_INTERVAL` in `lsb.params`.

OS memory limit enforcement

OS enforcement usually allows the process to eventually run to completion. LSF passes *mem_limit* to the OS, which uses it as a guide for the system scheduler and memory allocator. The system may allocate more memory to a process if there is a surplus. When memory is low, the system takes memory from and lowers the scheduling priority (re-nice) of a process that has exceeded its declared *mem_limit*.

OS memory limit enforcement is only available on systems that support `RLIMIT_RSS` for `setrlimit()`.

The following operating systems do not support the memory limit at the OS level:

- Microsoft Windows
- Sun Solaris 2.x

Process limit

Job syntax (bsub)	Queue syntax (lsb.queue)	Format/Default Units
<code>-p process_limit</code>	<code>PROCESSLIMIT=[default] maximum</code>	<i>integer</i>

Sets the limit of the number of processes to *process_limit* for the whole job. The default is no limit. Exceeding the limit causes the job to terminate.

Limits the number of concurrent processes that can be part of a job.

If a default process limit is specified, jobs submitted to the queue without a job-level process limit are killed when the default process limit is reached.

If you specify only one limit, it is the maximum, or hard, process limit. If you specify two limits, the first one is the default, or soft, process limit, and the second one is the maximum process limit.

Run time limit

Job syntax (bsub)	Queue syntax (lsb.queueues)	Format/Default Units
-W <i>run_limit</i>	RUNLIMIT=[default] maximum	[hours:]minutes[/host_name /host_model]

A run time limit is the maximum amount of time a job can run before it is terminated. It sets the run time limit of a job. The default is no limit. If the accumulated time the job has spent in the RUN state exceeds this limit, the job is sent a USR2 signal. If the job does not terminate within 10 minutes after being sent this signal, it is killed.

With deadline constraint scheduling configured, a run limit also specifies the amount of time a job is expected to take, and the minimum amount of time that must be available before a job can be started.

Jobs submitted to a chunk job queue are not chunked if the run limit is greater than 30 minutes.

Format

run_limit is in the form [hour:]*minute*, where *minute* can be greater than 59. 3.5 hours can either be specified as 3:30 or 210.

Normalized run time

The run time limit is *normalized* according to the CPU factor of the submission host and execution host. The run limit is scaled so that the job has approximately the same run time for a given run limit, even if it is sent to a host with a faster or slower CPU.

For example, if a job is submitted from a host with a CPU factor of 2 and executed on a host with a CPU factor of 3, the run limit is multiplied by 2/3 because the execution host can do the same amount of work as the submission host in 2/3 of the time.

If the optional host name or host model is not given, the run limit is scaled based on the DEFAULT_HOST_SPEC specified in the lsb.params file. (If DEFAULT_HOST_SPEC is not defined, the fastest batch host in the cluster is used as the default.) If host or host model is given, its CPU scaling factor is used to adjust the actual run limit at the execution host.

The following example specifies that myjob can run for 10 minutes on a DEC3000 host, or the corresponding time on any other host:

```
bsub -W 10/DEC3000 myjob
```

If ABS_RUNLIMIT=Y is defined in lsb.params, the run time limit is *not* normalized by the host CPU factor. Absolute wall-clock run time is used for all jobs submitted with a run limit.

Platform MultiCluster

For MultiCluster jobs, if no other CPU time normalization host is defined and information about the submission host is not available, LSF uses the host with the largest CPU factor (the fastest host in the cluster). The ABS_RUNLIMIT parameter in lsb.params is not supported in either MultiCluster model; run time limit is normalized by the CPU factor of the execution host.

Thread limit

Job syntax (bsub)	Queue syntax (lsb.queueues)	Format/Default Units
-T <i>thread_limit</i>	THREADLIMIT=[default] maximum	integer

Sets the limit of the number of concurrent threads to *thread_limit* for the whole job. The default is no limit.

Exceeding the limit causes the job to terminate. The system sends the following signals in sequence to all processes belongs to the job: SIGINT, SIGTERM, and SIGKILL.

If a default thread limit is specified, jobs submitted to the queue without a job-level thread limit are killed when the default thread limit is reached.

If you specify only one limit, it is the maximum, or hard, thread limit. If you specify two limits, the first one is the default, or soft, thread limit, and the second one is the maximum thread limit.

Stack segment size limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-S <i>stack_limit</i>	STACKLIMIT= <i>limit</i>	<i>integer</i> KB

Sets a per-process (hard) stack segment size limit for all of the processes belonging to a job. Application-level and job-level stack segment size limits overwrite this value as the soft limit, but cannot exceed the hard limit set in `lsb.queues`.

By default, the limit is specified in KB. Use `LSF_UNIT_FOR_LIMITS` in `lsf.conf` to specify a larger unit for the the limit (MB, GB, TB, PB, or EB).

An `sbrk()` call to extend the stack segment beyond the stack limit causes the process to be terminated. The default is no limit.

Virtual memory (swap) limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-v <i>swap_limit</i>	SWAPLIMIT= <i>limit</i>	<i>integer</i> KB

Sets a total process virtual memory limit for the whole job. The default is no limit. Exceeding the limit causes the job to terminate.

By default, the limit is specified in KB. Use `LSF_UNIT_FOR_LIMITS` in `lsf.conf` to specify a larger unit for the the limit (MB, GB, TB, PB, or EB).

This limit applies to the whole job, no matter how many processes the job may contain.

Examples

Queue-level limits

CPULIMIT = 20/hostA 15

The first number is the default CPU limit. The second number is the maximum CPU limit.

However, the default CPU limit is ignored because it is a higher value than the maximum CPU limit.

CPULIMIT = 10/hostA

In this example, the lack of a second number specifies that there is no default CPU limit. The specified number is considered as the default and maximum CPU limit.

RUNLIMIT = 10/hostA 15

The first number is the default run limit. The second number is the maximum run limit.

The first number specifies that the default run limit is to be used for jobs that are submitted without a specified run limit (without the -W option of bsub).

RUNLIMIT = 10/hostA

No default run limit is specified. The specified number is considered as the default and maximum run limit.

THREADLIMIT=6

No default thread limit is specified. The value 6 is the default and maximum thread limit.

THREADLIMIT=6 8

The first value (6) is the default thread limit. The second value (8) is the maximum thread limit.

Job-level limits

bsub -M 5000 myjob

Submits myj ob with a memory limit of 5000 KB.

bsub -W 14 myjob

myj ob is expected to run for 14 minutes. If the run limit specified with bsub -W exceeds the value for the queue, the job is rejected.

bsub -T 4 myjob

Submits myj ob with a maximum number of concurrent threads of 4.

CPU time and run time normalization

To set the CPU time limit and run time limit for jobs in a platform-independent way, LSF scales the limits by the CPU factor of the hosts involved. When a job is dispatched to a host for execution, the limits are then normalized according to the CPU factor of the execution host.

Whenever a normalized CPU time or run time is given, the actual time on the execution host is the specified time multiplied by the CPU factor of the normalization host then divided by the CPU factor of the execution host.

If `ABS_RUNLIMIT=Y` is defined in `l sb. params` or in `l sb. appl i cat i ons` for the application associated with your job, the run time limit and run time estimate are not normalized by the host CPU factor. Absolute wall-clock run time is used for all jobs submitted with a run time limit or a run time estimate.

Normalization host

If no host or host model is given with the CPU time or run time, LSF uses the default CPU time normalization host defined at the queue level (`DEFAULT_HOST_SPEC` in `l sb. queues`) if it has been configured, otherwise uses the default CPU time normalization host defined at the cluster level (`DEFAULT_HOST_SPEC` in `l sb. params`) if it has been configured, otherwise uses the submission host.

Example

CPULIMIT=10/hostA

If hostA has a CPU factor of 2, and hostB has a CPU factor of 1 (hostB is slower than hostA), this specifies an actual time limit of 10 minutes on hostA, or on any other host that has a CPU factor of 2. However, if hostB is the execution host, the actual time limit on hostB is 20 minutes ($10 * 2 / 1$).

Normalization hosts for default CPU and run time limits

The first valid CPU factor encountered is used for both CPU limit and run time limit. To be valid, a host specification must be a valid host name that is a member of the LSF cluster. The CPU factor is used even if the specified limit is not valid.

If the CPU and run limit have different host specifications, the CPU limit host specification is enforced.

If no host or host model is given with the CPU or run time limits, LSF determines the default normalization host according to the following priority:

1. `DEFAULT_HOST_SPEC` is configured in `l sb. queues`
2. `DEFAULT_HOST_SPEC` is configured in `l sb. params`
3. If `DEFAULT_HOST_SPEC` is not configured in `l sb. queues` or `l sb. params`, host with the largest CPU factor is used.

CPU time display (bacct, bhist, bqueues)

Normalized CPU time is displayed in the output of `bqueues`. CPU time is *not* normalized in the output of `bacct` and `bhist`.

PAM resource limits

PAM limits are system resource limits defined in `limits.conf`.

- Windows: Not applicable
- Linux: `/etc/pam.d/l sf`

When `USE_PAM_CREDS` is enabled in `lsb. queues` or `lsb. appl i cat i ons`, applies PAM limits to an application or queue when its job is dispatched to a Linux host using PAM. The job will fail if the execution host does not have PAM configured.

Configure a PAM file

When `USE_PAM_CREDS` is enabled in `lsb. queues` or `lsb. appl i cat i ons`, the limits specified in the PAM configuration file are applied to an application or queue when its job is dispatched to a Linux host using PAM. The job will fail if the execution host does not have PAM configured.

1. Create a PAM configuration file on each execution host you want.

`/etc/pam.d/l sf`

2. In the first two lines, specify the authentication and authorization you need to successfully run PAM limits. For example:

`auth required pam_localuser.so`

`account required pam_unix.so`

3. Specify any resource limits. For example:

`session required pam_limits.so`

On hosts that have a PAM configuration file with resource limits specified and when `USE_PAM_CREDS=y` in `lsb. queues` or `lsb. appl i cat i ons`, applies resource limits on jobs running on the execution host.

For more information about configuring a PAM file, check Linux documentation.

Load Thresholds

Automatic job suspension

Jobs running under LSF can be suspended based on the load conditions on the execution hosts. Each host and each queue can be configured with a set of suspending conditions. If the load conditions on an execution host exceed either the corresponding host or queue suspending conditions, one or more jobs running on that host are suspended to reduce the load.

When LSF suspends a job, it invokes the SUSPEND action. The default SUSPEND action is to send the signal SIGSTOP.

By default, jobs are resumed when load levels fall below the suspending conditions. Each host and queue can be configured so that suspended checkpointable or rerunnable jobs are automatically migrated to another host instead.

If no suspending threshold is configured for a load index, LSF does not check the value of that load index when deciding whether to suspend jobs.

Suspending thresholds can also be used to enforce inter-queue priorities. For example, if you configure a low-priority queue with an `r1m` (1 minute CPU run queue length) scheduling threshold of 0.25 and an `r1m` suspending threshold of 1.75, this queue starts one job when the machine is idle. If the job is CPU intensive, it increases the run queue length from 0.25 to roughly 1.25. A high-priority queue configured with a scheduling threshold of 1.5 and an unlimited suspending threshold sends a second job to the same host, increasing the run queue to 2.25. This exceeds the suspending threshold for the low priority job, so it is stopped. The run queue length stays above 0.25 until the high priority job exits. After the high priority job exits the run queue index drops back to the idle level, so the low priority job is resumed.

When jobs are running on a host, LSF periodically checks the load levels on that host. If any load index exceeds the corresponding per-host or per-queue suspending threshold for a job, LSF suspends the job. The job remains suspended until the load levels satisfy the scheduling thresholds.

At regular intervals, LSF gets the load levels for that host. The period is defined by the `SBD_SLEEP_TIME` parameter in the `lsb.params` file. Then, for each job running on the host, LSF compares the load levels against the host suspending conditions and the queue suspending conditions. If any suspending condition at either the corresponding host or queue level is satisfied as a result of increased load, the job is suspended. A job is only suspended if the load levels are too high for that particular job's suspending thresholds.

There is a time delay between when LSF suspends a job and when the changes to host load are seen by the LIM. To allow time for load changes to take effect, LSF suspends no more than one job at a time on each host.

Jobs from the lowest priority queue are checked first. If two jobs are running on a host and the host is too busy, the lower priority job is suspended and the higher priority job is allowed to continue. If the load levels are still too high on the next turn, the higher priority job is also suspended.

If a job is suspended because of its own load, the load drops as soon as the job is suspended. When the load goes back within the thresholds, the job is resumed until it causes itself to be suspended again.

Exceptions

In some special cases, LSF does not automatically suspend jobs because of load levels. LSF does not suspend a job:

- Forced to run with `brun -f`.
- If it is the only job running on a host, unless the host is being used interactively. When only one job is running on a host, it is not suspended for any reason except that the host is not interactively idle (the `it` interactive idle time load index is less than one minute). This means that once a job is started on a host, at least one job continues to run unless there is an interactive user on the host. Once the job

is suspended, it is not resumed until all the scheduling conditions are met, so it should not interfere with the interactive user.

- Because of the paging rate, unless the host is being used interactively. When a host has interactive users, LSF suspends jobs with high paging rates, to improve the response time on the host for interactive users. When a host is idle, the pg (paging rate) load index is ignored. The PG_SUSP_IT parameter in `lsb. params` controls this behavior. If the host has been idle for more than PG_SUSP_IT minutes, the pg load index is not checked against the suspending threshold.

Suspending conditions

LSF provides different alternatives for configuring suspending conditions. Suspending conditions are configured at the host level as load thresholds, whereas suspending conditions are configured at the queue level as either load thresholds, or by using the `STOP_COND` parameter in the `l sb. queues` file, or both.

The load indices most commonly used for suspending conditions are the CPU run queue lengths (`r15s`, `r1m`, and `r15m`), paging rate (`pg`), and idle time (`it`). The (`swp`) and (`tmp`) indices are also considered for suspending jobs.

To give priority to interactive users, set the suspending threshold on the `it` (idle time) load index to a non-zero value. Jobs are stopped when any user is active, and resumed when the host has been idle for the time given in the `it` scheduling condition.

To tune the suspending threshold for paging rate, it is desirable to know the behavior of your application. On an otherwise idle machine, check the paging rate using `lsl load`, and then start your application. Watch the paging rate as the application runs. By subtracting the active paging rate from the idle paging rate, you get a number for the paging rate of your application. The suspending threshold should allow at least 1.5 times that amount. A job can be scheduled at any paging rate up to the scheduling threshold, so the suspending threshold should be at least the scheduling threshold plus 1.5 times the application paging rate. This prevents the system from scheduling a job and then immediately suspending it because of its own paging.

The effective CPU run queue length condition should be configured like the paging rate. For CPU-intensive sequential jobs, the effective run queue length indices increase by approximately one for each job. For jobs that use more than one process, you should make some test runs to determine your job's effect on the run queue length indices. Again, the suspending threshold should be equal to at least the scheduling threshold plus 1.5 times the load for one job.

Resizable jobs

If new hosts are added for resizable jobs, LSF considers load threshold scheduling on those new hosts. If hosts are removed from allocation, LSF does not apply load threshold scheduling for resizing the jobs.

Configuring load thresholds at queue level

The queue definition (`l sb. queues`) can contain thresholds for 0 or more of the load indices. Any load index that does not have a configured threshold has no effect on job scheduling.

Syntax

Each load index is configured on a separate line with the format:

```
load_index = loadSched/loadStop
```

Specify the name of the load index, for example `r1m` for the 1-minute CPU run queue length or `pg` for the paging rate. `loadSched` is the scheduling threshold for this load index. `loadStop` is the suspending threshold. The `loadSched` condition must be satisfied by a host before a job is dispatched to it and also before a job suspended on a host can be resumed. If the `loadStop` condition is satisfied, a job is suspended.

The `loadSched` and `loadStop` thresholds permit the specification of conditions using simple AND/OR logic. For example, the specification:

```
MEM=100/10 SWAP=200/30
```

translates into a `loadSched` condition of `mem>=100 && swap>=200` and a `loadStop` condition of `mem < 10 || swap < 30`.

Theory

- The `r15s`, `r1m`, and `r15m` CPU run queue length conditions are compared to the effective queue length as reported by `lsload -E`, which is normalized for multiprocessor hosts. Thresholds for these parameters should be set at appropriate levels for single processor hosts.
- Configure load thresholds consistently across queues. If a low priority queue has higher suspension thresholds than a high priority queue, then jobs in the higher priority queue are suspended before jobs in the low priority queue.

Load thresholds at host level

A shared resource cannot be used as a load threshold in the `Hosts` section of the `lsf.cluster.cluster_name` file.

Configure suspending conditions at queue level

The condition for suspending a job can be specified using the queue-level `STOP_COND` parameter. It is defined by a resource requirement string. Only the `select` section of the resource requirement string is considered when stopping a job. All other sections are ignored.

This parameter provides similar but more flexible functionality for `loadstop`.

If `loadstop` thresholds have been specified, then a job is suspended if either the `STOP_COND` is `TRUE` or the `loadstop` thresholds are exceeded.

1. Modify a queue to suspend a job based on a condition.

For example, suspend a job based on the idle time for desktop machines and availability of swap and memory on compute servers.

Assume `cs` is a Boolean resource defined in the `lsf.shared` file and configured in the `lsf.cluster.cluster_name` file to indicate that a host is a compute server

```
Begin Queue
```

```
STOP_COND= select[ ((!cs && it < 5) || (cs && mem < 15 && swap < 50)) ]
```

```
End Queue
```

View host-level and queue-level suspending conditions

1. View suspending conditions using `bhosts -l` and `bqueues -l`.

View job-level suspending conditions

The thresholds that apply to a particular job are the more restrictive of the host and queue thresholds.

1. Run `bjobs -l`.

View suspend reason

1. Run `bjobs -lp`.

The load threshold that caused LSF to suspend a job, together with the scheduling parameters, display.

Note:

The use of `STOP_COND` affects the suspending reasons as displayed by the `bjobs` command. If `STOP_COND` is specified in the queue and

the `loadStop` thresholds are not specified, the suspending reasons for each individual load index are not displayed.

About resuming suspended jobs

Jobs are suspended to prevent overloading hosts, to prevent batch jobs from interfering with interactive use, or to allow a more urgent job to run. When the host is no longer overloaded, suspended jobs should continue running.

When LSF automatically resumes a job, it invokes the `RESUME` action. The default action for `RESUME` is to send the signal `SIGCONT`.

If there are any suspended jobs on a host, LSF checks the load levels in each dispatch turn.

If the load levels are within the scheduling thresholds for the queue and the host, and all the resume conditions for the queue (`RESUME_COND` in `lsb.queues`) are satisfied, the job is resumed.

If `RESUME_COND` is not defined, then the `loadSched` thresholds are used to control resuming of jobs: all the `loadSched` thresholds must be satisfied for the job to be resumed. The `loadSched` thresholds are ignored if `RESUME_COND` is defined.

Jobs from higher priority queues are checked first. To prevent overloading the host again, only one job is resumed in each dispatch turn.

Specify resume condition

1. Use `RESUME_COND` in `lsb.queues` to specify the condition that must be satisfied on a host if a suspended job is to be resumed.

Only the `select` section of the resource requirement string is considered when resuming a job. All other sections are ignored.

View resume thresholds

1. Run `bjobs -l`.

The scheduling thresholds that control when a job is resumed display.

Pre-Execution and Post-Execution Processing

The pre- and post-execution processing feature provides a way to run commands on the execution host prior to and after completion of LSF jobs. Use pre-execution commands to set up an execution host with the required directories, files, software licenses, environment, and user permissions. Use post-execution commands to define post-job processing such as cleaning up job files or transferring job output.

About pre- and post-execution processing

You can use the pre- and post-execution processing feature to run commands before a batch job starts or after it finishes. Typical uses of this feature include the following:

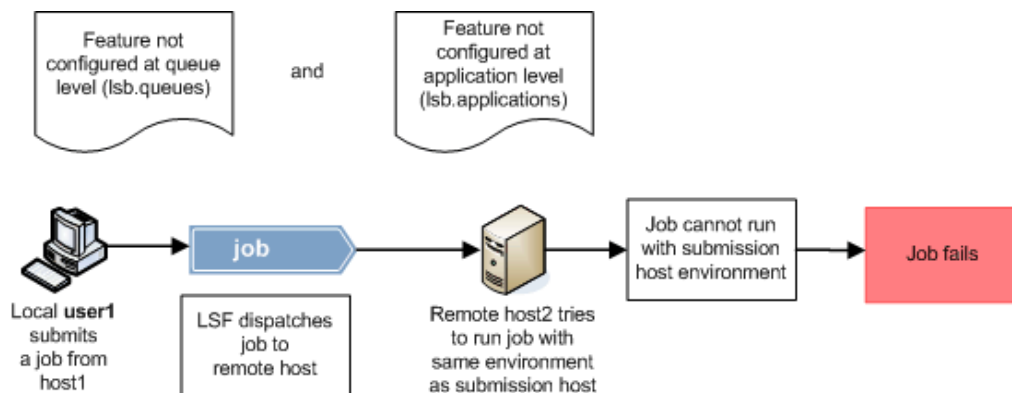
- Reserving resources such as tape drives and other devices not directly configurable in LSF
- Making job-starting decisions in addition to those directly supported by LSF
- Creating and deleting scratch directories for a job
- Customizing scheduling based on the exit code of a pre-execution command
- Checking availability of software licenses
- Assigning jobs to run on specific processors on SMP machines
- Transferring data files needed for job execution
- Modifying system configuration files before and after job execution
- Using a post-execution command to clean up a state left by the pre-execution command or the job

Pre-execution and post-execution commands can be defined at the queue, application, and job levels.

The command path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory, file name, and expanded values for %J (*job_ID*) and %I (*index_ID*).

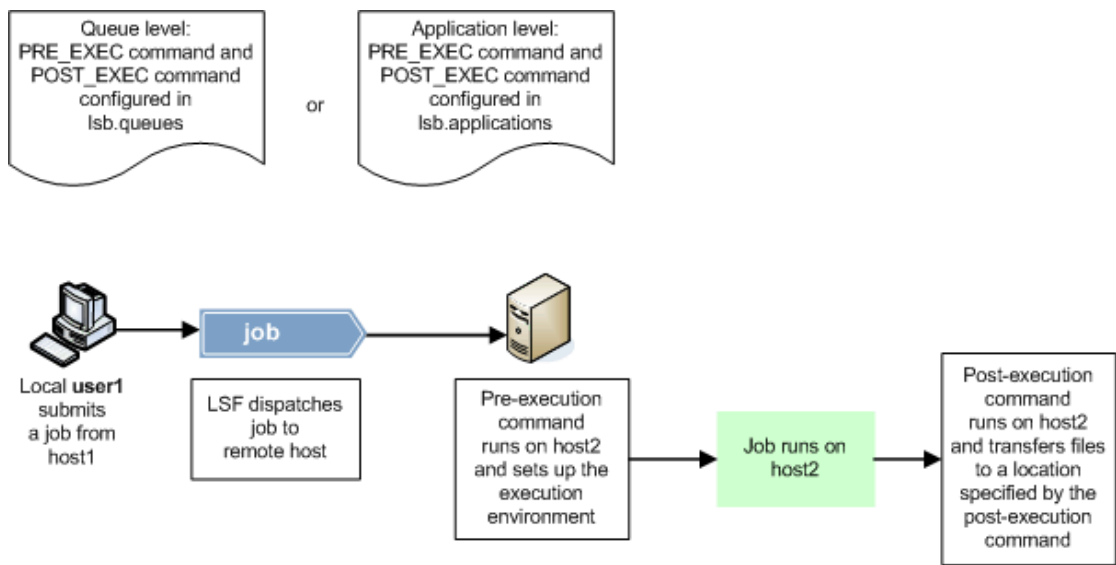
When JOB_INCLUDE_POSTPROC is defined in an application profile, a job is considered in RUN state while the job is in post exec stage (which is DONE state for regular jobs). When the job is also resizable, job grow requests are ignored. However job shrink requests can be processed. For either case, LSF does not invoke the job resized notification command.

Default behavior (feature not enabled)



With pre- and post-execution processing enabled at the queue or application level

The following example illustrates how pre- and post-execution processing works for setting the environment prior to job execution and for transferring resulting files after the job runs.



Any executable command line can serve as a pre-execution or post-execution command. By default, the commands run under the same user account, environment, home directory, and working directory as the job. For parallel jobs, the commands run on the first execution host.

Scope

Applicability	Details
Operating system	<ul style="list-style-type: none">• UNIX• Windows• A mix of UNIX and Windows hosts
Dependencies	<ul style="list-style-type: none">• UNIX and Windows user accounts must be valid on all hosts in the cluster and must have the correct permissions to successfully run jobs.• On a Windows Server 2003, x64 Edition platform, users must have read and execute privileges for <code>cmd.exe</code>.
Limitations	<ul style="list-style-type: none">• Applies to batch jobs only (jobs submitted using the <code>bsub</code> command)

Configuration to enable pre- and post-execution processing

The pre- and post-execution processing feature is enabled by defining at least one of the parameters `PRE_EXEC` or `POST_EXEC` at the application or queue level, or by using the `-E` option of the `bsub` command to specify a pre-execution command. In some situations, specifying a queue-level or application-level pre-execution command can have advantages over requiring users to use `bsub -E`. For example, license checking can be set up at the queue or application level so that users do not have to enter a pre-execution command every time they submit a job.

Configuration file	Parameter and syntax	Behavior
l sb. queues	<code>PRE_EXEC=command</code>	<ul style="list-style-type: none"> Enables pre-execution processing at the queue level. The pre-execution command runs on the execution host before the job starts. If the <code>PRE_EXEC</code> command exits with a non-zero exit code, LSF requeues the job to the front of the queue. The <code>PRE_EXEC</code> command uses the same environment variable values as the job.
	<code>POST_EXEC=command</code>	<ul style="list-style-type: none"> Enables post-execution processing at the queue level. The <code>POST_EXEC</code> command uses the same environment variable values as the job. The post-execution command for the queue remains associated with the job. The original post-execution command runs even if the job is requeued or if the post-execution command for the queue is changed after job submission. Before the post-execution command runs, <code>LSB_JOBEXIT_STAT</code> is set to the exit status of the job. The success or failure of the post-execution command has no effect on <code>LSB_JOBEXIT_STAT</code>. The post-execution command runs after the job finishes, even if the job fails. Specify the environment variable <code>\$USER_POSTEXEC</code> to allow UNIX users to define their own post-execution commands.

Configuration file	Parameter and syntax	Behavior
lsb. applications	PRE_EXEC=command	<ul style="list-style-type: none"> Enables pre-execution processing at the application level. The pre-execution command runs on the execution host before the job starts. If the PRE_EXEC command exits with a non-zero exit code, LSF requeues the job to the front of the queue. The PRE_EXEC command uses the same environment variable values as the job.
	POST_EXEC=command	<ul style="list-style-type: none"> Enables post-execution processing at the application level. The POST_EXEC command uses the same environment variable values as the job. The post-execution command for the application profile remains associated with the job. The original post-execution command runs even if the job is moved to a different application profile or is requeued, or if the post-execution command for the original application profile is changed after job submission. Before the post-execution command runs, LSB_JOBEXIT_STAT is set to the exit status of the job. The success or failure of the post-execution command has no effect on LSB_JOBEXIT_STAT. The post-execution command runs after the job finishes, even if the job fails. Specify the environment variable \$USER_POSTEXEC to allow UNIX users to define their own post-execution commands.

Examples

The following queue specifies the pre-execution command `/usr/share/lsf/pri_prexec` and the post-execution command `/usr/share/lsf/pri_postexec`.

```

Begin Queue
QUEUE_NAME      = priority
PRIORITY        = 43
NICE            = 10
PRE_EXEC       = /usr/share/lsf/pri_prexec
POST_EXEC      = /usr/share/lsf/pri_postexec
End Queue

```

The following application specifies the pre-execution `/usr/share/lsf/catia_prexec` and the post-execution command `/usr/share/lsf/catia_postexec`.

```

Begin Application
NAME            = catia
DESCRIPTION     = CATIA V5
CPULIMIT        = 24:0/hostA      # 24 hours of host hostA
FILELIMIT       = 20000
DATA LIMIT      = 20000           # jobs data segment limit
CORELIMIT       = 20000
PROCLIMIT       = 5              # job processor limit
PRE_EXEC       = /usr/share/lsf/catia_prexec
POST_EXEC      = /usr/share/lsf/catia_postexec
REQUEUE_EXIT_VALUES = 55 34 78
End Application

```

Pre- and post-execution processing behavior

Pre- and post-execution processing applies to both UNIX and Windows hosts.

Host type	Environment
UNIX	<ul style="list-style-type: none"> The pre- and post-execution commands run in the <code>/tmp</code> directory under <code>/bin/sh -c</code>, which allows the use of shell features in the commands. The following example shows valid configuration lines: <code>PRE_EXEC= /usr/share/lsf/misc/testq_pre >> /tmp/pre.out</code> <code>POST_EXEC= /usr/share/lsf/misc/testq_post grep -v "Testing. . ."</code> LSF sets the <code>PATH</code> environment variable to <code>PATH='/bin /usr/bin /sbin /usr/sbin'</code> The <code>stdin</code>, <code>stdout</code>, and <code>stderr</code> are set to <code>/dev/null</code>
Windows	<ul style="list-style-type: none"> The pre- and post-execution commands run under <code>cmd.exe /c</code> The standard input, standard output, and standard error are set to <code>NULL</code> The <code>PATH</code> is determined by the setup of the LSF Service

Note:

If the pre-execution or post-execution command is not in your usual execution path, you must specify the full path name of the command.

Order of command execution

Pre-execution commands run in the following order:

1. The queue-level command
2. The application-level or job-level command. If you specify a command at both the application and job levels, the job-level command overrides the application-level command; the application-level command is ignored.

If a pre-execution command is specified at the ...	Then the commands execute in the order of ...
Queue, application, and job levels	<ol style="list-style-type: none"> 1. Queue level 2. Job level
Queue and application levels	<ol style="list-style-type: none"> 1. Queue level 2. Application level
Queue and job levels	<ol style="list-style-type: none"> 1. Queue level 2. Job level
Application and job levels	<ol style="list-style-type: none"> 1. Job level

Post-execution commands run in the following order:

1. The application-level command
2. The queue-level command
3. The job-level command

If both application-level (POST_EXEC in `lsb. applications`) and job-level post-execution commands are specified, job level post-execution overrides application-level post-execution commands.

If a post-execution command is specified at the ...	Then the commands execute in the order of ...
Queue, application, and job levels	<ol style="list-style-type: none"> 1. Job level 2. Queue level
Queue and application levels	<ol style="list-style-type: none"> 1. Application level 2. Queue level
Queue and job levels	<ol style="list-style-type: none"> 1. Job level 2. Queue level

Pre-execution command behavior

A pre-execution command returns information to LSF by means of the exit status. LSF holds the job in the queue until the specified pre-execution command returns an exit code of zero (0). If the pre-execution command exits with a non-zero value, the job pends until LSF tries again to dispatch it. While the job remains in the PEND state, LSF dispatches other jobs to the execution host.

If the pre-execution command exits with a value of 99, the job exits without pending. This allows you to cancel the job if the pre-execution command fails.

You must ensure that the pre-execution command runs without side effects; that is, you should define a pre-execution command that does not interfere with the job itself. For example, if you use the pre-execution command to reserve a resource, you cannot also reserve the same resource as part of the job submission.

LSF users can specify a pre-execution command at job submission. LSF first finds a suitable host on which to run the job and then runs the pre-execution command on that host. If the pre-execution command runs successfully and returns an exit code of zero, LSF runs the job.

Post-execution command behavior

A post-execution command runs after the job finishes, regardless of the exit state of the job. Once a post-execution command is associated with a job, that command runs even if the job fails. You cannot configure the post-execution command to run only under certain conditions.

The resource usage of post-execution processing is not included in the job resource usage calculation, and post-execution command exit codes are not reported to LSF.

If `POST_EXEC=$USER_POSTEXEC` in either `lsb. applications` or `lsb. queues`, UNIX users can define their own post-execution commands:

```
setenv USER_POSTEXEC /path_name
```

where the path name for the post-execution command is an absolute path.

If <code>POST_EXEC=\$USER_POSTEXEC</code> and ...	Then ...
The user defines the <code>USER_POSTEXEC</code> environment variable	<ul style="list-style-type: none">• LSF runs the post-execution command defined by the environment variable <code>USER_POSTEXEC</code>• After the user-defined command runs, LSF reports successful completion of post-execution processing• If the user-defined command fails, LSF reports a failure of post-execution processing
The user does not define the <code>USER_POSTEXEC</code> environment variable	<ul style="list-style-type: none">• LSF reports successful post-execution processing without actually running a post-execution command

Important:

Do not allow users to specify a post-execution command when the pre- and post-execution commands are set to run under the `root` account.

Check job history for a pre-execution script failure

Each time your job tries to run on a host and the pre-execution script fails to run successfully, your job pends until it is dispatched again.

1. Run **`bhist -l job_number`**.

The history of the job displays, including any pending and dispatching on hosts due to pre-execution scripts exiting with an incorrect exit code.

Configuration to modify pre- and post-execution processing

Configuration parameters modify various aspects of pre- and post-execution processing behavior by:

- Preventing a new job from starting until post-execution processing has finished
- Controlling the length of time post-execution processing can run
- Specifying a user account under which the pre- and post-execution commands run
- Controlling how many times pre-execution retries

Configuration to modify when new jobs can start

When a job finishes, `sbatchd` reports a job finish status of `DONE` or `EXIT` to `mbatchd`. This causes LSF to release resources associated with the job, allowing new jobs to start on the execution host before post-execution processing from a previous job has finished.

In some cases, you might want to prevent the overlap of a new job with post-execution processing. Preventing a new job from starting prior to completion of post-execution processing can be configured at the application level or at the job level.

At the job level, the `bsub -w` option allows you to specify job dependencies; the keywords `post_done` and `post_err` cause LSF to wait for completion of post-execution processing before starting another job.

At the application level:

File	Parameter and syntax	Description
lsb. applications lsb. params	JOB_INCLUDE_POSTPROC=Y	<ul style="list-style-type: none"> • Enables completion of post-execution processing before LSF reports a job finish status of <code>DONE</code> or <code>EXIT</code> • Prevents a new job from starting on a host until post-execution processing is finished on that host

- `sbatchd` sends both job finish status (`DONE` or `EXIT`) and post-execution processing status (`POST_DONE` or `POST_ERR`) to `mbatchd` at the same time
- The job remains in the `RUN` state and holds its job slot until post-execution processing has finished
- Job requeue happens (if required) after completion of post-execution processing, not when the job itself finishes
- For job history and job accounting, the job CPU and run times include the post-execution processing CPU and run times
- The job control commands `bstop`, `bkill`, and `brsrm` have no effect during post-execution processing
- If a host becomes unavailable during post-execution processing for a rerunnable job, `mbatchd` sees the job as still in the `RUN` state and reruns the job
- LSF does not preempt jobs during post-execution processing

Configuration to modify the post-execution processing time

Controlling the length of time post-execution processing can run is configured at the application level.

File	Parameter and syntax	Description
lsb. applications lsb. params	JOB_POSTPROC_TIMEOUT= <i>minutes</i>	<ul style="list-style-type: none"> Specifies the length of time, in minutes, that post-execution processing can run. The specified value must be greater than zero. If post-execution processing takes longer than the specified value, sbat chd reports post-execution failure—a status of POST_ERR. On UNIX and Linux, it kills the entire process group of the job's pre-execution processes. On Windows, only the parent process of the pre-execution command is killed when the timeout expires, the child processes of the pre-execution command are not killed. If JOB_INCLUDE_POSTPROC=Y and sbat chd kills the post-execution process group, post-execution processing CPU time is set to zero, and the job's CPU time does not include post-execution CPU time.

Configuration to modify the pre- and post-execution processing user account

Specifying a user account under which the pre- and post-execution commands run is configured at the system level. By default, both the pre- and post-execution commands run under the account of the user who submits the job.

File	Parameter and syntax	Description
lsf. sudoers	LSB_PRE_POST_EXEC_USER= <i>user_name</i>	<ul style="list-style-type: none"> Specifies the user account under which pre- and post-execution commands run (UNIX only) This parameter applies only to pre- and post-execution commands configured at the queue level; pre-execution commands defined at the application or job level run under the account of the user who submits the job If the pre-execution or post-execution commands perform privileged operations that require root permissions on UNIX hosts, specify a value of root You must edit the lsf. sudoers file on all UNIX hosts within the cluster and specify the same user account

Configuration to control how many times pre-execution retries

By default, if job pre-execution fails, LSF retries the job automatically. The job remains in the queue and pre-execution is retried 5 times by default, to minimize any impact to performance and throughput.

Limiting the number of times LSF retries job pre-execution is configured cluster-wide (lsb. params), at the queue level (lsb. queues), and at the application level (lsb. applications). pre-execution retry

in l sb. appl i cat i ons overrides l sb. queues, and l sb. queues overrides l sb. params configuration.

Configuration file	Parameter and syntax	Behavior
l sb. params	LOCAL_MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the local cluster. Specify an integer greater than 0 <p>By default, the number of retries is unlimited.</p>
	MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the remote cluster. Specify an integer greater than 0 <p>By default, the number of retries is 5.</p>
	REMOTE_MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the remote cluster. <p>Equivalent to MAX_PREEEXEC_RETRY</p> <ul style="list-style-type: none"> Specify an integer greater than 0 <p>By default, the number of retries is 5.</p>
l sb. queues	LOCAL_MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the local cluster. Specify an integer greater than 0 <p>By default, the number of retries is unlimited.</p>
	MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the remote cluster. Specify an integer greater than 0 <p>By default, the number of retries is 5.</p>
	REMOTE_MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the remote cluster. <p>Equivalent to MAX_PREEEXEC_RETRY</p> <ul style="list-style-type: none"> Specify an integer greater than 0 <p>By default, the number of retries is 5.</p>
l sb. appl i cat i ons	LOCAL_MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the local cluster. Specify an integer greater than 0 <p>By default, the number of retries is unlimited.</p>

Configuration file	Parameter and syntax	Behavior
	<code>MAX_PREEEXEC_RETRY= integer</code>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the remote cluster. Specify an integer greater than 0 <p>By default, the number of retries is 5.</p>
	<code>REMOTE_MAX_PREEEXEC_R ENTRY=integer</code>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the remote cluster. <p>Equivalent to <code>MAX_PREEEXEC_RETRY</code></p> <ul style="list-style-type: none"> Specify an integer greater than 0 <p>By default, the number of retries is 5.</p>

When pre-execution retry is configured, if a job pre-execution fails and exits with non-zero value, the number of pre-exec retries is set to 1. When the pre-exec retry limit is reached, the job is suspended with PSUSP status.

The number of times that pre-execution is retried includes queue-level, application-level, and job-level pre-execution command specifications. When pre-execution retry is configured, a job will be suspended when the sum of its queue-level pre-exec retry times + application-level pre-exec retry times is greater than the value of the pre-execution retry parameter or if the sum of its queue-level pre-exec retry times + job-level pre-exec retry times is greater than the value of the pre-execution retry parameter.

The pre-execution retry limit is recovered when LSF is restarted and reconfigured. LSF replays the pre-execution retry limit in the PRE_EXEC_START or JOB_STATUS events in lsb. events.

Set host exclusion based on pre-execution scripts

You must know the exit values your pre-execution script exits with that indicate failure.

Any non-zero exit code in a pre-execution script indicates a failure. For those jobs that are designated as rerunable on failure, LSF filters on the pre-execution script failure to determine whether the job that failed in the pre-execution script should exclude the host where the pre-execution script failed. That host is no longer a candidate to run the job.

1. Create a pre-execution script that exits with a specific value if it is unsuccessful.

Example:

```
#!/bin/sh
# Usually, when pre_exec failed due to host reason like
# /tmp is full, we should exit directly to let LSF
# re-dispatch the job to a different host.
# For example:
#   define PREEEXEC_RETRY_EXIT_VALUES = 10 in lsb.params
#   exit 10 when pre_exec detect that /tmp is full.
# LSF will re-dispatch this job to a different host under
# such condition.
DISC=/tmp
PARTITION=df -Ph | grep -w $DISC | awk '{print $6}'`
FREE=`df -Ph | grep -w $DISC | awk '{print $5}' | awk -F% '{print $1}'`
echo "$FREE"
if [ "${FREE}" != "" ]
then
    if [ "${FREE}" -le "2" ] # When there's only 2% available space for
                           # /tmp on this host, we can let LSF
                           # re-dispatch the job to a different host
    then
```

```

        exit 10
    fi
fi
# Sometimes, when pre_exec failed due to nfs server being busy,
# it can succeed if we retry it several times in this script to
# affect LSF performance less.
RETRY=10
while [ $RETRY -gt 0 ]
do
    #mount host_name: /export/home/bill /home/bill
    EXIT=`echo $?`
    if [ $EXIT -eq 0 ]
    then
        RETRY=0
    else
        RETRY=`expr $RETRY - 1`
        if [ $RETRY -eq 0 ]
        then
            exit 99 # We have tried for 9 times.
                    # Something is wrong with nfs server, we need
                    # to fail the job and fix the nfs problem first.
                    # We need to submit the job again after nfs problem
                    # is resolved.
        fi
    fi
done

```

2. In lsb. params, use `PREEXEC_EXCLUDE_HOST_EXIT_VALUES` to set the exit values that indicate the pre-execution script failed to run.

Values from 1-255 are allowed, excepting 99 (reserved value). Separate values with a space.

For the example script above, set `PREEXEC_EXCLUDE_HOST_EXIT_VALUES=10`.

3. (Optional) Define `MAX_PREEXEC_RETRY` to limit the total number of times LSF retries the pre-execution script on hosts.
4. Run **badadmin reconfig**.

If a pre-execution script exits with value 10 (according to the example above), LSF adds this host to an exclusion list and attempts to reschedule the job on another host.

Pre- and post-execution processing commands

Commands for submission

The `bsub -E` option specifies a pre-execution command. Post-execution commands cannot be specified using `bsub`; post-execution processing can only be defined at the queue and application levels.

The `bsub -w` option allows you to specify job dependencies that cause LSF to wait for completion of post-execution processing before starting another job.

Command	Description
<code>bsub -E command</code>	<ul style="list-style-type: none"> Defines the pre-execution command at the job level.
<code>bsub -w 'post_done(j ob_i d "j ob_name")'</code>	<ul style="list-style-type: none"> Specifies the job dependency condition required to prevent a new job from starting on a host until post-execution processing on that host has finished without errors.
<code>bsub -w 'post_err(j ob_i d "j ob_name")'</code>	<ul style="list-style-type: none"> Specifies the job dependency condition required to prevent a new job from starting on a host until post-execution processing on that host has exited with errors.

Commands to monitor

Command	Description
<code>bhi st</code> <code>bhi st -l</code>	<ul style="list-style-type: none"> Displays the host status of all hosts, specific hosts, or specific host groups, including the <code>POST_DONE</code> and <code>POST_ERR</code> states, if the user submitted the job with the <code>-w</code> option of <code>bsub</code>. The CPU and run times shown do not include resource usage for post-execution processing unless the parameter <code>JOB_INCLUDE_POSTPROC</code> is defined in <code>lsb. appl i cat i ons</code> or <code>lsb. params</code>. Displays the job exit code and reason if the pre-exec retry limit is exceeded.
<code>bj obs -l</code>	<ul style="list-style-type: none"> Displays information about pending, running, and suspended jobs. During post-execution processing, the job status will be <code>RUN</code> if the parameter <code>JOB_INCLUDE_POSTPROC</code> is defined in <code>lsb. appl i cat i ons</code> or <code>lsb. params</code>. The resource usage shown does not include resource usage for post-execution processing. Displays the job exit code and reason if the pre-exec retry limit is exceeded.
<code>bacct</code>	<ul style="list-style-type: none"> Displays accounting statistics for finished jobs. The CPU and run times shown do not include resource usage for post-execution processing, unless the parameter <code>JOB_INCLUDE_POSTPROC</code> is defined in <code>lsb. appl i cat i ons</code> or <code>lsb. params</code>.

Commands to control

Command	Description
<code>bmod -E command</code>	<ul style="list-style-type: none"> Changes the pre-execution command at the job level.

Command	Description
<code>bmod -w 'post_done(j ob_id "j ob_name")'</code>	<ul style="list-style-type: none"> Specifies the job dependency condition required to prevent a new job from starting on a host until post-execution processing on that host has finished without errors.
<code>bmod -w 'post_err(j ob_id "j ob_name")'</code>	<ul style="list-style-type: none"> Specifies the job dependency condition required to prevent a new job from starting on a host until post-execution processing on that host has exited with errors.

Commands to display configuration

Command	Description
<code>bapp -l</code>	<ul style="list-style-type: none"> Displays information about application profiles configured in <code>lsb. appl i cat i ons</code>, including the values defined for <code>PRE_EXEC</code>, <code>POST_EXEC</code>, <code>JOB_INCLUDE_POSTPROC</code>, <code>JOB_POSTPROC_TIMEOUT</code>, <code>LOCAL_MAX_PREEEXEC_RETRY</code>, <code>MAX_PREEEXEC_RETRY</code>, and <code>REMOTE_MAX_PREEEXEC_RETRY</code>.
<code>bparams</code>	<ul style="list-style-type: none"> Displays the value of parameters defined in <code>lsb. par ams</code>, including the values defined for <code>LOCAL_MAX_PREEEXEC_RETRY</code>, <code>MAX_PREEEXEC_RETRY</code>, and <code>REMOTE_MAX_PREEEXEC_RETRY</code>.
<code>bqueues -l</code>	<ul style="list-style-type: none"> Displays information about queues configured in <code>lsb. queues</code>, including the values defined for <code>PRE_EXEC</code> and <code>POST_EXEC</code>, <code>LOCAL_MAX_PREEEXEC_RETRY</code>, <code>MAX_PREEEXEC_RETRY</code>, and <code>REMOTE_MAX_PREEEXEC_RETRY</code>.

Use a text editor to view the `lsf. sudoers` configuration file.

49

Job Starters

About job starters

A *job starter* is a specified shell script or executable program that sets up the environment for a job and then runs the job. The job starter and the job share the same environment. This chapter discusses two ways of running job starters in LSF and how to set up and use them.

Some jobs have to run in a particular environment, or require some type of setup to be performed before they run. In a shell environment, job setup is often written into a wrapper shell script file that itself contains a call to start the desired job.

A job starter is a specified wrapper script or executable program that typically performs environment setup for the job, then calls the job itself, which inherits the execution environment created by the job starter. LSF controls the job starter process, rather than the job. One typical use of a job starter is to customize LSF for use with specific application environments, such as Alias Renderer or IBM Rational ClearCase.

Two ways to run job starters

You run job starters two ways in LSF. You can accomplish similar things with either job starter, but their functional details are slightly different.

Command-level

Are user-defined. They run interactive jobs submitted using `l srun`, `l sgrun`, or `ch`. Command-level job starters have no effect on batch jobs, including interactive batch jobs run with `bsub -I`.

Use the `LSF_JOB_STARTER` environment variable to specify a job starter for interactive jobs.

Queue-level

Defined by the LSF administrator, and run batch jobs submitted to a queue defined with the `JOB_STARTER` parameter set. Use `bsub` to submit jobs to queues with job-level job starters.

A queue-level job starter is configured in the queue definition in `lsb.queues`.

Pre-execution commands are not job starters

A job starter differs from a pre-execution command. A pre-execution command must run successfully and exit before the LSF job starts. It can signal LSF to dispatch the job, but because the pre-execution command is an unrelated process, it does not control the job or affect the execution environment of the job. A job starter, however, is the process that LSF controls. It is responsible for invoking LSF and controls the execution environment of the job.

Examples

The following are some examples of job starters:

- In UNIX, a job starter defined as `/bin/ksh -c` causes jobs to be run under a Korn shell environment.
- In Windows, a job starter defined as `C:\cmd.exe /C` causes jobs to be run under a DOS shell environment.

Note:

For job starters that execute on a Windows Server 2003, x64 Edition platform, users must have “Read” and “Execute” privileges for `cmd.exe`.

- Setting the `JOB_STARTER` parameter in `lsb.queues` to `$USER_STARTER` enables users to define their own job starters by defining the environment variable `USER_STARTER`.

Restriction:

`USER_STARTER` can only be used in UNIX clusters. Mixed or Windows-only clusters are not supported.

- Setting a job starter to `make clean` causes the command `make clean` to be run before the user job.

Command-level job starters

A command-level job starter allows you to specify an executable file that does any necessary setup for the job and runs the job when the setup is complete. You can select an existing command to be a job starter, or you can create a script containing a desired set of commands to serve as a job starter.

This section describes how to set up and use a command-level job starter to run interactive jobs.

Command-level job starters have no effect on batch jobs, including interactive batch jobs.

A job starter can also be defined at the queue level using the `JOB_STARTER` parameter. Only the LSF administrator can configure queue-level job starters.

LSF_JOB_STARTER environment variable

Use the `LSF_JOB_STARTER` environment variable to specify a command or script that is the job starter for the interactive job. When the environment variable `LSF_JOB_STARTER` is defined, RES invokes the job starter rather than running the job itself, and passes the job to the job starter as a command-line argument.

Using command-level job starters

- **UNIX:** The job starter is invoked from within a Bourne shell, making the command-line equivalent:

```
/bin/sh -c "$LSF_JOB_STARTER command [argument ...]"
```

 where *command* and *argument* are the command-line arguments you specify in `l srun`, `l sgrun`, or `ch`.
- **Windows:** RES runs the job starter, passing it your commands as arguments:

```
LSF_JOB_STARTER command [argument ...]
```

Examples

UNIX

If you define the `LSF_JOB_STARTER` environment variable using the following C-shell command:

```
% setenv LSF_JOB_STARTER "/bin/sh -c"
```

Then you run a simple C-shell job:

```
% lsrunc "a.out; hostname"
```

The command that actually runs is

```
/bin/sh -c "/bin/sh -c 'a.out; hostname' "
```

The job starter can be a shell script. In the following example, the `LSF_JOB_STARTER` environment variable is set to the Bourne shell script named `job_starter`:

```
$ LSF_JOB_STARTER=/usr/local/job_starter
```

The `job_starter` script contains the following:

```
#!/bin/sh
set term = xterm eval "$@"
```

Windows

If you define the `LSF_JOB_STARTER` environment variable as follows:

```
set LSF_JOB_STARTER=C:\cmd.exe /C
```

Then you run a simple DOS shell job:

```
C:\> lsrun dir /p
```

The command that actually runs is:

```
C: \cmd.exe /C di r /p
```

Queue-level job starters

LSF administrators can define a job starter for an individual queue to create a specific environment for jobs to run in. A queue-level job starter specifies an executable that performs any necessary setup, and then runs the job when the setup is complete. The `JOB_STARTER` parameter in `lsb.queues` specifies the command or script that is the job starter for the queue.

This section describes how to set up and use a queue-level job starter.

Queue-level job starters have no effect on interactive jobs, unless the interactive job is submitted to a queue as an interactive batch job.

LSF users can also select an existing command or script to be a job starter for their interactive jobs using the `LSF_JOB_STARTER` environment variable.

Configure a queue-level job starter

1. Use the `JOB_STARTER` parameter in `lsb.queues` to specify a queue-level job starter in the queue definition. All jobs submitted to this queue are run using the job starter. The jobs are called by the specified job starter process rather than initiated by the batch daemon process.

For example:

```
Begin Queue
JOB_STARTER = xterm -e
End Queue
```

All jobs submitted to this queue are run under an `xterm` terminal emulator.

JOB_STARTER parameter (lsb.queues)

The `JOB_STARTER` parameter in the queue definition (`lsb.queues`) has the following format:

JOB_STARTER=*starter* [*starter*] ["%USRCMD"] [*starter*]

The string *starter* is the command or script that is used to start the job. It can be any executable that can accept a job as an input argument. Optionally, additional strings can be specified.

When starting a job, LSF runs the `JOB_STARTER` command, and passes the shell script containing the job commands as the argument to the job starter. The job starter is expected to do some processing and then run the shell script containing the job commands. The command is run under `/bin/sh -c` and can contain any valid Bourne shell syntax.

%USRCMD string

The special string `%USRCMD` indicates the position of the job starter command in the job command line. By default, the user commands run after the job starter, so the `%USRCMD` string is not usually required. For example, these two job starters both give the same results:

```
JOB_STARTER = /bin/csh -c
JOB_STARTER = /bin/csh -c "%USRCMD"
```

You must enclose the `%USRCMD` string in quotes. The `%USRCMD` string can be followed by additional commands. For example:

```
JOB_STARTER = /bin/csh -c "%USRCMD; sleep 10"
```

If a user submits the following job to the queue with this job starter:

bsub myjob arguments

the command that actually runs is:

```
/bin/csh -c "myjob arguments; sleep 10"
```

Control the execution environment with job starters

In some cases, using `bsub -L` does not result in correct environment settings on the execution host. LSF provides the following two job starters:

- `preservestarter` — preserves the default environment of the execution host. It does not include any submission host settings.
- `augmentstarter` — augments the default user environment of the execution host by adding settings from the submission host that are not already defined on the execution host

`bsub -L` cannot be used for a Windows execution host.

Where the job starter executables are located

By default, the job starter executables are installed in `LSF_BINDIR`. If you prefer to store them elsewhere, make sure they are in a directory that is included in the default `PATH` on the execution host.

For example:

- On Windows, put the job starter under `%WINDIR%`.
- On UNIX, put the job starter under `$HOME/bin`.

Source code for the job starters

The source code for the job starters is installed in `LSF_MISC/examples`.

Add to the initial login environment

By default, the `preservestarter` job starter preserves the environment that RES establishes on the execution host, and establishes an initial login environment for the user with the following variables from the user's login environment on the execution host:

- `HOME`
- `USER`
- `SHELL`
- `LOGNAME`

Any additional environment variables that exist in the user's login environment on the submission host must be added to the job starter source code.

Example

A user's `.login` script on the submission host contains the following setting:

```
if ($TERM != "xterm") then
    set TERM=`tset - -Q -m 'switch: ?vt100'` . . . .
else
    stty -tabs
endif
```

The `TERM` environment variable must also be included in the environment on the execution host for login to succeed. If it is missing in the job starter, the login fails, the job starter may fail as well. If the job starter can continue with only the initial environment settings, the job may execute correctly, but this is not likely.

50

Job Controls

Job Controls

After a job is started, it can be killed, suspended, or resumed by the system, an LSF user, or LSF administrator. LSF job control actions cause the status of a job to change. This chapter describes how to configure job control actions to override or augment the default job control actions.

Default job control actions

After a job is started, it can be killed, suspended, or resumed by the system, an LSF user, or LSF administrator. LSF job control actions cause the status of a job to change. LSF supports the following default actions for job controls:

- SUSPEND
- RESUME
- TERMINATE

On successful completion of the job control action, the LSF job control commands cause the status of a job to change.

The environment variable `LS_EXEC_T` is set to the value `JOB_CONTROLS` for a job when a job control action is initiated.

SUSPEND action

Change a running job from RUN state to one of the following states:

- USUSP or PSUSP in response to `bst op`
- SSUSP state when the LSF system suspends the job

The default action is to send the following signals to the job:

- SIGTSTP for parallel or interactive jobs. SIGTSTP is caught by the master process and passed to all the slave processes running on other hosts.
- SIGSTOP for sequential jobs. SIGSTOP cannot be caught by user programs. The SIGSTOP signal can be configured with the `LSB_SIGSTOP` parameter in `lsf.conf`.

LSF invokes the SUSPEND action when:

- The user or LSF administrator issues a `bst op` or `bkill` command to the job
- Load conditions on the execution host satisfy *any* of:
 - The suspend conditions of the queue, as specified by the `STOP_COND` parameter in `lsb.queues`
 - The scheduling thresholds of the queue or the execution host
- The run window of the queue closes
- The job is preempted by a higher priority job

RESUME action

Change a suspended job from SSUSP, USUSP, or PSUSP state to the RUN state. The default action is to send the signal SIGCONT.

LSF invokes the RESUME action when:

- The user or LSF administrator issues a `brresume` command to the job
- Load conditions on the execution host satisfy *all* of:

- The resume conditions of the queue, as specified by the RESUME_COND parameter in `lsb.queues`
- The scheduling thresholds of the queue and the execution host
- A closed run window of the queue opens again
- A preempted job finishes

TERMINATE action

Terminate a job. This usually causes the job change to EXIT status. The default action is to send SIGINT first, then send SIGTERM 10 seconds after SIGINT, then send SIGKILL 10 seconds after SIGTERM. The delay between signals allows user programs to catch the signals and clean up before the job terminates.

To override the 10 second interval, use the parameter JOB_TERMINATE_INTERVAL in the `lsb.params` file. See the *Platform LSF Configuration Reference* for information about the `lsb.params` file.

LSF invokes the TERMINATE action when:

- The user or LSF administrator issues a `bki ll` or `brequeue` command to the job
- The TERMINATE_WHEN parameter in the queue definition (`lsb.queues`) causes a SUSPEND action to be redirected to TERMINATE
- The job reaches its CPULIMIT, MEMLIMIT, RUNLIMIT or PROCESSLIMIT

If the execution of an action is in progress, no further actions are initiated unless it is the TERMINATE action. A TERMINATE action is issued for all job states except PEND.

Windows job control actions

On Windows, actions equivalent to the UNIX signals have been implemented to do the default job control actions. Job control messages replace the SIGINT and SIGTERM signals, but only customized applications will be able to process them. Termination is implemented by the `TerminateProcess()` system call.

See *Platform LSF Programmer's Guide* for more information about LSF signal handling on Windows.

Configure job control actions

Several situations may require overriding or augmenting the default actions for job control. For example:

- Notifying users when their jobs are suspended, resumed, or terminated
- An application holds resources (for example, licenses) that are not freed by suspending the job. The administrator can set up an action to be performed that causes the license to be released before the job is suspended and re-acquired when the job is resumed.
- The administrator wants the job checkpointed before being:
 - Suspended when a run window closes
 - Killed when the RUNLIMIT is reached
- A distributed parallel application must receive a catchable signal when the job is suspended, resumed or terminated to propagate the signal to remote processes.

To override the default actions for the SUSPEND, RESUME, and TERMINATE job controls, specify the JOB_CONTROLS parameter in the queue definition in `lsb.queues`.

JOB_CONTROLS parameter (lsb.queues)

The JOB_CONTROLS parameter has the following format:

```
Begin Queue
...
```

```
JOB_CONTROLS = SUSPEND[ signal | CHPNT | command ] \
RESUME[ signal | command ] \
TERMI NATE[ signal | CHPNT | command ]
...
End Queue
```

When LSF needs to suspend, resume, or terminate a job, it invokes one of the following actions as specified by SUSPEND, RESUME, and TERMINATE.

signal

A UNIX signal name (for example, SIGTSTP or SIGTERM). The specified signal is sent to the job.

The same set of signals is not supported on all UNIX systems. To display a list of the symbolic names of the signals (without the SIG prefix) supported on your system, use the `kill -l` command.

CHKPNT

Checkpoint the job. Only valid for SUSPEND and TERMINATE actions.

- If the SUSPEND action is CHPNT, the job is checkpointed and then stopped by sending the SIGSTOP signal to the job automatically.
- If the TERMINATE action is CHPNT, then the job is checkpointed and killed automatically.

command

A `/bin/sh` command line.

- Do not quote the command line inside an action definition.
- Do not specify a signal followed by an action that triggers the same signal (for example, do not specify `JOB_CONTROLS=TERMI NATE[bkill]` or `JOB_CONTROLS=TERMI NATE[brequeue]`). This will cause a deadlock between the signal and the action.

Use a command as a job control action

- The command line for the action is run with `/bin/sh -c` so you can use shell features in the command.
- The command is run as the user of the job.
- All environment variables set for the job are also set for the command action. The following additional environment variables are set:
 - `LSB_JOBPGIDS`: A list of current process group IDs of the job
 - `LSB_JOBPIIDS`: A list of current process IDs of the job
 - For the SUSPEND action command, the environment variables `LSB_SUSP_REASONS` and `LSB_SUSP_SUBREASONS` are also set. Use them together in your custom job control to determine the exact load threshold that caused a job to be suspended.
 - `LSB_SUSP_REASONS`: An integer representing a bitmap of suspending reasons as defined in `lsbat ch. h`. The suspending reason can allow the command to take different actions based on the reason for suspending the job.
 - `LSB_SUSP_SUBREASONS`: An integer representing the load index that caused the job to be suspended. When the suspending reason `SUSP_LOAD_REASON` (suspended by load) is set in `LSB_SUSP_REASONS`, `LSB_SUSP_SUBREASONS` is set to one of the load index values defined in `lsf. h`.

- The standard input, output, and error of the command are redirected to the NULL device, so you cannot tell directly whether the command runs correctly. The default null device on UNIX is `/dev/null`.
- You should make sure the command line is correct. If you want to see the output from the command line for testing purposes, redirect the output to a file inside the command line.

TERMINATE job actions

Use caution when configuring TERMINATE job actions that do more than just kill a job. For example, resource usage limits that terminate jobs change the job state to SSUSP while LSF waits for the job to end. If the job is not killed by the TERMINATE action, it remains suspended indefinitely.

TERMINATE_WHEN parameter (lsb.queues)

In certain situations you may want to terminate the job instead of calling the default SUSPEND action. For example, you may want to kill jobs if the run window of the queue is closed. Use the TERMINATE_WHEN parameter to configure the queue to invoke the TERMINATE action instead of SUSPEND.

See the *Platform LSF Configuration Reference* for information about the `lsb.queues` file and the TERMINATE_WHEN parameter.

Syntax

```
TERMI NATE_WHEN = [ LOAD] [ PREEMPT] [ WI NDOW]
```

Example

The following defines a night queue that will kill jobs if the run window closes.

```
Begin Queue
NAME           = ni ght
RUN_WI NDOW    = 20: 00- 08: 00
TERMI NATE_WHEN = WI NDOW
JOB_CONTROLS   = TERMI NATE[ ki ll -KILL $LSB_JOBIDS; \
    echo "job $LSB_JOBID killed by queue run window" | \
    mail $USER ]
End Queue
```

LSB_SIGSTOP parameter (lsf.conf)

Use LSB_SIGSTOP to configure the SIGSTOP signal sent by the default SUSPEND action.

If LSB_SIGSTOP is set to anything other than SIGSTOP, the SIGTSTP signal that is normally sent by the SUSPEND action is not sent. For example, if `LSB_SIGSTOP=SIGKILL`, the three default signals sent by the TERMINATE action (SIGINT, SIGTERM, and SIGKILL) are sent 10 seconds apart.

Avoid signal and action deadlock

Do not configure a job control to contain the signal or command that is the same as the action associated with that job control. This will cause a deadlock between the signal and the action.

For example, the `bki ll` command uses the TERMINATE action, so a deadlock results when the TERMINATE action itself contains the `bki ll` command.

Any of the following job control specifications will cause a deadlock:

- `JOB_CONTROLS=TERMI NATE[bki ll]`
- `JOB_CONTROLS=TERMI NATE[brequeue]`
- `JOB_CONTROLS=RESUME[bresume]`

- `JOB_CONTROLS=SUSPEND[bstop]`

Customize cross-platform signal conversion

LSF supports signal conversion between UNIX and Windows for remote interactive execution through RES.

On Windows, the CTRL+C and CTRL+BREAK key combinations are treated as signals for console applications (these signals are also called console control actions).

LSF supports these two Windows console signals for remote interactive execution. LSF regenerates these signals for user tasks on the execution host.

Default signal conversion

In a mixed Windows/UNIX environment, LSF has the following default conversion between the Windows console signals and the UNIX signals:

Windows	UNIX
CTRL+C	SIGINT
CTRL+BREAK	SIGQUIT

For example, if you issue the `l srunk` or `bsub -I` commands from a Windows console but the task is running on an UNIX host, pressing the CTRL+C keys will generate a UNIX SIGINT signal to your task on the UNIX host. The opposite is also true.

Custom signal conversion

For `l srunk` (but not `bsub -I`), LSF allows you to define your own signal conversion using the following environment variables:

- `LSF_NT2UNIX_CTRLC`
- `LSF_NT2UNIX_CTRLB`

For example:

- `LSF_NT2UNIX_CTRLC=SIGXXX`
- `LSF_NT2UNIX_CTRLB=SIGYYY`

Here, SIGXXX/SIGYYY are UNIX signal names such as SIGQUIT, SIGTINT, etc. The conversions will then be: CTRL+C=SIGXXX and CTRL+BREAK=SIGYYY.

If both `LSF_NT2UNIX_CTRLC` and `LSF_NT2UNIX_CTRLB` are set to the same value (`LSF_NT2UNIX_CTRLC=SIGXXX` and `LSF_NT2UNIX_CTRLB=SIGXXX`), CTRL+C will be generated on the Windows execution host.

For `bsub -I`, there is no conversion other than the default conversion.

External Job Submission and Execution Controls

The job submission and execution controls feature enables you to use external, site-specific executables to validate, modify, and reject jobs, transfer data, and modify the job execution environment. By writing external submission (`esub`) and external execution (`eexec`) binaries or scripts, you can, for example, prevent the overuse of resources, specify execution hosts, or set required environment variables based on the job submission options.

About job submission and execution controls

The job submission and execution controls feature uses the executables `esub` and `eexec` to control job options and the job execution environment.

External submission (`esub`)

An `esub` is an executable that you write to meet the job requirements at your site. The following are some of the things that you can use an `esub` to do:

- Validate job options
- Change the job options specified by a user
- Change user environment variables on the submission host (at job submission only)
- Reject jobs (at job submission only)
- Pass data to `stdi n` of `eexec`

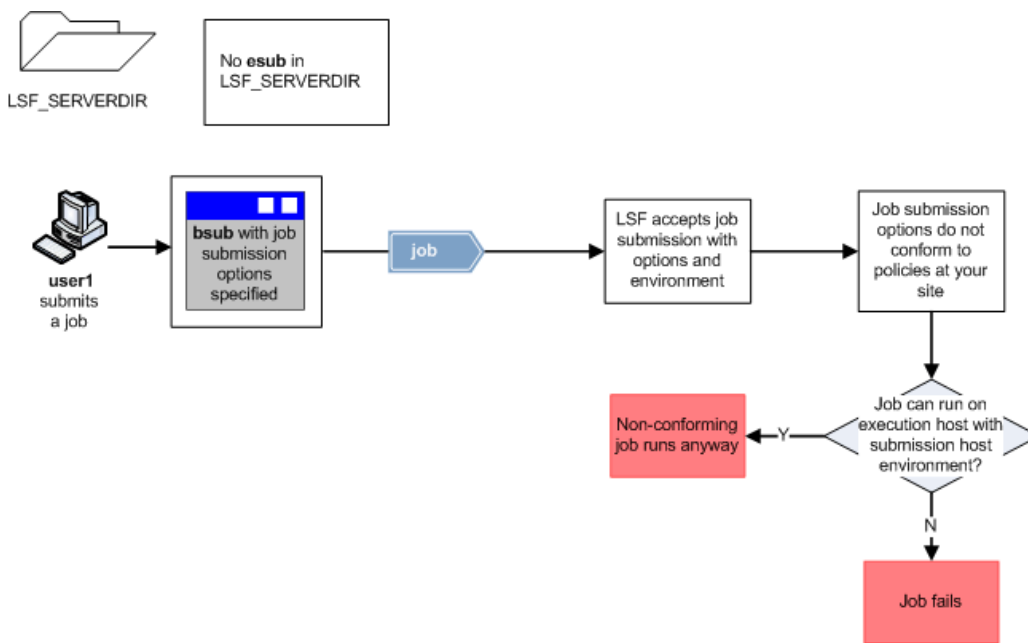
When a user submits a job using `bsub` or modifies a job using `bmod`, LSF runs the `esub` executable(s) on the submission host before accepting the job. If the user submitted the job with options such as `-R` to specify required resources or `-q` to specify a queue, an `esub` can change the values of those options to conform to resource usage policies at your site.

Note:

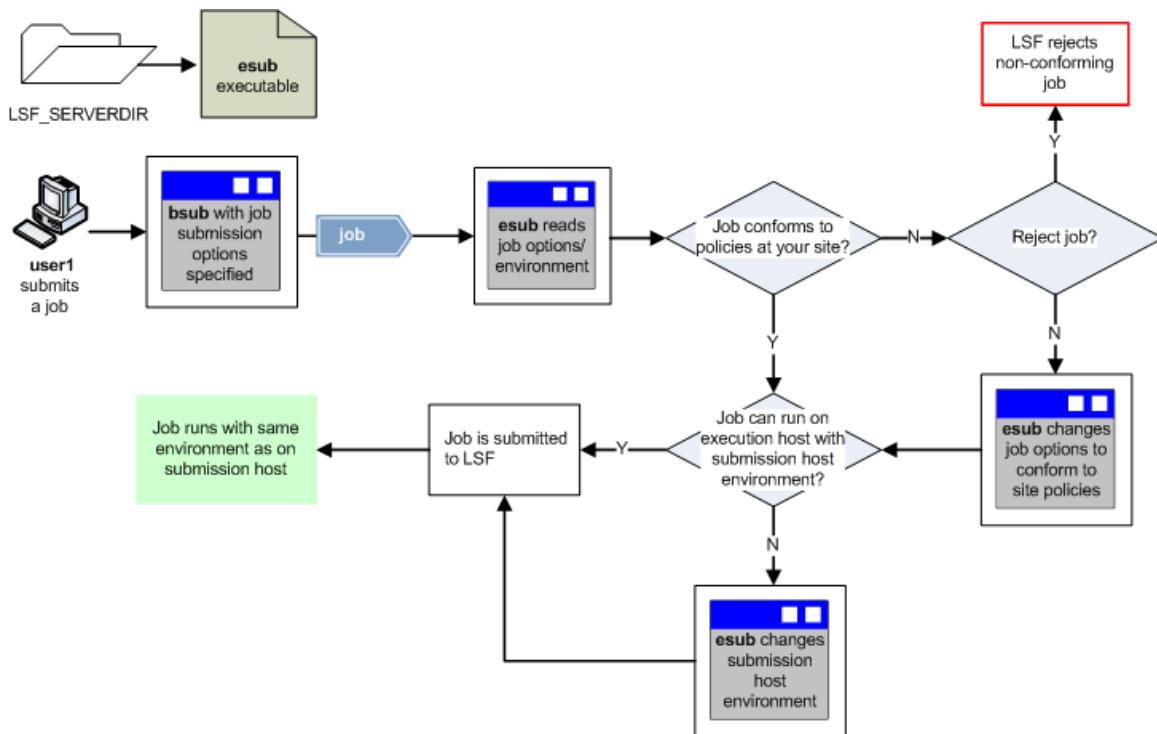
When compound resource requirements are used at any level, an `esub` can create job-level resource requirements which overwrite most application-level and queue-level resource requirements. `-R` merge rules are explained in detail in *Administering Platform LSF*.

An `esub` can also change the user environment on the submission host prior to job submission so that when LSF copies the submission host environment to the execution host, the job runs on the execution host with the values specified by the `esub`. For example, an `esub` can add user environment variables to those already associated with the job.

Use of `esub` not enabled



With esub enabled



An `esub` executable is typically used to enforce site-specific job submission policies and command-line syntax by validating or pre-parsing the command line. The file indicated by the environment variable `LSB_SUB_PARM_FILE` stores the values submitted by the user. An `esub` reads the `LSB_SUB_PARM_FILE` and then accepts or changes the option values or rejects the job. Because an `esub` runs before job submission, using an `esub` to reject incorrect job submissions improves overall system performance by reducing the load on the master batch daemon (`mbat chd`).

An `esub` can be used to:

- Reject any job that requests more than a specified number of CPUs
- Change the submission queue for specific user accounts to a higher priority queue
- Check whether the job specifies an application and, if so, submit the job to the correct application profile

Note:

If an `esub` executable fails, the job will still be submitted to LSF.

Multiple esub executables

LSF provides a master external submission executable (`LSF_SERVERDIR/mesub`) that supports the use of application-specific `esub` executables. Users can specify one or more `esub` executables using the `-a` option of `bsub` or `bmod`. When a user submits or modifies a job or when a user restarts a job that was submitted or modified with the `-a` option included, `mesub` runs the specified `esub` executables.

An LSF administrator can specify one or more mandatory `esub` executables by defining the parameter `LSB_ESUB_METHOD` in `lsf.conf`. If a mandatory `esub` is defined, `mesub` runs the mandatory `esub` for all jobs submitted to LSF in addition to any `esub` executables specified with the `-a` option.

The naming convention is `esub. application`. LSF always runs the executable named "esub" (without *application*) if it exists in LSF_SERVERDIR.

Note:

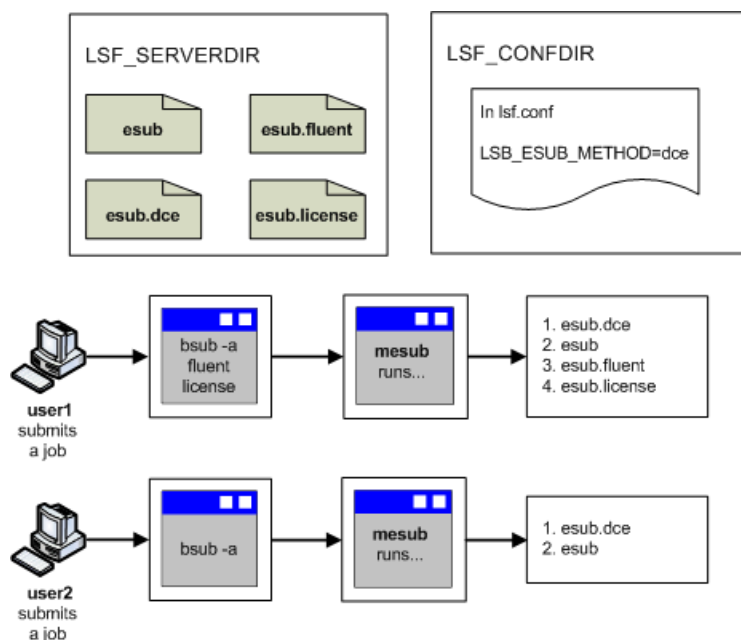
All esub executables must be stored in the LSF_SERVERDIR directory defined in `lsf.conf`.

The mesub runs multiple esub executables in the following order:

1. The mandatory esub or esubs specified by LSB_ESUB_METHOD in `lsf.conf`
2. Any executable with the name "esub" in LSF_SERVERDIR
3. One or more esubs in the order specified by `bsub -a`

Example of multiple esub execution

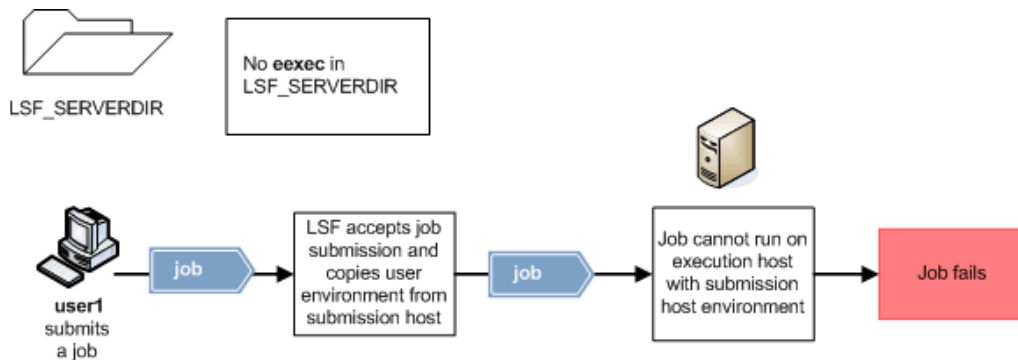
An esub runs only once, even if it is specified by both the `bsub -a` option and the parameter LSB_ESUB_METHOD.



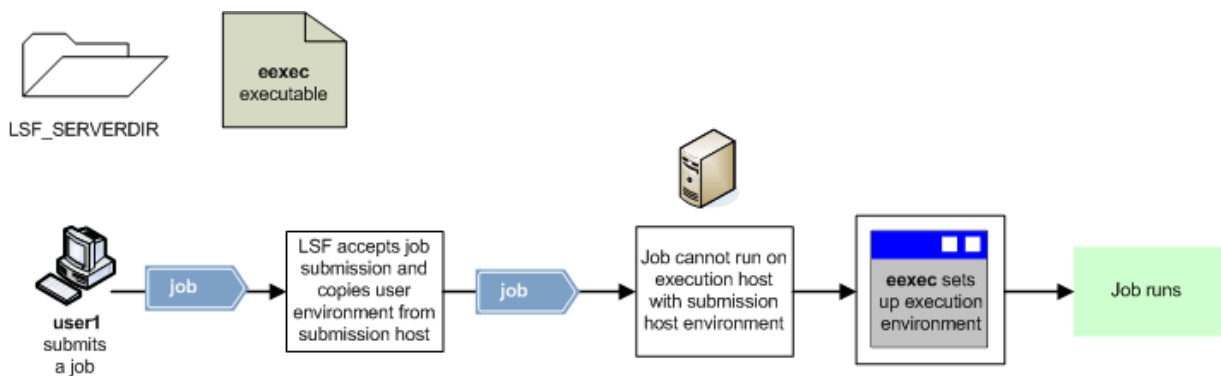
External execution (eexec)

An eexec is an executable that you write to control the job environment on the execution host.

Use of eexec not enabled



With eexec enabled



The following are some of the things that you can use an eexec to do:

- Set up the user environment variables on the execution host
- Monitor job state or resource usage
- Receive data from `stdout` of `esub`
- Run a shell script to create and populate environment variables needed by jobs
- Monitor the number of tasks running on a host and raise a flag when this number exceeds a pre-determined limit
- Pass DCE credentials and AFS tokens using a combination of `esub` and `eexec` executables; LSF functions as a pipe for passing data from the `stdout` of `esub` to the `stdin` of `eexec`

An `eexec` can change the user environment variable values transferred from the submission host so that the job runs on the execution host with a different environment.

For example, if you have a mixed UNIX and Windows cluster, the submission and execution hosts might use different operating systems. In this case, the submission host environment might not meet the job requirements when the job runs on the execution host. You can use an `eexec` to set the correct user environment between the two operating systems.

Typically, an `eexec` executable is a shell script that creates and populates the environment variables required by the job. An `eexec` can also monitor job execution and enforce site-specific resource usage policies.

If an `eexec` executable exists in the directory specified by `LSF_SERVERDIR`, LSF invokes that `eexec` for all jobs submitted to the cluster. By default, LSF runs `eexec` on the execution host before the job starts. The job process that invokes `eexec` waits for `eexec` to finish before continuing with job execution.

Unlike a pre-execution command defined at the job, queue, or application levels, an `eexec`:

- Runs at job start, finish, or checkpoint
- Allows the job to run without pending if `eexec` fails; `eexec` has no effect on the job state
- Runs for all jobs, regardless of queue or application profile

Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> • UNIX and Linux • Windows
Security	<ul style="list-style-type: none"> • Data passing between <code>esub</code> on the submission host and <code>eexec</code> on the execution host is not encrypted.
Job types	<ul style="list-style-type: none"> • Batch jobs submitted with <code>bsub</code> or modified by <code>bmod</code>. • Batch jobs restarted with <code>brstart</code>. • Interactive tasks submitted with <code>l srun</code> and <code>l sgrun</code> (<code>eexec</code> only).
Dependencies	<ul style="list-style-type: none"> • UNIX and Windows user accounts must be valid on all hosts in the cluster, or the correct type of account mapping must be enabled. <ul style="list-style-type: none"> • For a mixed UNIX/Windows cluster, UNIX/Windows user account mapping must be enabled. • For a cluster with a non-uniform user name space, between-host account mapping must be enabled. • For a MultiCluster environment with a non-uniform user name space, cross-cluster user account mapping must be enabled. • User accounts must have the correct permissions to successfully run jobs. • An <code>eexec</code> that requires <code>root</code> privileges to run on UNIX, must be configured to run as the <code>root</code> user.
Limitations	<ul style="list-style-type: none"> • Only an <code>esub</code> invoked by <code>bsub</code> can change the job environment on the submission host. An <code>esub</code> invoked by <code>bmod</code> or <code>brstart</code> cannot change the environment. • Any <code>esub</code> messages provided to the user must be directed to standard error, not to standard output. Standard output from any <code>esub</code> is automatically passed to <code>eexec</code>. • An <code>eexec</code> can handle only one standard output stream from an <code>esub</code> as standard input to <code>eexec</code>. You must make sure that your <code>eexec</code> handles standard output from correctly if any <code>esub</code> writes to standard output. • The <code>esub/eexec</code> combination cannot handle daemon authentication. To configure daemon authentication, you must enable external authentication, which uses the <code>eauth</code> executable.

Configuration to enable job submission and execution controls

This feature is enabled by the presence of at least one `esub` or one `eexec` executable in the directory specified by the parameter `LSF_SERVERDIR` in `lsf.conf`. LSF does not include a default `esub` or `eexec`; you should write your own executables to meet the job requirements of your site.

Executable file	UNIX naming convention	Windows naming convention
esub	LSF_SERVERDIR/ <i>esub. application</i>	LSF_SERVERDIR <i>\esub. application.exe</i>
		LSF_SERVERDIR <i>\esub. application.bat</i>
eexec	LSF_SERVERDIR/eexec	LSF_SERVERDIR\eeexec.exe
		LSF_SERVERDIR\eeexec.bat

The name of your `esub` should indicate the application with which it runs. For example: `esub.fluent`.

Restriction:

The name `esub.user` is reserved. Do not use the name `esub.user` for an application-specific `esub`.

Valid file names contain only alphanumeric characters, underscores (`_`), and hyphens (`-`).

Once the `LSF_SERVERDIR` contains one or more `esub` executables, users can specify the `esub` executables associated with each job they submit. If an `eexec` exists in `LSF_SERVERDIR`, LSF invokes that `eexec` for all jobs submitted to the cluster.

The following `esub` executables are provided as separate packages, available from Platform Computing Inc. upon request:

- `esub.openmpi` : OpenMPI job submission
- `esub.pvm`: PVM job submission
- `esub.poe` : POE job submission
- `esub.lsdyna` : LS-Dyna job submission
- `esub.fluent` : FLUENT job submission
- `esub.afs` or `esub.dce`: AFS or DCE security
- `esub.lammpi` LAM/MPI job submission
- `esub.mpi ch_gm`: MPICH-GM job submission
- `esub.intelmpi` : Intel® MPI job submission
- `esub.bproc`: Beowulf Distributed Process Space (BProc) job submission
- `esub.mpi ch2`: MPICH2 job submission
- `esub.mpi chp4`: MPICH-P4 job submission
- `esub.mvapi ch`: MVAPICH job submission
- `esub.tv`, `esub.tvlammpi`, `esub.tvmpi ch_gm`, `esub.tvpoe`: TotalView® debugging for various MPI applications.

Environment variables used by esub

When you write an `esub`, you can use the following environment variables provided by LSF for the `esub` execution environment:

LSB_SUB_PARM_FILE

Points to a temporary file that LSF uses to store the `bsub` options entered in the command line. An `esub` reads this file at job submission and either accepts the values, changes the values, or rejects the job. Job submission options are stored as name-value pairs on separate lines with the format `option_name=value`.

For example, if a user submits the following job,

```
bsub -q normal -x -P myproject -R "r1m rusage[mem=100]" -n 90 myjob
```

The `LSB_SUB_PARM_FILE` contains the following lines:

```
LSB_SUB_QUEUE="normal "
LSB_SUB_EXCLUSIVE=Y
LSB_SUB_RES_REQ="r1m usage[mem=100] "
LSB_SUB_PROJECT_NAME="myproject "
LSB_SUB_COMMAND_LINE="myjob"
LSB_SUB_NUM_PROCESSORS=90
LSB_SUB_MAX_NUM_PROCESSORS=90
```

An `esub` can change any or all of the job options by writing to the file specified by the environment variable `LSB_SUB_MODIFY_FILE`.

The temporary file pointed to by `LSB_SUB_PARM_FILE` stores the following information:

Option	bsub or bmod option	data type	Description
LSB_SUB_ADDITIONAL	-a	string	String that contains the application name or names of the <code>esub</code> executables requested by the user. Restriction: This is the only option that an <code>esub</code> cannot change or add at job submission.
LSB_SUB_BEGIN_TIME	-b	integer	Begin time, in seconds since 00:00:00 GMT, Jan. 1, 1970
LSB_SUB_CHKPNT_DIR	-k	string	Checkpoint directory The file path of the checkpoint directory can contain up to 4000 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.
LSB_SUB_COMMAND_LINE	bsub job command argument	string	<code>LSB_SUB_COMMANDNAME</code> must be set in <code>lsf.conf</code> to enable <code>esub</code> to use this variable.
LSB_SUB_CHKPNT_PERIOD	-k	integer	Checkpoint period

Option	bsub or bmod option	data type	Description
LSB_SUB_DEPEND_COND	-w	string	Dependency condition
LSB_SUB_ERR_FILE	-e, -eo	string	Standard error file name
LSB_SUB_EXCLUSIVE	-x	boolean	Exclusive execution, specified by "Y"
LSB_SUB_EXTSCHEM_PARAM	-ext	string	External scheduler options
LSB_SUB_HOLD	-H	boolean	Hold job
LSB_SUB_HOST_SPEC	-c or -w	string	Host specifier, limits the CPU time or RUN time.
LSB_SUB_HOSTS	-m	string	List of requested execution host names
LSB_SUB_IN_FILE	-i, -io	string	Standard input file name
LSB_SUB_INTERACTIVE	-I	boolean	Interactive job, specified by "Y"
LSB_SUB_LOGIN_SHELL	-L	string	Login shell
LSB_SUB_JOB_DESCRIPTION	-Jd	string	Job description
LSB_SUB_JOB_NAME	-J	string	Job name
LSB_SUB_JOB_WARNING_ACTION	-wa	string	Job warning action
LSB_SUB_JOB_ACTION_WARNING_TIME	-wt	integer	Job warning time period
LSB_SUB_MAIL_USER	-u	string	Email address to which LSF sends job-related messages
LSB_SUB_MAX_NUM_PROCESSORS	-n	integer	Maximum number of processors requested
LSB_SUB_MODIFY	bmod	boolean	Indicates that bmod invoked esub, specified by "Y".
LSB_SUB_MODIFY_ONCE	bmod	boolean	Indicates that the job options specified at job submission have already been modified by bmod, and that bmod is invoking esub again, specified by "Y".
LSB_SUB_NOTIFY_BEGIN	-B	boolean	LSF sends an email notification when the job begins, specified by "Y".
LSB_SUB_NOTIFY_END	-N	boolean	LSF sends an email notification when the job ends, specified by "Y".
LSB_SUB_NUM_PROCESSORS	-n	integer	Minimum number of processors requested.

Option	bsub or bmod option	data type	Description
LSB_SUB_OTHER_FILES	bmod -f	integer	<p>Indicates the number of files to be transferred. The value is SUB_RESET if bmod is being used to reset the number of files to be transferred.</p> <p>The file path of the directory can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the director and file name.</p>
LSB_SUB_OTHER_FILES_number	bsub -f	integer	<p>The <i>number</i> indicates the particular file transfer value in the specified file transfer expression.</p> <p>For example, for bsub -f "a > b" -f "c < d", the following would be defined:</p> <p>LSB_SUB_OTHER_FILES=2</p> <p>LSB_SUB_OTHER_FILES_0="a > b"</p> <p>LSB_SUB_OTHER_FILES_1="c < d"</p>
LSB_SUB_OUT_FILE	-o, -oo	string	Standard output file name.
LSB_SUB_PRE_EXEC	-E	string	<p>Pre-execution command.</p> <p>The file path of the directory can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.</p>
LSB_SUB_PROJECT_NAME	-P	string	Project name.
LSB_SUB_PTY	-lp	boolean	An interactive job with PTY support, specified by "Y"
LSB_SUB_PTY_SHELL	-ls	boolean	An interactive job with PTY shell support, specified by "Y"
LSB_SUB_QUEUE	-q	string	Submission queue name
LSB_SUB_RERUNNABLE	-r	boolean	<p>"Y" specifies a rerunnable job</p> <p>"N" specifies a nonrerunnable job (specified with bsub -rn). The job is not rerunnable even it was submitted to a rerunable queue or application profile</p> <p>For bmod -rn, the value is SUB_RESET.</p>
LSB_SUB_RES_REQ	-R	string	Resource requirement string— <i>does not</i> support multiple resource requirement strings
LSB_SUB_RESTART	brestart	boolean	"Y" indicates to esub that the job options are associated with a restarted job.

Option	bsub or bmod option	data type	Description
LSB_SUB_RESTART_FORCE	brestart - f	boolean	"Y" indicates to esub that the job options are associated with a forced restarted job.
LSB_SUB_RLIMIT_CORE	-C	integer	Core file size limit
LSB_SUB_RLIMIT_CPU	-c	integer	CPU limit
LSB_SUB_RLIMIT_DATA	-D	integer	Data size limit For AIX, if the XPG_SUS_ENV=ON environment variable is set in the user's environment before the process is executed and a process attempts to set the limit lower than current usage, the operation fails with errno set to EINVAL. If the XPG_SUS_ENV environment variable is not set, the operation fails with errno set to EFAULT.
LSB_SUB_RLIMIT_FSIZE	-F	integer	File size limit
LSB_SUB_RLIMIT_PROCESS	-p	integer	Process limit
LSB_SUB_RLIMIT_RSS	-M	integer	Resident size limit
LSB_SUB_RLIMIT_RUN	-W	integer	Wall-clock run limit in seconds. (Note this is not in minutes, unlike the run limit specified by bsub -W)
LSB_SUB_RLIMIT_STACK	-S	integer	Stack size limit
LSB_SUB_RLIMIT_THREAD	-T	integer	Thread limit
LSB_SUB_TERM_TIME	-t	integer	Termination time, in seconds, since 00:00:00 GMT, Jan. 1, 1970
LSB_SUB_TIME_EVENT	-wt	string	Time event expression
LSB_SUB_USER_GROUP	-G	string	User group name
LSB_SUB_WINDOW_SIG	-s	boolean	Window signal number
LSB_SUB2_JOB_GROUP	-g	string	Submits a job to a job group
LSB_SUB2_LICENSE_PROJECT	-Lp	string	Platform License Scheduler project name
LSB_SUB2_IN_FILE_SPOOL	-is	string	Spooled input file name
LSB_SUB2_JOB_CMD_SPOOL	-Zs	string	Spooled job command file name
LSB_SUB2_JOB_PRIORITY	-sp	integer	Job priority For bmod -spn, the value is SUB_RESET.
LSB_SUB2_SLA	-sla	string	SLA scheduling options
LSB_SUB2_USE_RSV	-U	string	Advance reservation ID

Option	bsub or bmod option	data type	Description
LSB_SUB3_ABSOLUTE_PRIORITY	bmod -aps bmod -apsn	string	For bmod -aps, the value equal to the APS string given. For bmod -apsn, the value is SUB_RESET.
LSB_SUB3_AUTO_RESIZABLE	-ar	boolean	Job autoresizable attribute. LSB_SUB3_AUTO_RESIZABLE=Y if bsub -ar or bmod -ar is specified. LSB_SUB3_AUTO_RESIZABLE=SUB_RESET if bmod -arn is used.
LSB_SUB3_APP	-app	string	Application profile name For bmod -appn, the value is SUB_RESET.
LSB_SUB3_CWD	-cwd	string	Current working directory
LSB_SUB3_INIT_CHKPNT_PERIOD	-k init	integer	Initial checkpoint period
LSB_SUB_INTERACTIVE LSB_SUB3_INTERACTIVE_SSH	bsub -IS	boolean	The session of the interactive job is encrypted with SSH.
LSB_SUB_INTERACTIVE LSB_SUB_PTY LSB_SUB3_INTERACTIVE_SSH	bsub -ISp	boolean	If LSB_SUB_INTERACTIVE is specified by "Y", LSB_SUB_PTY is specified by "Y", and LSB_SUB3_INTERACTIVE_SSH is specified by "Y", the session of interactive job with PTY support is encrypted by SSH.
LSB_SUB_INTERACTIVE LSB_SUB_PTY LSB_SUB_PTY_SHELL LSB_SUB3_INTERACTIVE_SSH	bsub -ISs	boolean	If LSB_SUB_INTERACTIVE is specified by "Y", LSB_SUB_PTY is specified by "Y", LSB_SUB_PTY_SHELL is specified by "Y", and LSB_SUB3_INTERACTIVE_SSH is specified by "Y", the session of interactive job with PTY shell support is encrypted by SSH.
LSB_SUB3_JOB_REQUEUE	-Q	string	String format parameter containing the job requeue exit values For bmod -Qn, the value is SUB_RESET.
LSB_SUB3_MIG	-mig -mign	integer	Migration threshold
LSB_SUB3_POST_EXEC	-Ep	string	Run the specified post-execution command on the execution host after the job finishes. The file path of the directory can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.

Option	bsub or bmod option	data type	Description
LSB_SUB3_RESIZE_NOTIFY_CMD	-rnc	string	Job resize notification command. LSB_SUB3_RESIZE_NOTIFY_CMD=<cmd> if bsub -rnc or bmod -rnc is specified. LSB_SUB3_RESIZE_NOTIFY_CMD=SUB_RESET if bmod -rnc is used.
LSB_SUB3_RUNTIME_ESTIMATION	-We	integer	Runtime estimate in seconds. (Note this is not in minutes, unlike the runtime estimate specified by bsub -We)
LSB_SUB3_RUNTIME_ESTIMATION_ACC	-We+	integer	Runtime estimate that is the accumulated run time plus the runtime estimate
LSB_SUB3_RUNTIME_ESTIMATION_PERC	-Wep	integer	Runtime estimate in percentage of completion
LSB_SUB3_USER_SHELL_LIMITS	-ul	boolean	Pass user shell limits to execution host
LSB_SUB_INTERACTIVE LSB_SUB3_XJOB_SSH	bsub -IX	boolean	If both are set to "Y", the session between the X-client and X-server as well as the session between the execution host and submission host are encrypted with SSH.

LSB_SUB_MODIFY_FILE

Points to the file that `esub` uses to modify the `bsub` job option values stored in the `LSB_SUB_PARM_FILE`. You can change the job options by having your `esub` write the new values to the `LSB_SUB_MODIFY_FILE` in any order, using the same format shown for the `LSB_SUB_PARM_FILE`. The value `SUB_RESET`, integers, and boolean values do not require quotes. String parameters must be entered with quotes around each string, or space-separated series of strings.

When your `esub` runs at job submission, LSF checks the `LSB_SUB_MODIFY_FILE` and applies changes so that the job runs with the revised option values.

Restriction:

`LSB_SUB_ADDITIONAL` is the only option that an `esub` cannot change or add at job submission.

LSB_SUB_MODIFY_ENVFILE

Points to the file that `esub` uses to modify the user environment variables with which the job is submitted (not specified by `bsub` options). You can change these environment variables by having your `esub` write the values to the `LSB_SUB_MODIFY_ENVFILE` in any order, using the format `variable_name=value`, or `variable_name="string"`.

LSF uses the `LSB_SUB_MODIFY_ENVFILE` to change the environment variables on the submission host. When your `esub` runs at job submission, LSF checks the `LSB_SUB_MODIFY_ENVFILE` and applies changes so that the job is submitted with

the new environment variable values. LSF associates the new user environment with the job so that the job runs on the execution host with the new user environment.

LSB_SUB_ABORT_VALUE

Indicates to LSF that a job should be rejected. For example, if you want LSF to reject a job, your `esub` should contain the line

```
exit $LSB_SUB_ABORT_VALUE
```

Restriction:

When an `esub` exits with the `LSB_SUB_ABORT_VALUE`, `esub` must not write to `LSB_SUB_MODIFY_FILE` or to `LSB_SUB_MODIFY_ENVFILE`.

If multiple `esubs` are specified and one of the `esubs` exits with a value of `LSB_SUB_ABORT_VALUE`, LSF rejects the job without running the remaining `esubs` and returns a value of `LSB_SUB_ABORT_VALUE`.

LSB_INVOKE_CMD

Specifies the name of the LSF command that most recently invoked an external executable.

Environment variables used by `eexec`

When you write an `eexec`, you can use the following environment variables in addition to all user-environment or application-specific variables.

LS_EXEC_T

Indicates the stage or type of job execution. LSF sets `LS_EXEC_T` to:

- `START` at the beginning of job execution
- `END` at job completion
- `CHKPNT` at job checkpoint start

LS_JOBPID

Stores the process ID of the LSF process that invoked `eexec`. If `eexec` is intended to monitor job execution, `eexec` must spawn a child and then have the parent `eexec` process exit. The `eexec` child should periodically test that the job process is still alive using the `LS_JOBPID` variable.

Job submission and execution controls behavior

The following examples illustrate how customized esub and eexec executables can control job submission and execution.

Validating job submission parameters using esub

When a user submits a job using bsub-P, LSF accepts any project name entered by the user and associates that project name with the job. This example shows an esub that supports project-based accounting by enforcing the use of valid project names for jobs submitted by users who are eligible to charge to those projects. If a user submits a job to any project other than proj 1 or proj 2, or if the user name is not user 1 or user 2, LSF rejects the job based on the exit value of LSB_SUB_ABORT_VALUE.

```
#!/bin/sh
. $LSB_SUB_PARM_FILE
# Redirect stderr to stdout so echo can be used for error messages exec 1>&2
# Check valid projects
if [ $LSB_SUB_PROJECT_NAME != "proj 1" -o $LSB_SUB_PROJECT_NAME != "proj 2" ]; then
    echo "Incorrect project name specified"
    exit $LSB_SUB_ABORT_VALUE
fi
USER=`whoami`
if [ $LSB_SUB_PROJECT_NAME="proj 1" ]; then
# Only user1 and user2 can charge to proj 1
    if [ $USER != "user1" -a $USER != "user2" ]; then
        echo "You are not allowed to charge to this project"
        exit $LSB_SUB_ABORT_VALUE
    fi
fi
```

Changing job submission parameters using esub

The following example shows an esub that modifies job submission options and environment variables based on the user name that submits a job. This esub writes the changes to LSB_SUB_MODIFY_FILE for userA and to LSB_SUB_MODIFY_ENVFILE for userB. LSF rejects all jobs submitted by userC without writing to either file:

```
#!/bin/sh
. $LSB_SUB_PARM_FILE
# Redirect stderr to stdout so echo can be used for error messages exec 1>&2
USER=`whoami`
# Make sure userA is using the right queue queueA
if [ $USER="userA" -a $LSB_SUB_QUEUE != "queueA" ]; then
    echo "userA has submitted a job to an incorrect queue"
    echo "...submitting to queueA"
    echo 'LSB_SUB_QUEUE="queueA"' > $LSB_SUB_MODIFY_FILE
fi
# Make sure userB is using the right shell (/bin/sh)
if [ $USER="userB" -a $SHELL != "/bin/sh" ]; then
    echo "userB has submitted a job using $SHELL"
    echo "...using /bin/sh instead"
    echo 'SHELL="/bin/sh"' > $LSB_SUB_MODIFY_ENVFILE
fi
# Deny userC the ability to submit a job
if [ $USER="userC" ]; then
    echo "You are not permitted to submit a job."
    exit $LSB_SUB_ABORT_VALUE
fi
```

Monitoring the execution environment using eexec

This example shows how you can use an eexec to monitor job execution:

```
#!/bin/sh
# eexec
# Example script to monitor the number of jobs executing through RES.
# This script works in cooperation with an elim that counts the
```

```
# number of files in the TASKDIR directory. Each RES process on a host
# will have a file in the TASKDIR directory.
# Don't want to monitor lsbatch jobs.
if [ "$LSB_JOBID" != "" ] ; then
    exit 0
fi
TASKDIR="/tmp/RES_dir"
# directory containing all the task files
# for the host.
# you can change this to whatever
# directory you wish, just make sure anyone
# has read/write permissions.
# if TASKDIR does not exist create it
if [ "test -d $TASKDIR" != "0" ] ; then
    mkdir $TASKDIR > /dev/null 2>&1
fi
# Need to make sure LS_JOBPID, and USER are defined
# exit normally
if [ "test -z $LS_JOBPID" = "0" ] ; then
    exit 0
elif [ "test -z $USER" = "0" ] ; then
    exit 0
fi
taskFile="$TASKDIR/$LS_JOBPID.$USER"
# Fork grandchild to stay around for the duration of the task
touch $taskFile > /dev/null 2>&1
(
    (while ;
    do
        kill -0 $LS_JOBPID > /dev/null 2>&1
        if [ $? -eq 0 ] ; then
            sleep 10 # this is the poll interval
                    # increase it if you want but
                    # see the elim for its
                    # corresponding update interval
        else
            rm $taskFile > /dev/null 2>&1
            exit 0
        fi
    done) &
) &
wait
```

Passing data between esub and eexec

A combination of esub and eexec executables can be used to pass AFS/DCE tokens from the submission host to the execution host. LSF passes data from the standard output of esub to the standard input of eexec. A daemon wrapper script can be used to renew the tokens.

Configuration to modify job submission and execution controls

There are configuration parameters that modify various aspects of job submission and execution controls behavior by:

- Defining a mandatory esub that applies to all jobs in the cluster
- Specifying the eexec user account (UNIX only)

Configuration to define a mandatory esub

Configuration file	Parameter and syntax	Behavior
lsf.conf	LSB_ESUB_METHOD=" <i>esub_appl i cat i on [esub_appl i cat i on] ...</i> "	<ul style="list-style-type: none"> • The specified esub or esubs run for all jobs submitted to the cluster, in addition to any esub specified by the user in the command line • For example, to specify a mandatory esub named esub.fluent, define LSB_ESUB_METHOD=fluent

Configuration to specify the eexec user account

The eexec executable runs under the submission user account. You can modify this behavior for UNIX hosts by specifying a different user account.

Configuration file	Parameter and syntax	Behavior
lsf.sudoers	LSF_EEXEC_USER= <i>user_name</i>	<ul style="list-style-type: none"> • Changes the user account under which eexec runs

Job submission and execution controls commands

Commands for submission

Command	Description
<code>bsub -a esub_application [esub_application] ...</code>	<ul style="list-style-type: none"> Specifies one or more esub executables to run at job submission For example, to specify the esub named <code>esub.fluent</code>, use <code>bsub -a fluent</code> LSF runs any esub executables defined by <code>LSB_ESUB_METHOD</code>, followed by the executable named "esub" if it exists in <code>LSF_SERVERDIR</code>, followed by the esub executables specified by the <code>-a</code> option LSF runs <code>eexec</code> if an executable file with that name exists in <code>LSF_SERVERDIR</code>
<code>brestart</code>	<ul style="list-style-type: none"> Restarts a checkpointed job and runs the esub executables specified when the job was submitted LSF runs any esub executables defined by <code>LSB_ESUB_METHOD</code>, followed by the executable named "esub" if it exists in <code>LSF_SERVERDIR</code>, followed by the esub executables specified by the <code>-a</code> option LSF runs <code>eexec</code> if an executable file with that name exists in <code>LSF_SERVERDIR</code>
<code>l srun</code>	<ul style="list-style-type: none"> Submits an interactive task; LSF runs <code>eexec</code> if an <code>eexec</code> executable exists in <code>LSF_SERVERDIR</code> LSF runs <code>eexec</code> only at task startup (<code>LS_EXEC_T=START</code>)
<code>l sgrun</code>	<ul style="list-style-type: none"> Submits an interactive task to run on a set of hosts; LSF runs <code>eexec</code> if an <code>eexec</code> executable exists in <code>LSF_SERVERDIR</code> LSF runs <code>eexec</code> only at task startup (<code>LS_EXEC_T=START</code>)

Commands to monitor

Not applicable: There are no commands to monitor the behavior of this feature.

Commands to control

Command	Description
<code>bmod -a esub_application [esub_application] ...</code>	<ul style="list-style-type: none"> Resubmits a job and changes the esubs previously associated with the job LSF runs any esub executables defined by <code>LSB_ESUB_METHOD</code>, followed by the executable named "esub" if it exists in <code>LSF_SERVERDIR</code>, followed by the esub executables specified by the <code>-a</code> option of <code>bmod</code> LSF runs <code>eexec</code> if an executable file with that name exists in <code>LSF_SERVERDIR</code>

Command	Description
<code>bmod -an</code>	<ul style="list-style-type: none"> • Dissociates from a job all <code>esub</code> executables that were previously associated with the job • LSF runs any <code>esub</code> executables defined by <code>LSB_ESUB_METHOD</code>, followed by the executable named "esub" if it exists in <code>LSF_SERVERDIR</code> • LSF runs <code>eeexec</code> if an executable file with that name exists in <code>LSF_SERVERDIR</code>

Commands to display configuration

Command	Description
<code>badmi n showconf</code>	<ul style="list-style-type: none"> • Displays all configured parameters and their values set in <code>lsf.conf</code> or <code>ego.conf</code> that affect <code>mbatchd</code> and <code>sbatchd</code>. Use a text editor to view other parameters in the <code>lsf.conf</code> or <code>ego.conf</code> configuration files. • In a MultiCluster environment, displays the parameters of daemons on the local cluster.

Use a text editor to view the `lsf.sudoers` configuration file.

Interactive Jobs with bsub

About interactive jobs

It is sometimes desirable from a system management point of view to control all workload through a single centralized scheduler.

Running an interactive job through the LSF batch system allows you to take advantage of batch scheduling policies and host selection features for resource-intensive jobs. You can submit a job and the least loaded host is selected to run the job.

Since all interactive batch jobs are subject to LSF policies, you will have more control over your system. For example, you may dedicate two servers as interactive servers, and disable interactive access to all other servers by defining an interactive queue that only uses the two interactive servers.

Scheduling policies

Running an interactive batch job allows you to take advantage of batch scheduling policies and host selection features for resource-intensive jobs.

An interactive batch job is scheduled using the same policy as all other jobs in a queue. This means an interactive job can wait for a long time before it gets dispatched. If fast response time is required, interactive jobs should be submitted to high-priority queues with loose scheduling constraints.

Interactive queues

You can configure a queue to be interactive-only, batch-only, or both interactive and batch with the parameter `INTERACTIVE` in `lsb.queues`.

Interactive jobs with non-batch utilities

Non-batch utilities such as `lsrun`, `lsgroup`, etc., use LIM simple placement advice for host selection when running interactive tasks.

Submit interactive jobs

Use the `bsub -I` option to submit batch interactive jobs, and the `bsub -Is` and `-Ip` options to submit batch interactive jobs in pseudo-terminals.

Pseudo-terminals are not supported for Windows.

For more details, see the `bsub` command.

Find out which queues accept interactive jobs

Before you submit an interactive job, you need to find out which queues accept interactive jobs with the `bqueues -l` command.

If the output of this command contains the following, this is a batch-only queue. This queue does not accept interactive jobs:

```
SCHEDULING POLICIES: NO_INTERACTIVE
```

If the output contains the following, this is an interactive-only queue:

```
SCHEDULING POLICIES: ONLY_INTERACTIVE
```

If none of the above are defined or if `SCHEDULING POLICIES` is not in the output of `bqueues -l`, both interactive and batch jobs are accepted by the queue.

You configure interactive queues in the `lsb.queues` file.

Submit an interactive job

1. Use the `bsub -I` option to submit an interactive batch job.

For example:

`bsub -Is`

Submits a batch interactive job which displays the output of `ls` at the user's terminal.

```
% bsub -I -q interactive -n 4,10 lsmake
```

```
<<Waiting for dispatch ... >>
```

This example starts Platform Make on 4 to 10 processors and displays the output on the terminal.

A new job cannot be submitted until the interactive job is completed or terminated.

When an interactive job is submitted, a message is displayed while the job is awaiting scheduling. The `bsub` command stops display of output from the shell until the job completes, and no mail is sent to the user by default. A user can issue a `ctrl - c` at any time to terminate the job.

Interactive jobs cannot be checkpointed.

Interactive batch jobs cannot be rerunnable (`bsub -r`)

You can submit interactive batch jobs to rerunnable queues (`RERUNNABLE=y` in `lsb.queues`) or rerunnable application profiles (`RERUNNABLE=y` in `lsb.applications`).

Submit an interactive job by using a pseudo-terminal

Submission of interaction jobs using pseudo-terminal is not supported for Windows for either `lsrun` or `bsub` LSF commands.

Some applications such as `vi` require a pseudo-terminal in order to run correctly.

You can also submit an interactive job using a pseudo-terminal with shell mode support. This option should be specified for submitting interactive shells, or applications which redefine the CTRL-C and CTRL-Z keys (for example, `jobs`).

1. Submit a batch interactive job using a pseudo-terminal.

```
bsub -lp vi myfile
```

Submits a batch interactive job to edit `myfile`.

When you specify the `-lp` option, `bsub` submits a batch interactive job and creates a pseudo-terminal when the job starts.

2. Submit a batch interactive job and create a pseudo-terminal with shell mode support.

```
bsub -ls csh
```

Submits a batch interactive job that starts up `csh` as an interactive shell.

When you specify the `-ls` option, `bsub` submits a batch interactive job and creates a pseudo-terminal with shell mode support when the job starts.

Submit an interactive job and redirect streams to files

`bsub -i, -o, -e`

You can use the `-I` option together with the `-i`, `-o`, and `-e` options of `bsub` to selectively redirect streams to files. For more details, see the `bsub(1)` man page.

1. To save the standard error stream in the `job.err` file, while standard input and standard output come from the terminal:

```
% bsub -I -q interactive -e job.err ls make
```

Split stdout and stderr

If in your environment there is a wrapper around `bsub` and LSF commands so that end-users are unaware of LSF and LSF-specific options, you can redirect standard output and standard error of batch interactive jobs to a file with the `>` operator.

By default, both standard error messages and output messages for batch interactive jobs are written to `stdout` on the submission host.

1. To write both `stderr` and `stdout` to `mystdout`:

```
bsub -I myjob 2>mystderr 1>mystdout
```

2. To redirect both `stdout` and `stderr` to different files, set `LSF_INTERACTIVE_STDERR=y` in `lsf.conf` or as an environment variable.

For example, with `LSF_INTERACTIVE_STDERR` set:

```
bsub -I myjob 2>mystderr 1>mystdout
```

`stderr` is redirected to `mystderr`, and `stdout` to `mystdout`.

See the *Platform LSF Configuration Reference* for more details on `LSF_INTERACTIVE_STDERR`.

Submit an interactive job, redirect streams to files, and display streams

When using any of the interactive `bsub` options (for example: `-I`, `-Is`, `-ISs`) as well as the `-o` or `-e` options, you can also have your output displayed on the console by using the `-tty` option.

1. To run an interactive job, redirect the error stream to file, and display the stream to the console:

```
% bsub -l -q interactive -e job.err -tty lsmake
```

Performance tuning for interactive batch jobs

LSF is often used on systems that support both interactive and batch users. On one hand, users are often concerned that load sharing will overload their workstations and slow down their interactive tasks. On the other hand, some users want to dedicate some machines for critical batch jobs so that they have guaranteed resources. Even if all your workload is batch jobs, you still want to reduce resource contentions and operating system overhead to maximize the use of your resources.

Numerous parameters can be used to control your resource allocation and to avoid undesirable contention.

Types of load conditions

Since interferences are often reflected from the load indices, LSF responds to load changes to avoid or reduce contentions. LSF can take actions on jobs to reduce interference before or after jobs are started. These actions are triggered by different load conditions. Most of the conditions can be configured at both the queue level and at the host level. Conditions defined at the queue level apply to all hosts used by the queue, while conditions defined at the host level apply to all queues using the host.

Scheduling conditions

These conditions, if met, trigger the start of more jobs. The scheduling conditions are defined in terms of load thresholds or resource requirements.

At the queue level, scheduling conditions are configured as either resource requirements or scheduling load thresholds, as described in `l sb. queues`. At the host level, the scheduling conditions are defined as scheduling load thresholds, as described in `l sb. hosts`.

Suspending conditions

These conditions affect running jobs. When these conditions are met, a SUSPEND action is performed to a running job.

At the queue level, suspending conditions are defined as STOP_COND as described in `l sb. queues` or as suspending load threshold. At the host level, suspending conditions are defined as stop load threshold as described in `l sb. hosts`.

Resuming conditions

These conditions determine when a suspended job can be resumed. When these conditions are met, a RESUME action is performed on a suspended job.

At the queue level, resume conditions are defined as by RESUME_COND in `l sb. queues`, or by the `loadSched` thresholds for the queue if RESUME_COND is not defined.

Types of load indices

To effectively reduce interference between jobs, correct load indices should be used properly. Below are examples of a few frequently used parameters.

Paging rate (pg)

The paging rate (pg) load index relates strongly to the perceived interactive performance. If a host is paging applications to disk, the user interface feels very slow.

The paging rate is also a reflection of a shortage of physical memory. When an application is being paged in and out frequently, the system is spending a lot of time performing overhead, resulting in reduced performance.

The paging rate load index can be used as a threshold to either stop sending more jobs to the host, or to suspend an already running batch job to give priority to interactive users.

This parameter can be used in different configuration files to achieve different purposes. By defining paging rate threshold in `lsf.cluster.cluster_name`, the host will become busy from LIM's point of view; therefore, no more jobs will be advised by LIM to run on this host.

By including paging rate in queue or host scheduling conditions, jobs can be prevented from starting on machines with a heavy paging rate, or can be suspended or even killed if they are interfering with the interactive user on the console.

A job suspended due to pg threshold will not be resumed even if the resume conditions are met unless the machine is interactively idle for more than PG_SUSP_IT seconds.

Interactive idle time (it)

Strict control can be achieved using the idle time (it) index. This index measures the number of minutes since any interactive terminal activity. Interactive terminals include hard wired ttys, rlogin and lsllogin sessions, and X shell windows such as xterm. On some hosts, LIM also detects mouse and keyboard activity.

This index is typically used to prevent batch jobs from interfering with interactive activities. By defining the suspending condition in the queue as `it<1 && pg>50`, a job from this queue will be suspended if the machine is not interactively idle and the paging rate is higher than 50 pages per second. Furthermore, by defining the resuming condition as `it>5 && pg<10` in the queue, a suspended job from the queue will not resume unless it has been idle for at least five minutes and the paging rate is less than ten pages per second.

The it index is only non-zero if no interactive users are active. Setting the it threshold to five minutes allows a reasonable amount of think time for interactive users, while making the machine available for load sharing, if the users are logged in but absent.

For lower priority batch queues, it is appropriate to set an it suspending threshold of two minutes and scheduling threshold of ten minutes in the `lsb.queues` file. Jobs in these queues are suspended while the execution host is in use, and resume after the host has been idle for a longer period. For hosts where all batch jobs, no matter how important, should be suspended, set a per-host suspending threshold in the `lsb.hosts` file.

CPU run queue length (r15s, r1m, r15m)

Running more than one CPU-bound process on a machine (or more than one process per CPU for multiprocessors) can reduce the total throughput because of operating system overhead, as well as interfering with interactive users. Some tasks such as compiling can create more than one CPU-intensive task.

Queues should normally set CPU run queue scheduling thresholds below 1.0, so that hosts already running compute-bound jobs are left alone. LSF scales the run queue thresholds for multiprocessor hosts by using the effective run queue lengths, so multiprocessors automatically run one job per processor in this case.

For short to medium-length jobs, the r1m index should be used. For longer jobs, you might want to add an r15m threshold. An exception to this are high priority queues, where turnaround time is more important than total throughput. For high priority queues, an r1m scheduling threshold of 2.0 is appropriate.

CPU utilization (ut)

The `ut` parameter measures the amount of CPU time being used. When all the CPU time on a host is in use, there is little to gain from sending another job to that host unless the host is much more powerful than others on the network. A `ut` threshold of 90% prevents jobs from going to a host where the CPU does not have spare processing cycles.

If a host has very high `pg` but low `ut`, then it may be desirable to suspend some jobs to reduce the contention.

Some commands report `ut` percentage as a number from 0-100, some report it as a decimal number between 0-1. The configuration parameter in the `lsf.cluster.cluster_name` file, the configuration files, and the `bsub -R` resource requirement string take a fraction in the range from 0 to 1.

The command `bhist` shows the execution history of batch jobs, including the time spent waiting in queues or suspended because of system load.

The command `bjobs -p` shows why a job is pending.

Scheduling conditions and resource thresholds

Three parameters, `RES_REQ`, `STOP_COND` and `RESUME_COND`, can be specified in the definition of a queue. Scheduling conditions are a more general way for specifying job dispatching conditions at the queue level. These parameters take resource requirement strings as values which allows you to specify conditions in a more flexible manner than using the `loadSched` or `loadStop` thresholds.

Interactive batch job messaging

LSF can display messages to `stderr` or the Windows console when the following changes occur with interactive batch jobs:

- Job state
- Pending reason
- Suspend reason

Other job status changes, like switching the job's queue, are not displayed.

Limitations

Interactive batch job messaging is not supported in a MultiCluster environment.

Windows

Interactive batch job messaging is not fully supported on Windows. Only changes in the job state that occur before the job starts running are displayed. No messages are displayed after the job starts.

Configure interactive batch job messaging

Messaging for interactive batch jobs can be specified cluster-wide or in the user environment.

1. Enable interactive batch job messaging for all users in the cluster.

In `lsf.conf`:

- `LSB_INTERACT_MSG_ENH=Y`
- (Optional) `LSB_INTERACT_MSG_INTVAL`

`LSB_INTERACT_MSG_INTVAL` specifies the time interval, in seconds, in which LSF updates messages about any changes to the pending status of the job. The default interval is 60 seconds. `LSB_INTERACT_MSG_INTVAL` is ignored if `LSB_INTERACT_MSG_ENH` is not set.

OR

2. Enable messaging for interactive batch jobs.

Define `LSB_INTERACT_MSG_ENH` and `LSB_INTERACT_MSG_INTVAL` as environment variables.

Result: The user-level definition of `LSB_INTERACT_MSG_ENH` overrides the definition in `lsf.conf`.

Example messages

Job in pending state

The following example shows messages displayed when a job is in pending state:

```
bsub -ls -R "ls < 2" csh
Job <2812> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<< Job's resource requirements not satisfied: 2 hosts; >>
<< Load information unavailable: 1 host; >>
<< Just started a job recently: 1 host; >>
<< Load information unavailable: 1 host; >>
<< Job's resource requirements not satisfied: 1 host; >>
```

Job terminated by user

The following example shows messages displayed when a job in pending state is terminated by the user:

```
bsub -m hostA -b 13:00 -ls sh
Job <2015> is submitted to default queue <normal>.
Job will be scheduled after Fri Nov 19 13:00:00 2009
<<Waiting for dispatch ...>>
<< New job is waiting for scheduling >>
<< The job has a specified start time >>
bkill 2015
<< Job <2015> has been terminated by user or administrator >>
<<Terminated while pending>>
```

Job suspended then resumed

The following example shows messages displayed when a job is dispatched, suspended, and then resumed:

```
bsub -m hostA -ls sh
Job <2020> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>
<< New job is waiting for scheduling >>
<<Starting on hostA>>
bstop 2020
<< The job was suspended by user >>
bresume 2020
<< Waiting for re-scheduling after being resumed by user >>
```

Run X applications with bsub

You can start an X session on the least loaded host by submitting it as a batch job:

```
bsub xterm
```

An `xterm` is started on the least loaded host in the cluster.

When you run X applications using `lrun` or `bsub`, the environment variable `DISPLAY` is handled properly for you. It behaves as if you were running the X application on the local machine.

Configure SSH X11 forwarding for jobs

X11 forwarding must already be working outside LSF.

1. Install SSH and enable X11 forwarding for all hosts that will submit and run these jobs (UNIX hosts only).
2. (Optional) In `lsf.conf`, specify an SSH command for `LSB_SSH_XFORWARD_CMD`.

The command can include full PATH and options.

Write job scripts

You can build a job file one line at a time, or create it from another file, by running `bsub` without specifying a job to submit. When you do this, you start an interactive session in which `bsub` reads command lines from the standard input and submits them as a single batch job. You are prompted with `bsub>` for each line.

You can use the `bsub -Zs` command to spool a file.

For more details on `bsub` options, see the `bsub(1)` man page.

Write a job file one line at a time

UNIX example:

```
% bsub -q simulation
bsub> cd /work/data/myhomedir bsub> myjob arg1 arg2 .....
bsub> rm myjob.log
bsub> ^D
Job <1234> submitted to queue <simulation>.
```

In the above example, the 3 command lines run as a Bourne shell (`/bin/sh`) script. Only valid Bourne shell command lines are acceptable in this case.

Windows example:

```
C:\> bsub -q simulation
bsub> cd \\server\data\myhomedir
bsub> myjob arg1 arg2 .....
bsub> del myjob.log
bsub> ^Z
Job <1234> submitted to queue <simulation>.
```

In the above example, the 3 command lines run as a batch file (`.BAT`). Note that only valid Windows batch file command lines are acceptable in this case.

Specify job options in a file

In this example, options to run the job are specified in the `options_file`.

```
% bsub -q simulation < options_file
Job <1234> submitted to queue <simulation>.
```

On UNIX, the `options_file` must be a text file that contains Bourne shell command lines. It cannot be a binary executable file.

On Windows, the `options_file` must be a text file containing Windows batch file command lines.

Spool a job command file

Use `bsub -Zs` to spool a job command file to the directory specified by the `JOB_SPOOL_DIR` parameter in `lsb.params`, and use the spooled file as the command file for the job.

Use the `bmod -Zsn` command to modify or remove the command file after the job has been submitted. Removing or modifying the original input file does not affect the submitted job.

Redirect a script to bsub standard input

You can redirect a script to the standard input of the `bsub` command:

```
% bsub < myscript
```

```
Job <1234> submitted to queue <test>.
```

In this example, the `myscript` file contains job submission options as well as command lines to execute. When the `bsub` command reads a script from its standard input, it can be modified right after `bsub` returns for the next job submission.

When the script is specified on the `bsub` command line, the script is not spooled:

```
% bsub myscript
Job <1234> submitted to default queue <normal>.
```

In this case the command line `myscript` is spooled, instead of the contents of the `myscript` file. Later modifications to the `myscript` file can affect job behavior.

Specify embedded submission options

You can specify job submission options in scripts read from standard input by the `bsub` command using lines starting with `#BSUB`:

```
% bsub -q simulation bsub> #BSUB -q test
bsub> #BSUB -o outfile -R "mem>10"
bsub> myjob arg1 arg2
bsub> #BSUB -J simjob
bsub> ^D
Job <1234> submitted to queue <simulation>.
```

Note:

- Command-line options override embedded options. In this example, the job is submitted to the `simulation` queue rather than the `test` queue.
- Submission options can be specified anywhere in the standard input. In the above example, the `-J` option of `bsub` is specified after the command to be run.
- More than one option can be specified on one line, as shown in the example above.

Run a job under a particular shell

By default, LSF runs batch jobs using the Bourne (`/bin/sh`) shell. You can specify the shell under which a job is to run. This is done by specifying an interpreter in the first line of the script.

For example:

```
% bsub
bsub> #!/bin/csh -f
bsub> set coredump='ls |grep core'
bsub> if ( "$coredump" != "" ) then
bsub> mv core.core.'date | cut -d " " -f1'
bsub> endif
bsub> myjob
bsub> ^D
Job <1234> is submitted to default queue <normal>.
```

The `bsub` command must read the job script from standard input to set the execution shell. If you do not specify a shell in the script, the script is run using `/bin/sh`. If the first line of the script starts with a `#` not immediately followed by an exclamation mark (`!`), then `/bin/csh` is used to run the job.

For example:

```
% bsub
bsub> # This is a comment line. This tells the system to use /bin/csh to
bsub> # interpret the script.
```

```
bsub>  
bsub> setenv DAY 'date | cut -d" " -f1'  
bsub> myjob bsub> ^D  
Job <1234> is submitted to default queue <normal>.
```

If running jobs under a particular shell is required frequently, you can specify an alternate shell using a command-level job starter and run your jobs interactively.

Register utmp file entries for interactive batch jobs

LSF administrators can configure the cluster to track user and account information for interactive batch jobs submitted with `bsub -I p` or `bsub -I s`. User and account information is registered as entries in the UNIX `utmp` file, which holds information for commands such as `who`. Registering user information for interactive batch jobs in `utmp` allows more accurate job accounting.

Configuration and operation

To enable `utmp` file registration, the LSF administrator sets the `LSB_UTMP` parameter in `lsf.conf`.

When `LSB_UTMP` is defined, LSF registers the job by adding an entry to the `utmp` file on the execution host when the job starts. After the job finishes, LSF removes the entry for the job from the `utmp` file.

Limitations

- Registration of `utmp` file entries is supported on the following platforms:
 - SGI IRIX (6.4 and later)
 - Solaris (all versions)
 - HP-UX (all versions)
 - Linux (all versions)
- `utmp` file registration is not supported in a MultiCluster environment.
- Because interactive batch jobs submitted with `bsub -I` are not associated with a pseudo-terminal, `utmp` file registration is not supported for these jobs.

Interactive and Remote Tasks

You can run tasks interactively and remotely with non-batch utilities such as `l srun`, `l sgrun`, and `l sl o g i n`.

Run remote tasks

`l srun` is a non-batch utility to run tasks on a remote host. `l sgrun` is a non-batch utility to run the same task on many hosts, in sequence one after the other, or in parallel.

The default for `l srun` is to run the job on the host with the least CPU load (represented by the lowest normalized CPU run queue length) and the most available memory. Command-line arguments can be used to select other resource requirements or to specify the execution host.

To avoid typing in the `l srun` command every time you want to execute a remote job, you can also use a shell alias or script to run your job.

For a complete description of `l srun` and `l sgrun` options, see the `l srun(1)` and `l sgrun(1)` man pages.

Run a task on the best available host

1. Submit your task using `l srun`.

l srun mytask

LSF automatically selects a host of the same type as the local host, if one is available. By default the host with the lowest CPU and memory load is selected.

Run a task on a host with specific resources

If you want to run `mytask` on a host that meets specific resource requirements, you can specify the resource requirements using the `-R res_req` option of `l srun`.

1. **`l srun -R 'cserver && swp>100' mytask`**

In this example `mytask` must be run on a host that has the resource `cserver` and at least 100 MB of virtual memory available.

You can also configure LSF to store the resource requirements of specific tasks. If you configure LSF with the resource requirements of your task, you do not need to specify the `-R res_req` option of `l srun` on the command-line. If you do specify resource requirements on the command line, they override the configured resource requirements.

See the *LSF Configuration Reference* for information about configuring resource requirements in the `lsf.task` file.

Resource usage

Resource reservation is only available for batch jobs. If you run jobs using only LSF Base, LIM uses resource usage to determine the placement of jobs. Resource usage requests are used to temporarily increase the load so that a host is not overloaded. When LIM makes a placement advice, external load indices are not considered in the resource usage string. In this case, the syntax of the resource usage string is

```
res[=value]:res[=value]:...:res[=value]
```

The `res` is one of the resources whose value is returned by the `lsl load` command.

```
rusage[r1m=0.5:mem=20:swp=40]
```

The above example indicates that the task is expected to increase the 1-minute run queue length by 0.5, consume 20 MB of memory and 40 MB of swap space.

If no value is specified, the task is assumed to be intensive in using that resource. In this case no more than one task will be assigned to a host regardless of how many CPUs it has.

The default resource usage for a task is `r15s=1.0: r1m=1.0: r15m=1.0`. This indicates a CPU-intensive task which consumes few other resources.

Run a task on a specific host

1. If you want to run your task on a particular host, use the `lshrun -m` option:

```
lshrun -m hostD mytask
```

Run a task by using a pseudo-terminal

Submission of interaction jobs using pseudo-terminal is not supported for Windows for either `lshrun` or `bsub` LSF commands.

Some tasks, such as text editors, require special terminal handling. These tasks must be run using a pseudo-terminal so that special terminal handling can be used over the network.

1. The `-P` option of `lshrun` specifies that the job should be run using a pseudo-terminal:

```
lshrun -P vi
```

Run the same task on many hosts in sequence

The `lshrun` command allows you to run the same task on many hosts, one after the other, or in parallel.

1. For example, to merge the `/tmp/out` file on hosts `hostA`, `hostD`, and `hostB` into a single file named `gout`, enter:

```
lshrun -m "hostA hostD hostB" cat /tmp/out >> gout
```

Run parallel tasks

`lshrun -p`

The `-p` option tells `lshrun` that the task specified should be run in parallel. See `lshrun(1)` for more details.

1. To remove the `/tmp/core` file from all 3 hosts, enter:

```
lshrun -m "hostA hostD hostB" -p rm -r /tmp/core
```

Run tasks on hosts specified by a file

`lshrun -f host_file`

1. The `lshrun -f host_file` option reads the `host_file` file to get a list of hosts on which to run the task.

Interactive tasks

LSF supports transparent execution of tasks on all server hosts in the cluster. You can run your program on the best available host and interact with it just as if it were running directly on your workstation. Keyboard signals such as CTRL-Z and CTRL-C work as expected.

Interactive tasks communicate with the user in real time. Programs like `vi` use a text-based terminal interface. Computer Aided Design and desktop publishing applications usually use a graphic user interface (GUI).

This section outlines issues for running interactive tasks with the non-batch utilities `l srun`, `l sgrun`, etc. To run interactive tasks with these utilities, use the `-i` option.

For more details, see the `l srun(1)` and `l sgrun(1)` man pages.

Interactive tasks on remote hosts

Job controls

When you run an interactive task on a remote host, you can perform most of the job controls as if it were running locally. If your shell supports job control, you can suspend and resume the task and bring the task to background or foreground as if it were a local task.

For a complete description, see the `l srun(1)` man page.

Hide remote execution

You can also write one-line shell scripts or `csh` aliases to hide remote execution. For example:

```
#!/bin/sh
#Script to remotely execute mytask exec
l srun -m hostD mytask
```

or

```
alias mytask "l srun -m hostD mytask"
```

Interactive processing and scheduling policies

LSF lets you run interactive tasks on any computer on the network, using your own terminal or workstation. Interactive tasks run immediately and normally require some input through a text-based or graphical user interface. All the input and output is transparently sent between the local host and the job execution host.

Shared files and user IDs

When LSF runs a task on a remote host, the task uses standard UNIX system calls to access files and devices. The user must have an account on the remote host. All operations on the remote host are done with the user's access permissions.

Tasks that read and write files access the files on the remote host. For load sharing to be transparent, your files should be available on all hosts in the cluster using a file sharing mechanism such as NFS or AFS. When your files are available on all hosts in the cluster, you can run your tasks on any host without worrying about how your task will access files.

LSF can operate correctly in cases where these conditions are not met, but the results may not be what you expect. For example, the `/tmp` directory is usually private on each host. If you copy a file into `/tmp` on a remote host, you can only read that file on the same remote host.

LSF can also be used when files are not available on all hosts. LSF provides the `lsrcp` command to copy files across LSF hosts. You can use pipes to redirect the standard input and output of remote commands, or write scripts to copy the data files to the execution host.

Shell mode for remote execution

On UNIX, shell mode support is provided for running interactive applications through RES.

Not supported for Windows.

Shell mode support is required for running interactive shells or applications that redefine the CTRL-C and CTRL-Z keys (for example, `java`).

The `-S` option of `lshrun`, `ch` or `lshgrun` creates the remote task with shell mode support. The default is not to enable shell mode support.

Run windows

Some run windows are only applicable to batch jobs. Interactive jobs scheduled by LIM are controlled by another set of run windows.

Redirect streams to files

By default, both standard error messages and standard output messages of interactive tasks are written to `stdout` on the submission host.

To separate `stdout` and `stderr` and redirect to separate files, set `LSF_INTERACTIVE_STDERR=y` in `lsf.conf` or as an environment variable.

1. To redirect both `stdout` and `stderr` to different files with the parameter set:

```
lshrun mytask 2>mystderr 1>mystdout
```

The result of the above example is for `stderr` to be redirected to `mystderr`, and `stdout` to `mystdout`. Without `LSF_INTERACTIVE_STDERR` set, both `stderr` and `stdout` will be redirected to `mystdout`.

See the *LSF Configuration Reference* for more details on `LSF_INTERACTIVE_STDERR`.

Load sharing interactive sessions

There are different ways to use LSF to start an interactive session on the best available host.

Log on to the least loaded host

1. To log on to the least loaded host, use the `lsl login` command.

When you use `lsl login`, LSF automatically chooses the best host and does an `rlogin` to that host.

With no argument, `lsl login` picks a host that is lightly loaded in CPU, has few login sessions, and whose binary is compatible with the current host.

Log on to a host with specific resources

1. If you want to log on a host that meets specific resource requirements, use the `lsl login -R res_req` option.

```
lslogin -R "solaris order[ls:cpu]"
```

This command opens a remote login to a host that has the `sunos` resource, few other users logged in, and a low CPU load level. This is equivalent to using `lsplace` to find the best host and then using `rlogin` to log in to that host:

```
rlogin 'lsplace -R "sunos order[ls:cpu]"
```

Load sharing X applications

Start an xterm

1. If you are using the X Window System, you can start an `xterm` that opens a shell session on the least loaded host by entering:

```
lshrun sh -c xterm &
```

The `&` in this command line is important as it frees resources on the host once `xterm` is running, by running the X terminal in the background.

In this example, no processes are left running on the local host. The `lshrun` command exits as soon as `xterm` starts, and the `xterm` on the remote host connects directly to the X server on the local host.

xterm on a PC

Each X application makes a separate network connection to the X display on the user's desktop. The application generally gets the information about the display from the `DISPLAY` environment variable.

X-based systems such as `eXceed` start applications by making a remote shell connection to the UNIX server, setting the `DISPLAY` environment variable, and then invoking the X application. Once the application starts, it makes its own connection to the display and the initial remote shell is no longer needed.

This approach can be extended to allow load sharing of remote applications. The client software running on the X display host makes a remote shell connection to any server host in the LSF cluster. Instead of running the X application directly, the client invokes a script that uses LSF to select the best available host and starts the application on that host. Because the application then makes a direct connection to the display, all of the intermediate connections can be closed. The client software on the display host must select a host in the cluster to start the connection. You can choose an arbitrary host for this; once LSF selects the best host and starts the X application there, the initial host is no longer involved. There is no ongoing load on the initial host.

Set up Exceed to log on the least loaded host

If you are using a PC as a desktop machine and are running an X Window server on your PC, then you can start an X session on the least loaded host.

The following steps assume you are using `Exceed` from Hummingbird Communications. This procedure can be used to load share any X-based application.

You can customize host selection by changing the resource requirements specified with `-R "..."`. For example, a user could have several icons in the `xterm` program group: one called `Best`, another called `Best_Sun`, another `Best_SGI`.

1. Click the `Xstart` icon in the `Exceed` program group.
2. Choose `REXEC (TCP/IP, ...)` as start method, program type is X window.
3. Set the host to be any server host in your LSF cluster:

```
lshrun -R "type==any order[cpu:mem:login]" xterm -sb -ls -display your_PC:0.0
```

4. Set description to be `Best`.
5. Click `Install` in the `Xstart` window.

This installs `Best` as an icon in the program group you chose (for example, `xterm`).

The user can now log on to the best host by clicking Best in the Xterm program group.

Start an xterm in Exceed

To start an xterm:

1. Double-click Best.

An xterm starts on the least loaded host in the cluster and is displayed on your screen.

Examples

Run any application on the least loaded host

To run appY on the best machine licensed for it, you could set the command line in Exceed to be the following and set the description to appY:

```
lsrun -R "type==any && appY order[mem:cpu]" sh -c "appY -display your_PC:0.0 &"
```

You must make sure that all the UNIX servers licensed for appY are configured with the resource "appY". In this example, appY requires a lot of memory when there are embedded graphics, so we make "mem" the most important consideration in selecting the best host among the eligible servers.

Start an X session on the least loaded host in any X desktop environment

The above approach also applies to other X desktop environments. In general, if you want to start an X session on the best host, run the following on an LSF host:

```
lsrun -R "resource_requirement" my_Xapp -display your_PC:0.0
```

where

resource_requirement is your resource requirement string

Script for automatically specifying resource requirements

The above examples require the specification of resource requirement strings by users. You may want to centralize this such that all users use the same resource specifications.

You can create a central script (for example `lslaunch`) and place it in the `/lsf/bin` directory. For example:

```
#!/bin/sh
lsrun -R "order[cpu:mem:login]" $@
exit $?
```

Which would simplify the command string to:

```
lslaunch xterm -sb -ls -display your_PC:0.0
```

Taking this one step further, you could create a script named `lsxterm`:

```
#!/bin/sh
lsrun -R "order[cpu:mem:login]" xterm -sb -ls $@
exit $?
```

Which would simplify the command string to:

```
lsxterm -display your_PC:0.0
```


VIII

Appendices



Submitting Jobs Using JSDL

The Job Submission Description Language (JSDL) provides a convenient format for describing job requirements. You can save a set of job requirements in a JSDL XML file, and then reuse that file as needed to submit jobs to LSF.

For detailed information about JSDL, see the "Job Submission Description Language (JSDL) Specification" at <http://www.gridforum.org/documents/GFD.56.pdf>.

Use JSDL files with LSF

LSF complies with the JSDL specification by supporting most valid JSDL elements and POSIX extensions. The LSF extension schema allows you to use LSF features not included in the JSDL standard schema.

The following sections describe how LSF supports the use of JSDL files for job submission.

Where to find the JSDL schema files

The JSDL schema (`j s d l . x s d`), the POSIX extension (`j s d l - p o s i x . x s d`), and the LSF extension (`j s d l - l s f . x s d`) are located in the `LSF_LI B D I R` directory.

Supported JSDL and POSIX extension elements

The following table maps the supported JSDL standard and POSIX extension elements to LSF submission options.

Note:

For information about how to specify JSDL element types such as range values, see the "Job Submission Description Language (JSDL) Specification" at <http://www.gridforum.org/documents/GFD.56.pdf>.

Table 4: Supported JSDL and POSIX extension elements

Element	bsub Option	Description	Example
Job Structure Elements			

Element	bsub Option	Description	Example
JobDefinition	N/A	Root element of the JSDL document. Contains the mandatory child element JobDescription.	<code><JobDefinition> <JobDescription> ... </JobDescription> </JobDefinition></code>
JobDescription	-P	High-level container element that holds more specific description elements.	
Job Identity Elements			
JobName	-J	String used to name the job.	<code><j s d l : J o b N a m e>myj ob</j s d l : J o b N a m e></code>
JobProject	-P	String that specifies the project to which the job belongs.	<code><j s d l : J o b P r o j e c t>myproj ect </j s d l : J o b P r o j e c t></code>
Application Elements			
Application	N/A	High-level container element that holds more specific application definition elements.	
ApplicationName	-app	String that defines the name of an application profile defined in lsb. appl i cat i ons .	<code><j s d l : A p p l i c a t i o n> <j s d l : A p p l i c a t i o n N a m e>A p p l i c a t i o n X </j s d l : A p p l i c a t i o n N a m e> </j s d l : A p p l i c a t i o n></code>
ApplicationVersion	-app	String that defines the version of the application defined in lsb. appl i cat i ons .	<code><j s d l : A p p l i c a t i o n> <j s d l : A p p l i c a t i o n N a m e> A p p l i c a t i o n X</j s d l : A p p l i c a t i o n N a m e> <j s d l : A p p l i c a t i o n V e r s i o n>5. 5 </j s d l : A p p l i c a t i o n V e r s i o n> ... </j s d l : A p p l i c a t i o n></code>
Resource Elements			
CandidateHosts	-m	Complex type element that specifies the set of named hosts that can be selected to run the job.	<code><j s d l : C a n d i d a t e H o s t s> <j s d l : H o s t N a m e>host 1</j s d l : H o s t N a m e><j s d l : H o s t N a m e>host 2</j s d l : H o s t N a m e> </j s d l : C a n d i d a t e H o s t s></code>
HostName	-m	Contains a single name of a host or host group. See the previous example (CandidateHosts).	

Element	bsub Option	Description	Example
ExclusiveExecution	-x	Boolean that designates whether the job must have exclusive access to the resources it uses.	<pre><j s d l : Excl u s i v e E x e c u t i o n>true </ j s d l : Excl u s i v e E x e c u t i o n></pre>
OperatingSystemName	-R	A token type that contains the operating system name. LSF uses the external resource "osname."	<pre><j s d l : Operat i n g S y s t e m N a m e>L I N U X </ j s d l : Operat i n g S y s t e m N a m e></pre>
OperatingSystemVersion	-R	A token type that contains the operating system version. LSF uses the external resource "osver."	<pre><j s d l : Operat i n g S y s t e m V e r s i o n>5. 7 </ j s d l : Operat i n g S y s t e m V e r s i o n></pre>
CPUArchitectureName	-R	Token that specifies the CPU architecture required by the job in the execution environment. LSF uses the external resource "cpuarch."	<pre><j s d l : C P U A r c h i t e c t u r e N a m e>s p a r c </ j s d l : C P U A r c h i t e c t u r e N a m e></pre>
IndividualCPUSpeed	-R	Range value that specifies the speed of each CPU required by the job in the execution environment, in Hertz (Hz). LSF uses the external resource "cpuspeed."	<pre><j s d l : I n d i v i d u a l C P U S p e e d> <j s d l : LowerBoundedRange>1073741824. 0 </ j s d l : LowerBoundedRange> </ j s d l : I n d i v i d u a l C P U S p e e d></pre>
IndividualCPUCount	-n	Range value that specifies the number of CPUs for each resource.	<pre><j s d l : I n d i v i d u a l C P U C o u n t> <j s d l : exact>2. 0</j s d l : exact> </ j s d l : I n d i v i d u a l C P U C o u n t></pre>
IndividualPhysicalMemory	-R	Range value that specifies the amount of physical memory required on each resource, in bytes.	<pre><j s d l : I n d i v i d u a l P h y s i c a l M e m o r y> <j s d l : LowerBoundedRange>1073741824. 0 </ j s d l : LowerBoundedRange> </ j s d l : I n d i v i d u a l P h y s i c a l M e m o r y></pre>
IndividualVirtualMemory	-R	Range value that specifies the amount of virtual memory required for each resource, in bytes.	<pre><j s d l : I n d i v i d u a l V i r t u a l M e m o r y> <j s d l : LowerBoundedRange>1073741824. 0 </ j s d l : LowerBoundedRange> </ j s d l : I n d i v i d u a l V i r t u a l M e m o r y></pre>

Element	bsub Option	Description	Example
IndividualNetworkBandwidth	-R	Range value that specifies the bandwidth requirements of each resource, in bits per second (bps). LSF uses the external resource "bandwidth."	<pre><j s d l : I n d i v i d u a l N e t w o r k B a n d w i d t h> <j s d l : L o w e r B o u n d e d R a n g e> 104857600. 0 </ j s d l : L o w e r B o u n d e d R a n g e> </ j s d l : I n d i v i d u a l N e t w o r k B a n d w i d t h></pre>
TotalCPUCount	-n	Range value that specifies the total number of CPUs required for the job.	<pre><j s d l : T o t a l C P U C o u n t><j s d l : e x a c t>2. 0 </ j s d l : e x a c t></j s d l : T o t a l C P U C o u n t></pre>
TotalPhysicalMemory	-R	Range value that specifies the required amount of physical memory for all resources for the job, in bytes.	<pre><j s d l : T o t a l P h y s i c a l M e m o r y> <j s d l : L o w e r B o u n d e d R a n g e> 10737418240. 0 </j s d l : L o w e r B o u n d e d R a n g e> </ j s d l : T o t a l P h y s i c a l M e m o r y></pre>
TotalVirtualMemory	-R	Range value that specifies the required amount of virtual memory for the job, in bytes.	<pre><j s d l : T o t a l V i r t u a l M e m o r y> <j s d l : L o w e r B o u n d e d R a n g e> 1073741824. 0 </ j s d l : L o w e r B o u n d e d R a n g e> </ j s d l : T o t a l V i r t u a l M e m o r y></pre>
TotalResourceCount	-n	Range value that specifies the total number of resources required by the job.	<pre><j s d l : R e s o u r c e s> . . . <j s d l : T o t a l R e s o u r c e C o u n t> <j s d l : e x a c t>5. 0</j s d l : e x a c t> </ j s d l : T o t a l R e s o u r c e C o u n t></pre>
Data Staging Elements			
FileName	-f	String that specifies the local name of the file or directory on the execution host. For a directory, you must specify the relative path.	<pre><j s d l : D a t a S t a g i n g><j s d l : F i l e N a m e> j o b 1 / i n p u t / c o n t r o l . t x t </ j s d l : F i l e N a m e> . . . </j s d l : D a t a S t a g i n g></pre>
CreationFlag	-f	Specifies whether the file created on the local execution system overwrites or append to an existing file.	<pre><j s d l : D a t a S t a g i n g> <j s d l : C r e a t i o n F l a g> o v e r w r i t e </ j s d l : C r e a t i o n F l a g> . . . </ j s d l : D a t a S t a g i n g></pre>

Element	bsub Option	Description	Example
Source	N/A	Contains the location of the file or directory on the remote system. In LSF, the file location is specified by the URI element. The file is staged in before the job is executed. See the example for the Target element.	
URI	-f	Specifies the location used to stage in (Source) or stage out (Target) a file. For use with LSF, the URI must be a file path only, without a protocol.	
Target	N/A	Contains the location of the file or directory on the remote system. In LSF, the file location is specified by the URI element. The file is staged out after the job is executed.	<pre><j s d l : DataStagi ng><j s d l : Source> <j s d l : URI> //input/myj obs/control . txt </ j s d l : URI></j s d l : Source> <j s d l : Target><j s d l : URI> //output/ myj obs/control . txt </j s d l : URI></ j s d l : Target> ... </j s d l : DataStagi ng></pre>
POSIX Extension Elements			
Executable	N/A	String that specifies the command to execute.	<pre><j s d l - posi x: Execut abl e>myscri pt </j s d l - posi x: Execut abl e></pre>
Argument	N/A	Constrained normalized string that specifies an argument for the application or command.	<pre><j s d l - posi x: Argument>10 </j s d l - posi x: Argument></pre>
Input	-i	String that specifies the Standard Input for the command.	<pre>... <j s d l - posi x: Input>i nput. txt </j s d l - posi x: Input>...</pre>
Output	-o	String that specifies the Standard Output for the command.	<pre>... <j s d l - posi x: Out put>out put. txt </ j s d l - posi x: Out put>...</pre>
Error	-e	String that specifies the Standard Error for the command.	<pre>... <j s d l - posi x: Error>error. txt </j s d l - posi x: Error>...</pre>

Element	bsub Option	Description	Example
WorkingDirectory	N/A	String that specifies the starting directory required for job execution. If no directory is specified, LSF sets the starting directory on the execution host to the current working directory on the submission host. If the current working directory is not accessible on the execution host, LSF runs the job in the /tmp directory on the execution host.	<pre>... <j sdl - posi x: Worki ngDi rectory> / home</j sdl - posi x: Worki ngDi rectory> ...</pre>
Environment	N/A	Specifies the name and value of an environment variable defined for the job in the execution environment. LSF maps the JSDL element definitions to the matching LSF environment variables.	<pre><j sdl - posi x: Envi ronment name="SHELL"> / bi n/bash</j sdl - posi x: Envi ronment></pre>
WallTimeLimit	-W	Positive integer that specifies the soft limit on the duration of the application's execution, in seconds.	<pre><j sdl - posi x: Wal l Ti meLi mi t>60 </j sdl - posi x: Wal l Ti meLi mi t></pre>
FileSizeLimit	-F	Positive integer that specifies the maximum size of any file associated with the job, in bytes.	<pre><j sdl - posi x: Fi l eSi zeLi mi t>1073741824 </ j sdl - posi x: Fi l eSi zeLi mi t></pre>
CoreDumpLimit	-C	Positive integer that specifies the maximum size of core dumps a job may create, in bytes.	<pre><j sdl - posi x: CoreDumpLi mi t>0 </j sdl - posi x: CoreDumpLi mi t></pre>
DataSegmentLimit	-D	Positive integer that specifies the maximum data segment size, in bytes.	<pre><j sdl - posi x: DataSegmentLi mi t>32768 </ j sdl - posi x: DataSegmentLi mi t></pre>

Element	bsub Option	Description	Example
MemoryLimit	-M	Positive integer that specifies the maximum amount of physical memory that the job can use during execution, in bytes.	<code><j sdl - posi x: MemoryLi mi t>67108864 </j sdl - posi x: MemoryLi mi t></code>
StackSizeLimit	-S	Positive integer that specifies the maximum size of the execution stack for the job, in bytes.	<code><j sdl - posi x: StackSi zeLi mi t>1048576 </j sdl - posi x: StackSi zeLi mi t></code>
CPUTimeLimit	-c	Positive integer that specifies the number of CPU time seconds a job can consume before a SIGXCPU signal is sent to the job.	<code><j sdl - posi x: CPUTi meLi mi t>30 </j sdl - posi x: CPUTi meLi mi t></code>
ProcessCountLimit	-p	Positive integer that specifies the maximum number of processes the job can spawn.	<code><j sdl - posi x: ProcessCountLi mi t>8 </j sdl - posi x: ProcessCountLi mi t></code>
VirtualMemoryLimit	-v	Positive integer that specifies the maximum amount of virtual memory the job can allocate, in bytes.	<code><j sdl - posi x: Vi rtual MemoryLi mi t>134217728 </j sdl - posi x: Vi rtual MemoryLi mi t></code>
ThreadCountLimit	-T	Positive integer that specifies the number of threads the job can create.	<code><j sdl - posi x: ThreadCountLi mi t>8 </j sdl - posi x: Vi rtual MemoryLi mi t></code>

LSF extension elements

To use all available LSF features, add the elements described in the following table to your JSDL file.

Table 5: LSF extension elements

Element	bsub Option	Description
SchedulerParams	N/A	Complex type element that specifies various scheduling parameters (starting with Queue and ending with JobGroup). <pre> <j s d l - l s f : S c h e d u l e r P a r a m s > <j s d l - l s f : Q u e u e > n o r m a l </j s d l - l s f : Q u e u e > <j s d l - l s f : R e s o u r c e R e q u i r e m e n t s > " s e l e c t [s w p > 1 5 && h p u x] o r d e r [u t] " </j s d l - l s f : R e s o u r c e R e q u i r e m e n t s > <j s d l - l s f : S t a r t > 1 2 : 0 6 : 0 9 : 5 5 </j s d l - l s f : S t a r t > <j s d l - l s f : D e a d l i n e > 8 : 2 2 : 1 5 : 5 0 </j s d l - l s f : D e a d l i n e > <j s d l - l s f : R e s e r v a t i o n I D > " u s e r 1 # 0 " </j s d l - l s f : R e s e r v a t i o n I D > <j s d l - l s f : D e p e n d e n c i e s > ' d o n e m y j o b 1 ' </j s d l - l s f : D e p e n d e n c i e s > <j s d l - l s f : R e r u n n a b l e > t r u e </j s d l - l s f : R e r u n n a b l e > <j s d l - l s f : U s e r P r i o r i t y > 3 </j s d l - l s f : U s e r P r i o r i t y > <j s d l - l s f : S e r v i c e C l a s s > p l a t i n u m </j s d l - l s f : S e r v i c e C l a s s > <j s d l - l s f : G r o u p > s y s a d m i n </j s d l - l s f : G r o u p > <j s d l - l s f : E x t e r n a l S c h e d u l e r > p s e t </j s d l - l s f : E x t e r n a l S c h e d u l e r > <j s d l - l s f : H o l d > t r u e </j s d l - l s f : H o l d > <j s d l - l s f : J o b G r o u p > / r i s k _ g r o u p / p o r t f o l i o 1 / c u r r e n t </j s d l - l s f : J o b G r o u p > </j s d l - l s f : S c h e d u l e r P a r a m s > </pre>
Queue	-q	String that specifies the queue in which the job runs.
ResourceRequirements	-R	String that specifies one or more resource requirements of the job. Multiple rusage sections are not supported.
Start	-b	String that specifies the earliest time that the job can start.
Deadline	-t	String that specifies the job termination deadline.
ReservationID	-U	String that specifies the reservation ID returned when you use brsvadd to add a reservation.
Dependencies	-w	String that specifies a dependency expression. LSF does not run your job unless the dependency expression evaluates to TRUE.
Rerunnable	-r	Boolean value that specifies whether to reschedule a job on another host if the execution host becomes unavailable while the job is running.
UserPriority	-sp	Positive integer that specifies the user-assigned job priority. This allows users to order their own jobs within a queue.
ServiceClass	-sla	String that specifies the service class where the job is to run.
Group	-G	String that associates the job with the specified group for fairshare scheduling.
ExternalScheduler	-ext [sched]	String used to set application-specific external scheduling options for the job.
Hold	-H	Boolean value that tells LSF to hold the job in the PSUSP state when the job is submitted. The job is not scheduled until you tell the system to resume the job.
JobGroup	-g	String that specifies the job group to which the job is submitted.

Element	bsub Option	Description
NotificationParams	N/A	Complex type element that tells LSF when and where to send notification email for a job. See the following example: <pre><j s d l - l s f : N o t i f i c a t i o n P a r a m s > <j s d l - l s f : N o t i f y A t S t a r t > t r u e </j s d l - l s f : N o t i f y A t S t a r t > <j s d l - l s f : N o t i f y A t F i n i s h > t r u e </j s d l - l s f : N o t i f y A t F i n i s h > <j s d l - l s f : N o t i f i c a t i o n E m a i l > - u u s e r 1 0 </j s d l - l s f : N o t i f i c a t i o n E m a i l > </j s d l - l s f : N o t i f i c a t i o n P a r a m s ></pre>
NotifyAtStart	-B	Boolean value that tells LSF to notify the user who submitted the job that the job has started.
NotifyAtFinish	-N	Boolean value that tells LSF to notify the user who submitted the job that the job has finished.
NotificationEmail	-u	String that specifies the user who receives notification emails.
RuntimeParams	N/A	Complex type element that contains values for LSF runtime parameters. <pre><j s d l - l s f : R u n t i m e P a r a m s > <j s d l - l s f : I n t e r a c t i v e > I </j s d l - l s f : I n t e r a c t i v e > <j s d l - l s f : B l o c k > t r u e </j s d l - l s f : B l o c k > <j s d l - l s f : P r e E x e c > m y s c r i p t </j s d l - l s f : P r e E x e c > <j s d l - l s f : C h e c k p o i n t > m y j o b s / c h e c k p o i n t d i r </j s d l - l s f : C h e c k p o i n t > <j s d l - l s f : L o g i n S h e l l > / c s h </j s d l - l s f : L o g i n S h e l l > <j s d l - l s f : S i g n a l J o b > 1 8 </j s d l - l s f : S i g n a l J o b > <j s d l - l s f : W a r n i n g A c t i o n > ' U R G ' </j s d l - l s f : W a r n i n g A c t i o n > <j s d l - l s f : W a r n i n g T i m e > ' 2 ' </j s d l - l s f : W a r n i n g T i m e > <j s d l - l s f : S p o o l C o m m a n d > t r u e </j s d l - l s f : S p o o l C o m m a n d > <j s d l - l s f : C h e c k p o i n t > </j s d l - l s f : R u n t i m e P a r a m s ></pre>
Interactive	-I[s p]	String that specifies an interactive job with a defined pseudo-terminal mode.
Block	-K	Boolean value that tells LSF to complete the submitted job before allowing the user to submit another job.
PreExec	-E	String that specifies a pre-exec command to run on the batch job's execution host before actually running the job. For a parallel job, the pre-exec command runs on the first host selected for the parallel job.
Checkpoint	-k	String that makes a job checkpointable and specifies the checkpoint directory.
LoginShell	-L	For UNIX jobs, string that tells LSF to initialize the execution environment using the specified login shell.
SignalJob	-s	String that specifies the signal to send when a queue-level run window closes. Use this to override the default signal that suspends jobs running in the queue.
WarningAction	-wa	String that specifies the job action prior to the job control action. Requires that you also specify the job action warning time.
WarningTime	-wt	Positive integer that specifies the amount of time prior to a job control action that the job warning action should occur.
SpoolCommand	-is	Boolean value that spools a job command file to the directory specified by JOB_SPOOL_DIR, and uses the spooled file as the command file for the job.

Unsupported JSDL and POSIX extension elements

The current version of LSF does not support the following elements:

Job structure elements

- Description

Job identity elements

- JobAnnotation

Resource elements

- FileSystem
- MountPoint
- MountSource
- DiskSpace
- FileSystemType
- OperatingSystemType
- IndividualCPUTime
- IndividualDiskSpace
- TotalCPUTime
- TotalDiskSpace

Data staging elements

- FileSystemName
- DeleteOnTermination

POSIX extension elements

- LockedMemoryLimit
- OpenDescriptorsLimit
- PipeSizeLimit
- UserName
- GroupName

Submit a job using a JSDL file

1. To submit a job using a JSDL file, use one of the following bsub command options:
 - a) To submit a job that uses elements included in the LSF extension, use the `-jsdl` option.
 - b) To submit a job that uses only standard JSDL elements and POSIX extensions, use the `-jsdl_strict` option. Error messages indicate invalid elements, including:
 - Elements that are not part of the JSDL specification
 - Valid JSDL elements that are not supported in this version of LSF
 - Elements that are not part of the JSDL standard and POSIX extension schema

If you specify duplicate or conflicting job submission parameters, LSF resolves the conflict by applying the following rules:

- The parameters specified in the command line override all other parameters.
- A job script or user input for an interactive job overrides parameters specified in the JSDL file.

Collect resource values using elim.jsdl

To support the use of JSDL files at job submission, LSF collects the following load indices:

Attribute name	Attribute type	Resource name
OperatingSystemName	string	osname
OperatingSystemVersion	string	osver
CPUArchitectureName	string	cpuarch
IndividualCPUSpeed	int64	cpuspeed
IndividualNetworkBandwidth	int64	bandwidth(This is the maximum bandwidth).

The file `elim.jsdl` is automatically configured to collect these resources, but you must enable its use by modifying the files `lsf.cluster.cluster_name` and `lsf.shared`.

Enable JSDL resource collection

1. In the file `lsf.cluster.cluster_name`, locate the ResourcesMap section.
2. In the file `lsf.shared`, locate the Resource section.
3. Uncomment the lines for the following resources in both files:
 - `osname`
 - `osver`
 - `cpuarch`
 - `cpuspeed`
 - `bandwidth`
4. To propagate the changes through the LSF system, run the following commands.
 - a) `lsadmin reconfig`
 - b) `badmin mbdrestart`

You have now configured LSF to use the `elim.jsdl` file to collect JSDL resources.

B

Using Istch

This chapter describes `lstcsh`, an extended version of the `tcsh` command interpreter. The `lstcsh` interpreter provides transparent load sharing of user jobs.

This chapter is not a general description of the `tcsh` shell. Only load sharing features are described in detail.

Interactive tasks, including `lstcsh`, are not supported on Windows.

About Istcsh

The `lstcsh` shell is a load-sharing version of the `tcsh` command interpreter. It is compatible with `csh` and supports many useful extensions. `csh` and `tcsh` users can use `lstcsh` to send jobs to other hosts in the cluster without needing to learn any new commands. You can run `lstcsh` from the command-line, or use the `chsh` command to set it as your login shell.

With `lstcsh`, your commands are sent transparently for execution on faster hosts to improve response time or you can run commands on remote hosts explicitly.

`lstch` provides a high degree of network transparency. Command lines executed on remote hosts behave the same as they do on the local host. The remote execution environment is designed to mirror the local one as closely as possible by using the same values for environment variables, terminal setup, current working directory, file creation mask, and so on. Each modification to the local set of environment variables is automatically reflected on remote hosts. Note that shell variables, the `nice` value, and resource usage limits are not automatically propagated to remote hosts.

For more details on `lstcsh`, see the `lstcsh(1)` man page.

Task Lists

LSF maintains two task lists for each user, a local list (`.lsftask`) and a remote list (`lsf.task`). Commands in the local list must be executed locally. Commands in the remote list can be executed remotely.

See the *LSF Configuration Reference* for information about the `.lsftask` and `lsf.task` files.

Resource requirements for specific commands can be configured using task lists. You can optionally associate resource requirements with each command in the remote list to help LSF find a suitable execution host for the command.

If there are multiple eligible commands on a command-line, their resource requirements are combined for host selection.

If a command is in neither list, you can choose how `lstcsh` handles the command.

Change task list membership

1. Use the LSF commands `lsl tasks` and `lsrt asks` to inspect and change the memberships of local and remote task lists.

Local and remote modes

`lstcsh` has two modes of operation:

- Local
- Remote

Local mode

The local mode is the default mode. In local mode, a command line is eligible for remote execution only if all of the commands on the line are present in the remote task list, or if the `@` character is specified on the command-line to force it to be eligible.

Local mode is conservative and can fail to take advantage of the performance benefits and load-balancing advantages of LSF.

Remote mode

In remote mode, a command line is considered eligible for remote execution if none of the commands on the line is in the local task list.

Remote mode is aggressive and makes more extensive use of LSF. However, remote mode can cause inconvenience when `lstcsh` attempts to send host-specific commands to other hosts.

Automatic Remote Execution

Every time you enter a command, `lstcsh` looks in your task lists to determine whether the command can be executed on a remote host and to find the configured resource requirements for the command.

See the *LSF Configuration Reference* for information about task lists and `lsf.task` file.

If the command can be executed on a remote host, `lstcsh` contacts LIM to find the best available host.

The first time a command is run on a remote host, a server shell is started on that host. The command is sent to the server shell, and the server shell starts the command on the remote host. All commands sent to the same host use the same server shell, so the start-up overhead is only incurred once.

If no host is found that meets the resource requirements of your command, the command is run on the local host.

Differences from other shells

When a command is running in the foreground on a remote host, all keyboard input (type-ahead) is sent to the remote host. If the remote command does not read the input, it is lost.

`lstcsh` has no way of knowing whether the remote command reads its standard input. The only way to provide any input to the command is to send everything available on the standard input to the remote command in case the remote command needs it. As a result, any type-ahead entered while a remote command is running in the foreground, and not read by the remote command, is lost.

@ character

The @ character has a special meaning when it is preceded by white space. This means that the @ must be escaped with a backslash \ to run commands with arguments that start with @, like `finger`. This is an example of using `finger` to get a list of users on another host:

```
finger @other.domain
```

Normally the `finger` command attempts to contact the named host. Under `lstcsh`, the @ character is interpreted as a request for remote execution, so the shell tries to contact the RES on the host *other.domain* to remotely execute the `finger` command. If this host is not in your LSF cluster, the command fails. When the @ character is escaped, it is passed to `finger` unchanged and `finger` behaves as expected.

```
finger \@hostB
```

Limitations

A shell is a very complicated application by itself. `lstcsh` has certain limitations:

Native language system

Native Language System is not supported. To use this feature of the `tcsh`, you must compile `tcsh` with `SHORT_STRINGS` defined. This causes complications for characters flowing across machines.

Shell variables

Shell variables are not propagated across machines. When you set a shell variable locally, then run a command remotely, the remote shell will not see that shell variable. Only environment variables are automatically propagated.

fg command

The `fg` command for remote jobs must use @.

tcsh version

`lstcsh` is based on `tcsh 6.03 (7 bit mode)`. It does not support the new features of the latest `tcsh`.

Start lstcsh

If you normally use some other shell, you can start `lstcsh` from the command-line.

1. Make sure that the LSF commands are in your `PATH` environment variable, then enter:

```
lstcsh
```

If you have a `.cshrc` file in your home directory, `lstcsh` reads it to set variables and aliases.

Exit lstcsh

1. Use the `exit` command to get out of `lstcsh`.

Use lstcsh as your login shell

If your system administrator allows, you can use LSF as your login shell. The `/etc/shells` file contains a list of all the shells you are allowed to use as your login shell.

Use chsh

The `chsh` command can set your login shell to any of those shells. If the `/etc/shells` file does not exist, you cannot set your login shell to `lstcsh`.

1. Run the command:

```
chsh user3 -s /usr/share/lsf/bin/lstcsh
```

The next time `user3` logs in, the login shell will be `lstcsh`.

Use a standard system shell

If you cannot set your login shell using `chsh`, you can use one of the standard system shells to start `lstcsh` when you log in.

To set up `lstcsh` to start when you log in:

1. Use `chsh` to set `/bin/sh` to be your login shell.
2. Edit the `.profile` file in your home directory to start `lstcsh`, as shown below:

```
SHELL=/usr/share/lsf/bin/lstcsh
export SHELL
exec $SHELL -l
```

Host redirection

Host redirection overrides the task lists, so you can force commands from your local task list to execute on a remote host or override the resource requirements for a command.

You can explicitly specify the eligibility of a command-line for remote execution using the `@` character. It may be anywhere in the command line except in the first position (`@` as the first character on the line is used to set the value of shell variables).

You can restrict who can use `@` for host redirection in `lstcsh` with the parameter `LSF_SHELL_AT_USERS` in `lsf.conf`. See the *LSF Configuration Reference* for more details.

Examples

```
hostname @hostD
<< remote execution on hostD >>
hostD
hostname @/type==linux
<< remote execution on hostB >>
hostB
```

@ character

<code>@</code>	<code>@</code> followed by nothing means that the command line is eligible for remote execution.
<code>@host_name</code>	<code>@</code> followed by a host name forces the command line to be executed on that host.
<code>@local</code>	<code>@</code> followed by the reserved word <code>local</code> forces the command line to be executed on the local host only.
<code>@/res_req</code>	<code>@</code> followed by <code>/</code> and a resource requirement string means that the command is eligible for remote execution and that the specified resource requirements must be used instead of those in the remote task list.

For ease of use, the host names and the reserved word `local` following `@` can all be abbreviated as long as they do not cause ambiguity.

Similarly, when specifying resource requirements following the `@`, it is necessary to use `/` only if the first requirement characters specified are also the first characters of a host name. You do not have to type in resource requirements for each command line you type if you put these task names into remote task list together with their resource requirements by running `l srtasks`.

Task control

Task control in `l st csh` is the same as in `t csh` except for remote background tasks. `l st csh` numbers shell tasks separately for each execution host.

jobs command

The output of the built-in command `jobs` lists background tasks together with their execution hosts. This break of transparency is intentional to give you more control over your background tasks.

```
sleep 30 @hostD &
<< remote execution on hostD >>
[1] 27568
sleep 40 @hostD &
<< remote execution on hostD >>
[2] 10280
sleep 60 @hostB &
<< remote execution on hostB >>
[1] 3748
jobs
<hostD>
[1]  + Runni ng          sleep 30
[2]  + Runni ng          sleep 40
<hostB>
[1]  + Runni ng          sleep 60
```

Bring a remote background task to the foreground

1. To bring a remote background task to the foreground, the host name must be specified together with `@`, as in the following example:

```
fg %2 @hostD

<< remote execution on hostD >> sleep 40
```

Built-in commands

`l st csh` supports two built-in commands to control load sharing, `l smode` and `connect`.

lsmode

Syntax

```
lsmode [on|off] [local|remote] [e|-e] [v|-v] [t|-t]
```

Description

The `l smode` command reports that LSF is enabled if `l st csh` was able to contact LIM when it started up. If LSF is disabled, no load-sharing features are available.

The `l smode` command takes a number of arguments that control how `l st csh` behaves.

With no arguments, `l smode` displays the current settings:

```
lsmode
```

```
LSF
Copyright Platform Computing Corporation
LSF enabled, local mode, LSF on, verbose, no_eligibility_verbose,
no_timing.
```

Options

[on | off]

Turns load sharing on or off. When turned off, you can send a command line to a remote host only if force eligibility is specified with @.

The default is on.

[local | remote]

Sets `lstcsh` to use local or remote mode.

The default is local.

[e | -e]

Turns eligibility verbose mode on (e) or off (-e). If eligibility verbose mode is on, `lstcsh` shows whether the command is eligible for remote execution, and displays the resource requirement used if the command is eligible.

The default is off.

[v | -v]

Turns task placement verbose mode on (v) or off (-v). If verbose mode is on, `lstcsh` displays the name of the host on which the command is run, if the command is not run on the local host. The default is on.

[t | -t]

Turns wall-clock timing on (t) or off (-t).

If timing is on, the actual response time of the command is displayed. This is the total elapsed time in seconds from the time you submit the command to the time the prompt comes back.

This time includes all remote execution overhead. The `csch time` builtin does not include the remote execution overhead.

This is an impartial way of comparing the response time of jobs submitted locally or remotely, because all the load sharing overhead is included in the displayed elapsed time.

The default is off.

connect

Syntax

```
connect [host_name]
```

Description

`lstcsh` opens a connection to a remote host when the first command is executed remotely on that host. The same connection is used for all future remote executions on that host.

The `connect` command with no argument displays connections that are currently open.

The `connect host_name` command creates a connection to the named host. By connecting to a host before any command is run, the response time is reduced for the first remote command sent to that host.

`lstch` has a limited number of ports available to connect to other hosts. By default each shell can only connect to 15 other hosts.

Examples

```
connect
CONNECTED WITH      SERVER SHELL
hostA                +
connect hostB
Connected to hostB
connect
CONNECTED WITH      SERVER SHELL
hostA                +
hostB                -
```

In this example, the `connect` command created a connection to host `hostB`, but the server shell has not started.

Shell scripts in lstch

You should write shell scripts in `/bin/sh` and use the `lstools` commands for load sharing. However, `lstch` can be used to write load-sharing shell scripts.

By default, an `lstch` script is executed as a normal `tch` script with load-sharing disabled.

Run a script with load sharing enabled

The `lstch -L` option tells `lstch` that a script should be executed with load sharing enabled, so individual commands in the script may be executed on other hosts.

There are three different ways to run an `lstch` script with load sharing enabled:

- Run `lstch -L script_name`, or
- Make the script executable and put the following as the first line of the script. By default, `lstch` is installed in `LSF_BINDIR`.

The following assumes you installed `lstch` in the `/usr/share/lsf/bin` directory):

```
#!/usr/share/lsf/bin/lstch -L
```

1. Start an interactive `lstch`.
2. Enable load sharing, and set to remote mode:

lsmode on remote

3. Use the `source` command to read the script in.



Using Session Scheduler

About Platform Session Scheduler

While traditional Platform LSF job submission, scheduling, and dispatch methods such as job arrays or job chunking are well suited to a mix of long and short running jobs, or jobs with dependencies on each other, Session Scheduler is ideal for large volumes of independent jobs with short run times.

As clusters grow and the volume of workload increases, the need to delegate scheduling decisions increases. Session Scheduler improves throughput and performance of the LSF scheduler by enabling multiple tasks to be submitted as a single LSF job.

Platform Session Scheduler implements a hierarchical, personal scheduling paradigm that provides very low-latency execution. With very low latency per job, Session Scheduler is ideal for executing very short jobs, whether they are a list of tasks, or job arrays with parametric execution.

The Session Scheduler provides users with the ability to run large collections of short duration tasks within the allocation of an LSF job using a job-level task scheduler that allocates resources for the job once, and reuses the allocated resources for each task.

Each Session Scheduler is dynamically scheduled in a similar manner to a parallel job. Each instance of the `ssched` command then manages its own workload within its assigned allocation. Work is submitted as a task array or a task definition file.

Session Scheduler satisfies the following goals for running a large volume of short jobs:

- Minimize the latency when scheduling short jobs
- Improve overall cluster utilization and system performance
- Allocate resources according to LSF policies
- Support existing LSF pre-execution, post-execution programs, job starters, resources limits, etc.
- Handle thousands of users and more than 50000 short jobs per user

Session Scheduler system requirements

Supported operating systems Session Scheduler is delivered in the following distributions:

- `lsf8.0_ssched_linux2.4-glibc2.3-x86.tar.Z`
- `lsf8.0_ssched_linux2.6-glibc2.3-x86.tar.Z`

- `lsf8.0_ssched_linux2.6-glibc2.3-x86_64.tar.Z`

Required libraries

Note: These libraries may not be installed by default by all Linux distributions.

On Linux 2.4 (x86 and x86_64), the following external libraries are required:

- `libstdc++.so.5`
- `libpthread-0.60.so` or later
- `libgcc_s.so.1`

On Linux 2.6 (x86), the following external libraries are required:

- `libpthread-2.3.4.so` or later

On Linux 2.6 (x86_64), the following external libraries are required:

- `libstdc++.so.5`
- `libpthread-2.3.4.so` or later

Compatible Linux distributions

Certified compatible distributions include:

- Red Hat Enterprise Linux AS 3 or later
- SUSE Linux Enterprise Server 10

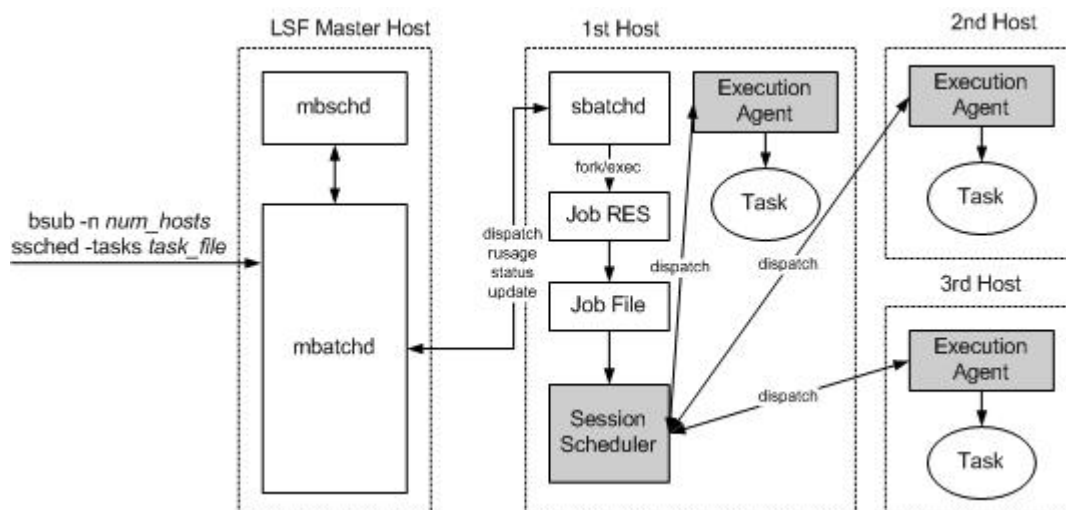
Platform LSF

Session Scheduler is included in Platform LSF 7 Update 3 or later

Session Scheduler terminology

Job	A traditional LSF job that is individually scheduled and dispatched to <code>sbatchd</code> by <code>mbatchd</code> and <code>mbschd</code>
Task	Similar to a job, a unit of workload that describes an executable and its environment that runs on an execution node. Tasks are managed and dispatched by the Session Scheduler.
Job Session	An LSF job that is individually scheduled by <code>mbatchd</code> , but is not dispatched as an LSF job. Instead, a running Session Scheduler job session represents an allocation of nodes for running large collections of tasks
Scheduler	The component that accepts and dispatches tasks within the nodes allocated for a job session.

Session Scheduler architecture



Session Scheduler jobs are submitted, scheduled, and dispatched like normal LSF jobs.

When the Session Scheduler begins running, it starts a Session Scheduler execution agent on each host in its allocation.

The Session Scheduler then reads in the task definition file, which contains a list of tasks to run. Tasks are sent to an execution agent and run. When a task finishes, the next task in the list is dispatched to the available host. This continues until all tasks have been run.

Tasks submitted through Session Scheduler bypass the LSF `mbatchd` and `mbschd`. The LSF `mbatchd` is unaware of individual tasks.

Session Scheduler components

Session Scheduler comprises the following components.

Session Scheduler command (ssched)

The `ssched` command accepts and dispatches tasks within the nodes allocated for a job session. It reads the task definition file and sends tasks to the execution agents. `ssched` also logs errors, performs task accounting, and requeues tasks as necessary.

sservice and sschild

These components are the execution agents. They run on each remote host in the allocation. They set up the task execution environment, run the tasks, and enable task monitoring and resource usage collection.

Session Scheduler performance

Session Scheduler has been tested to support up to 50,000 tasks. Based on performance tests, the best maximum allocation size (specified by `bsub -n`) depends on the average runtime of the tasks. Here are some typical results:

Average Runtime (seconds)	Recommended maximum allocation size (slots)
0	12

Average Runtime (seconds)	Recommended maximum allocation size (slots)
5	64
15	256
30	512

Session Scheduler licensing

You must license each cluster or host with a Session Scheduler license on a per core basis. Full and partial licensing modes are supported.

Full mode

In this mode, licenses are required for each core in the whole cluster. If there are not enough available licenses for the whole cluster, only the hosts enabled with Session Scheduler can run Session Scheduler tasks..

Append keyword "LSF_Session_Scheduler" for PRODUCTS in the parameter section of the lsf.cluster file as follows:

```
Begin Parameters
PRODUCTS=LSF_Make LSF_Base LSF_Manager LSF_Session_Scheduler
# LSF_HOST_ADDR_RANGE=*. *. *. *
# FLOAT_CLIENTS_ADDR_RANGE=*. *. *. *
# FLOAT_CLIENTS=10
End Parameters
```

Partial mode

In this mode, LSF requires Session Scheduler licenses for each core of selected hosts. Only hosts that are configured with "LSF_Session_Scheduler" require a license. If there are not enough "lsf_session_scheduler" licenses for all the selected hosts, only the hosts enabled with Session Scheduler can run Session Scheduler tasks.

Define "LSF_Session_Scheduler" in the RESOURCE column of the host section in the lsf.cluster file as follows:

```
Begin Host
HOSTNAME      model type server rlm mem swp RESOURCES #Keywords
w864          !    !    1      3.5 () () (mg)
i quadcore-02 !    !    1      3.5 () () (mg LSF_Session_
Scheduler)
End Host
```

Dispatching session jobs to licensed hosts

In order to tell the scheduler to dispatch session jobs to licensed hosts, you can specify a resource requirement string, either at the job level (-R) or in the application profile (RESREQ inside "ssched"). Note that the -R option is suitable for the partial licensing mode only.

```
bsub -app ssched -R "LSF_Session_Scheduler" ssched -tasks ./my.tasks
```

Administrators can also specify a dedicated hostgroup for job submission.

How Session Scheduler Runs Tasks

Once a Session Scheduler session job has been dispatched and starts running, Session Scheduler parses the task definition file specified on the ssched command. Each line of the task definition file is one task. Tasks run on the hosts in the allocation in any order. Dependencies between tasks are not supported.

Session Scheduler status is posted to the Session Scheduler session job through the LSF `bpost` command. Use `bread` or `bjobs -l` to view Session Scheduler status. The status includes the current number of pending, running and completed tasks. LSF administrators can configure how often the status is updated.

When all tasks are completed, the Session Scheduler exits normally.

`ssched` runs under the submission user account. Any processes it creates, either locally or remotely, also run under the submission user account. Session Scheduler does not require any privileges beyond those normally granted a user.

Session Scheduler job sessions

The Session Scheduler session job is compatible with all currently supported LSF job submission and execution parameters, including pre-execution, post-execution, job-starters, I/O redirection, queue and application profile configuration.

Run limits are interpreted and enforced as normal LSF parallel jobs. Application-level checkpointing is also supported. Job chunking is not relevant to Session Scheduler jobs since a single Session Scheduler session is generally long running and should not be chunked.

If the Session Scheduler session is killed (`bkill`) or requeued (`brequeue`), the Session Scheduler kills all running tasks, execution agents, and any other processes it has started, both local and remote. The session scheduler also cleans up any temporary files created and then exits. If the session scheduler is then requeued and restarted, all tasks are rerun.

If the Session Scheduler session is suspended (`bstop`), the Session Scheduler and all local and remote components will be stopped until the session is resumed (`brsume`).

Session Scheduler tasks

`ssched` and `sservice` and `sschild` execution agents ensure that the user submission environment variables are set correctly for each task. In order to minimize the load on the LSF, `mbatchd` does not have any knowledge of individual tasks.

Task definition file format

The task definition file is an ASCII file. Each line represents one task, or an array of tasks. Each line has the following format.

```
[task_options] command [arguments]
```

Session and task accounting

Jobs corresponding to the Session Scheduler session have one record in `lsb.acct`. This record represents the aggregate resource usage of all tasks in the allocation.

If task accounting is enabled with `SSCHED_ACCT_DIR` in `lsb.params`, Session Scheduler creates task accounting files for each Session Scheduler session job and appends an accounting record to the end of the file. This record follows a similar format to the LSF accounting file `lsb.acct` format, but with additional fields/

The accounting file is named `jobID.ssched.acct`. If no directory is specified, accounting records are not written.

The Session Scheduler accounting directory must be accessible and writable from all hosts in the cluster. Each Session Scheduler session (each `ssched` instance) creates one accounting file. Each file contains one accounting entry for each task. Each completed task index has one line in the file. Each line records the resource usage of one task.

Task accounting file format

Task accounting records have a similar format as the 1 sb. acct JOB_FINISH event record. See the *Platform LSF Configuration Reference* for more information about JOB_FINISH event fields.

Field	Description
Event type (%s)	TASK_FINISH
Version Number (%s)	8.0
Event Time (%d)	Time the event was logged (in seconds since the epoch)
jobId (%d)	ID for the job
userId (%d)	UNIX user ID of the submitter
options (%d)	Always 0
numProcessors (%d)	Always 1
submitTime (%d)	Task enqueue time
beginTime (%d)	Always 0
termTime (%d)	Always 0
startTime (%d)	Task start time
userName (%s)	User name of the submitter
queue (%s)	Always empty
resReq (%s)	Always empty
dependCond (%s)	Always empty
preExecCmd (%s)	Task pre-execution command
fromHost (%s)	Submission host name
cwd (%s)	Execution host current working directory (up to 4094 characters)
inFile (%s)	Task input file name (up to 4094 characters)
outFile (%s)	Task output file name (up to 4094 characters)
errFile (%s)	Task error output file name (up to 4094 characters)
jobFile (%s)	Task script file name
numAskedHosts (%d)	Always 0
askedHosts (%s)	Always empty
numExHosts (%d)	Always 1
execHosts (%s)	Name of task execution host

Field	Description
jStatus (%d)	64 indicates task completed normally. 32 indicates task exited abnormally
hostFactor (%f)	CPU factor of the task execution host
jobName (%s)	Always empty
command (%s)	Complete batch task command specified by the user (up to 4094 characters)
lsfRusage (%f)	All rusage fields contain resource usage information for the task
mailUser (%s)	Always empty
projectName (%s)	Always empty
exitStatus (%d)	UNIX exit status of the task
maxNumProcessors (%d)	Always 1
loginShell (%s)	Always empty
timeEvent (%s)	Always empty
idx (%d)	Session Job Index
maxRMem (%d)	Always 0
maxRSwap (%d)	Always 0
inFileSpool (%s)	Always empty
commandSpool (%s)	Always empty
rsvld (%s)	Always empty
sla (%s)	Always empty
exceptMask (%d)	Always 0
additionalInfo (%s)	Always empty
exitInfo (%d)	Always 0
warningAction (%s)	Always empty
warningTimePeriod (%d)	Always 0
chargedSAAP (%s)	Always empty
licenseProject (%s)	Always empty
options3 (%d)	Always 0
app (%s)	Always empty
taskID (%d)	Task ID

Field	Description
taskIdx (%d)	Task index
taskName (%s)	Task name
taskOptions (%d)	Bit mask of task options: <ul style="list-style-type: none"> TASK_IN_FILE (0x01)—specify input file TASK_OUT_FILE (0x02)—specify output file TASK_ERR_FILE (0x04)—specify error file TASK_PRE_EXEC (0x08)—specify pre-exec command TASK_POST_EXEC (0x10)—specify post-exec command TASK_NAME (0x20)—specify task name
taskExitReason (%d)	Task exit reason: <ul style="list-style-type: none"> TASK_EXIT_NORMAL = 0— normal exit TASK_EXIT_INIT = 1—generic task initialization failure TASK_EXIT_PATH = 2—failed to initialize path TASK_EXIT_NO_FILE = 3—failed to create task file TASK_EXIT_PRE_EXEC = 4— task pre-exec failed TASK_EXIT_NO_PROCESS = 5—fork failed TASK_EXIT_XDR = 6—xdr communication error TASK_EXIT_NOMEM = 7— no memory TASK_EXIT_SYS = 8—system call failed TASK_EXIT_TSCHILD_EXEC = 9—failed to run sschild TASK_EXIT_RUNLIMIT = 10—task reaches run limit TASK_EXIT_IO = 11—I/O failure TASK_EXIT_RSRC_LIMIT = 12—set task resource limit failed

Running and monitoring Session Scheduler jobs

Create a Session Scheduler session and run tasks

1. Create task definition file.

For example:

```
cat my.tasks
```

```
sleep 10
```

```
hostname
```

```
uname
```

```
ls
```

2. Use bsub with the ssched application profile to submit a Session Scheduler job with the task definition.

```
bsub -app ssched bsub_options ssched [task_options] [-tasks task_definition_file] [command [arguments]]
```

For example:

```
bsub -app ssched ssched -tasks my.tasks
```

When all tasks finish, Session Scheduler exits, all temporary files are deleted, the session job is cleaned from the system, and Session Scheduler output is captured and included in the standard LSF job e-mail. You can also submit a Session Scheduler job without a task definition file to specify a single task.

Note:

The submission directory path can contain up to 4094 characters.

See the `ssched` command reference for detailed information about all task options.

Submit a Session Scheduler job as a parallel Platform LSF job

1. Use the `-n` option of `bsub` to submit a Session Scheduler job as a parallel LSF job.

```
bsub -app ssched -n num_hosts ssched [task_options] [-tasks task_definition_file]
[command [arguments]]
```

For example:

```
bsub -app ssched -n 2 ssched -tasks my.tasks
```

Submit task array jobs

1. Use the `-J` option to submit a task array via the command line, and no task definition file is needed:

```
-J task_name[index_list]
```

The index list must be enclosed in square brackets. The index list is a comma-separated list whose elements have the syntax ***start[-end[:step]]*** where *start*, *end* and *step* are positive integers. If the step is omitted, a step of one (1) is assumed. The task array index starts at one (1).

All tasks in the array share the same option parameters. Each element of the array is distinguished by its array index.

See the `ssched` command reference for detailed information about all task options.

Submit tasks with automatic task requeue

1. Use the `-Q` option to specify requeue exit values for the tasks:

```
-Q "exit_code . . . "
```

`-Q` enables automatic task requeue and sets the `LSB_EXIT_REQUEUE` environment variable. Use spaces to separate multiple exit codes. LSF does not save the output from the failed task, and does not notify the user that the task failed.

If a job is killed by a signal, the exit value is `128+signal_value`. Use the sum of 128 and the signal value as the exit code in the parameter. For example, if you want a task to rerun if it is killed with a signal 9 (SIGKILL), the exit value is `128+9=137`.

The `SSCHED_REQUEUE_LIMIT` setting limits the number of times a task can be requeued.

See the `ssched` command reference for detailed information about all task options.

Monitor Session Scheduler jobs

1. Run `bj obs -ss` to get summary information for Session Scheduler jobs and tasks.

	JOBID	OWNER	JOB_NAME	NTASKS	PEND	DONE	RUN	EXIT
1	1	lsfadmin	job1	10	4	4	2	0
2	1	lsfadmin	job2	10	10	0	0	0
3	1	lsfadmin	job3	10	10	0	0	0

Information displays about your session scheduler job, including Job ID, the owner, the job name, the number of total tasks, and the number of tasks in any of the following states: pend, run, done, exit.

2. Use `bj obs -l -ss` or `bread` to track the progress of the Session Scheduler job.

Kill a Session Scheduler session

1. Use `bkill` to kill the Session Scheduler session. All temporary files are deleted, and the session job is cleaned from the system.

Check your job submission

1. Use the `-C` option to sanity-check all parameters and the task definition file.

`ssched` exits after the check is complete. An exit code of 0 indicates no errors were found. A non-zero exit code indicates errors. You can run `ssched -C` outside of LSF.

See the `ssched` command reference for detailed information about all task options.

Example output of `ssched -C`:

ssched -C -tasks my.tasks

Error in tasks file line 1: -XXX 123 sleep 0

Unsupported option: -XXX

Error in tasks file line 2: -o my.out

A command must be specified

Only the `ssched` parameters are checked, not the `ssched` task command itself. The task command must exist and be executable. `ssched -C` cannot detect whether the task command exists or is executable. To check a task definitions file, remember to specify the `-tasks` option.

Enable recoverable Session Scheduler sessions

By default, Session Scheduler sessions are unrecoverable. In the event of a system crash, the session job must be resubmitted and all tasks are resubmitted and rerun.

However, the Session Scheduler supports application-level checkpoint/restart using Platform LSF's existing facilities. If the user specifies a checkpoint directory when submitting the session job, the job can be restarted using `brestart`. After a restart, only those tasks that have not yet completed are resubmitted and run.

1. To enable recoverable sessions, when submitting the session job:
 - a) Provide a writable directory on a shared file system.
 - b) Specify the `ssched` checkpoint method with the `bsub -k` option.

You do not need to call `bchkpnt`. The Session Scheduler automatically checkpoints itself after each task completes.

For example:

```
bsub -app ssched -k "/share/scratch method=ssched" -n 8 ssched -tasks
simpton.tasks
```

```
Job <123> is submitted to default queue <normal>.
```

```
...
```

```
brestart /share/scratch 123
```

Resizable jobs

Enabling resizable jobs allows Session Scheduler to run jobs with minimum and maximum slots requested and dynamically use the number of slots available at any given time.

Session Scheduler automatically releases idle resources for all resizable jobs. The typical use case is a "long tail" scenario, where all short running tasks complete within one session except a few long running tasks. Those long running tasks occupy a small number of hosts, leaving most of the originally allocated resources idle. Session Scheduler automatically detects those idle resources, shuts down the execution agents running on those hosts and releases resources back to LSF.

When additional resources are added to Session Scheduler, it recognizes those resources and makes use of them to run tasks.

Resizable Job

A job whose job slot allocation can grow and shrink during its run time. The allocation change request may be triggered automatically or by the `bresize` command.

When users run the `bresize release` command to forcibly release resources from Session Scheduler, it recognizes those released resources and shuts down the execution agents running on those hosts.

Autoresizable job

A resizable job with a minimum and maximum slot request. LSF automatically schedules and allocates additional resources to satisfy job maximum request as the job runs.

Configuring resizable jobs

Session Scheduler jobs are resizable when the parameter **RESIZABLE_JOBS=Y** in the `ssched` application profile.

Session Scheduler jobs are autoresizable when the parameter **RESIZABLE_JOBS=AUTO** in the `ssched` application profile, or when **RESIZABLE_JOBS=Y** in the `ssched` application profile and jobs are submitted with the `bjobs` option `-ar`.

Session Scheduler jobs are not resizable when the parameter **RESIZABLE_JOBS** in the `ssched` application profile is commented out.

When the `bresize release` command is run on Session Scheduler jobs, the parameter **DJOB_RESIZE_GRACE_PERIOD=seconds** in the `ssched` application profile configures a time interval for Session Scheduler to react and take necessary actions such as shutting down execution agents.

Examples

Adding new resources

For an autoresizable job, new resources are added in when they become available. Consider the example where **RESIZABLE_JOBS=AUTO** in the `ssched` application profile:

```
bsub -app ssched -n 1,3 ssched -tasks ./my.tasks
```

The autoresizable job is requesting 3 hosts and only 1 is available. The job starts running on 1 host and pends on 2 more. When the additional hosts are free, the job allocation changes to 3 hosts automatically. Selected output from `bhist`:

```
Dispatched to <host1>;
Starting (Pid 24721);
"Tasks: PEND=3 RUN=0 DONE=0 EXIT=0"
"Tasks: PEND=2 RUN=1 DONE=0 EXIT=0"
"Tasks: PEND=2 RUN=1 DONE=0 EXIT=0"
Additional allocation on 1 Hosts/Processors <host2>
Resize notification acceptedExternal Message "Tasks: PEND=2 RUN=1
DONE=0 EXIT=0"
Additional allocation on 1 Hosts/Processors <host03>Resize
notification accepted
External Message "Tasks: PEND=1 RUN=2 DONE=0 EXIT=0"
External Message "Tasks: PEND=0 RUN=3 DONE=0 EXIT=0"
```

Releasing resources

Idle resources are released for both resizable and autoresizable jobs. For example, **RESIZABLE_JOBS=Y** in the `ssched` application profile and an autoresizable job is submitted:

```
bsub -ar -app ssched -n 1,3 ssched -tasks ./my.longtail
```

The autoresizable job is requesting 1 to 3 hosts and 3 are available so the job starts running on 3 right away. This longtail job has many short tasks and only a few long ones, and is soon finished running on 2 of the 3 hosts. Session Scheduler sees the 2 idle hosts and releases them from the allocation. Selected output from `bhist`:

```
Submitted from host <host11>, to Queue <normal>, CWD <$HOME>, 3
Processors Requested
Dispatched to 3 Hosts/Processors <host01> <host02> <host03>;
"Tasks: PEND=3 RUN=0 DONE=0 EXIT=0"
"Tasks: PEND=0 RUN=3 DONE=0 EXIT=0"
```

After two tasks are done, Session Scheduler releases two hosts:

```
Release allocation on 2 Hosts/Processors <host02> <host03> by user or
administrator <user01>, Cancel pending allocation request;
```

```
Resize notification accepted;
```

```
"Tasks: PEND=0 RUN=1 DONE=2 EXIT=0"
```

Releasing resources using bresize

Resources can be released on demand for both resizable and autoresizable jobs. For example, **DJOB_RESIZE_GRACE_PERIOD=30** in **RESIZABLE_JOBS=Y** in the ssched application profile and a resizable job is submitted:

```
bsub -app ssched -n 1,3 ssched -tasks ./my.longtail
```

```
Job <5> is submitted to the default queue <normal>
```

The resizable job starts running on host02, host03, and host04. The following command releases host02 from the job:

```
bresize release "host02" 5
```

Session Scheduler shuts down the execution agent on host02. The job continues to run on host03 and host04.

Selected output from bhist:

```
Submitted from host <delpe07.lsf.platform.com>, to Queue <normal>, CWD <
$HOME>, 3 Processors Requested;
```

```
Dispatched to 3 Hosts/Processors <host02> <host03> <host04>;
```

```
"Tasks: PEND=3 RUN=0 DONE=0 EXIT=0"
```

```
"Tasks: PEND=0 RUN=3 DONE=0 EXIT=0"
```

```
Release allocation on 1 Hosts/Processors <host02> by user or administrator
<user01>
```

```
Resize notification accepted
```

```
"Tasks: PEND=1 RUN=2 DONE=0 EXIT=0"
```

Troubleshooting

Use any of the following methods to troubleshoot your Session Scheduler jobs.

ssched environment variables

Before submitting the ssched command, You can set the following environment variables to enable additional debugging information:

SSCHED_DEBUG_LOG_MASK=[LOG_INFO | LOG_DEBUG | LOG_DEBUG1 | ...] Controls the amount of logging

SSCHED_DEBUG_CLASS=ALL or
SSCHED_DEBUG_CLASS=[LC_TRACE]
[LC_FILE] [...]

- Filters out some log classes, or shows all log classes
- By default, no log classes are shown

SSCHED_DEBUG_MODULES=ALL or
SSCHED_DEBUG_MODULES=[ssched]
[libvem.so] [sservice] [sschild]

- Enables logging on some or all components
- By default, logging is disabled on all components

- `libvem.so` controls logging by the `libvem.so` loaded by the SD, SSM and `ssched`
 - Enabling debugging of the Session Scheduler automatically enables logging by the `libvem.so` loaded by the Session Scheduler
- `SSCHED_DEBUG_REMOTE_HOSTS=ALL` or
`SSCHED_DEBUG_REMOTE_HOSTS=[hostname1] [hostname2] [...]`
- Enables logging on some/all hosts
 - By default, logging is disabled on all remote hosts
- `SSCHED_DEBUG_REMOTE_FILE=Y`
- Directs logging to `/tmp/ssched/job_ID.job_index/` instead of `stderr` on each remote host
 - Useful if too much debugging info is slowing down the network connection
 - By default, debugging info is sent to `stderr`

ssched debug options

The `ssched` options -1, -2, and -3 are shortcuts for the following environment variables.

- `ssched -1` Is a shortcut for:
- `SSCHED_DEBUG_LOG_MASK=LOG_WARNING`
 - `SSCHED_DEBUG_CLASS=ALL`
 - `SSCHED_DEBUG_MODULES=ALL`
- `ssched -2` Is a shortcut for:
- `SSCHED_DEBUG_LOG_MASK=LOG_INFO`
 - `SSCHED_DEBUG_CLASS=ALL`
 - `SSCHED_DEBUG_MODULES=ALL`
- `ssched -3` Is a shortcut for:
- `SSCHED_DEBUG_LOG_MASK=LOG_DEBUG`
 - `SSCHED_DEBUG_CLASS=ALL`
 - `SSCHED_DEBUG_MODULES=ALL`

Example output of `ssched -2`:

```
Nov 20 14:35:01 2011 4546 6 8.0 SSCHED_UPDATE_SUMMARY_INTERVAL = 1
Nov 20 14:35:01 2011 4546 6 8.0 SSCHED_UPDATE_SUMMARY_BY_TASK = 0
Nov 20 14:35:01 2011 4546 6 8.0 SSCHED_REQUEUE_LIMIT = 1
Nov 20 14:35:01 2011 4546 6 8.0 SSCHED_RETRY_LIMIT = 1
Nov 20 14:35:01 2011 4546 6 8.0 SSCHED_MAX_TASKS = 10
Nov 20 14:35:01 2011 4546 6 8.0 SSCHED_MAX_RUNLIMIT = 600
Nov 20 14:35:01 2011 4546 6 8.0 SSCHED_ACCT_DIR = /home/user1/ssched
Nov 20 14:35:03 2011 4546 6 8.0 Task <1[1]> submitted. Command <sleep 0>;
Nov 20 14:35:03 2011 4546 6 8.0 Task <1[2]> submitted. Command <sleep 0>;
Nov 20 14:35:03 2011 4546 6 8.0 Task <1[3]> submitted. Command <sleep 0>;
Nov 20 14:35:03 2011 4546 6 8.0 Task <1[4]> submitted. Command <sleep 0>;
Nov 20 14:35:03 2011 4546 6 8.0 Task <1[5]> submitted. Command <sleep 0>;
Nov 20 14:35:05 2011 4546 6 8.0 Task <1[1]> done successfully. The CPU time used is
0.030993 seconds;
```

```

Nov 20 14:35:05 2011 4546 6 8.0 Task <1[2]> done successfully. The CPU time used is
0.039992 seconds;
Nov 20 14:35:05 2011 4546 6 8.0 Task <1[3]> done successfully. The CPU time used is
0.033993 seconds;
Nov 20 14:35:05 2011 4546 6 8.0 Task <1[4]> done successfully. The CPU time used is
0.026994 seconds;
Nov 20 14:35:05 2011 4546 6 8.0 Task <1[5]> done successfully. The CPU time used is
0.036992 seconds;
Task Summary
Submitted:          5
Done:              5

```

Example output of ssched -2 with requeue

```

Nov 20 14:35:47 2011 4748 6 8.0 SSCHED_UPDATE_SUMMARY_INTERVAL = 1
Nov 20 14:35:47 2011 4748 6 8.0 SSCHED_UPDATE_SUMMARY_BY_TASK = 0
Nov 20 14:35:47 2011 4748 6 8.0 SSCHED_REQUEUE_LIMIT = 1
Nov 20 14:35:47 2011 4748 6 8.0 SSCHED_RETRY_LIMIT = 1
Nov 20 14:35:47 2011 4748 6 8.0 SSCHED_MAX_TASKS = 10
Nov 20 14:35:47 2011 4748 6 8.0 SSCHED_MAX_RUNLIMIT = 600
Nov 20 14:35:47 2011 4748 6 8.0 SSCHED_ACCT_DIR = /home/user1/ssched
Nov 20 14:35:49 2011 4748 6 8.0 Task <1> submitted. Command <exit 1>;
Nov 20 14:35:50 2011 4748 6 8.0 Task <1> exited with status 1.
Nov 20 14:35:50 2011 4748 6 8.0 Task <1> submitted. Command <exit 1>;
Nov 20 14:35:50 2011 4748 6 8.0 Task <1> exited with status 1.
Task Summary
Submitted:          1
Requeued:           1
Done:               0
Exited:             2
  Execution Errors: 2
  Dispatch Errors: 0
  Other Errors:     0
Task Error Summary
Execution Error
Task ID:            1
Submit Time:        Tue Nov 20 14:35:49 2011
Start Time:         Tue Nov 20 14:35:50 2011
End Time:           Tue Nov 20 14:35:50 2011
Exit Code:          1
Exit Reason:        Normal exit
Exec Hosts:         ibm03
Exec Home:          /home/user1
Exec Dir:           /home/user1/src/l sf7ss/ssched/ssched
Command:            exit 1
Action:             Requeue exit value match; task will be requeued
Execution Error
Task ID:            1
Submit Time:        Tue Nov 20 14:35:50 2011
Start Time:         Tue Nov 20 14:35:50 2011
End Time:           Tue Nov 20 14:35:50 2011
Exit Code:          1
Exit Reason:        Normal exit
Exec Hosts:         ibm03
Exec Home:          /home/user1
Exec Dir:           /home/user1/src/l sf7ss/ssched/ssched
Command:            exit 1
Action:             Task requeue limit reached; task will not be requeued

```

Example output of ssched -2 with retry

```

Nov 20 14:37:04 2011 5049 6 8.0 SSCHED_UPDATE_SUMMARY_INTERVAL = 1
Nov 20 14:37:04 2011 5049 6 8.0 SSCHED_UPDATE_SUMMARY_BY_TASK = 0
Nov 20 14:37:04 2011 5049 6 8.0 SSCHED_REQUEUE_LIMIT = 1
Nov 20 14:37:04 2011 5049 6 8.0 SSCHED_RETRY_LIMIT = 1
Nov 20 14:37:04 2011 5049 6 8.0 SSCHED_MAX_TASKS = 10
Nov 20 14:37:04 2011 5049 6 8.0 SSCHED_MAX_RUNLIMIT = 600
Nov 20 14:37:04 2011 5049 6 8.0 SSCHED_ACCT_DIR = /home/user1/ssched
Nov 20 14:37:06 2011 5049 6 8.0 Task <1> submitted. Command <sleep 0>;
Nov 20 14:37:08 2011 5049 6 8.0 Task <1> had a dispatch error.
Nov 20 14:37:08 2011 5049 6 8.0 Task <1> submitted. Command <sleep 0>;
Nov 20 14:37:08 2011 5049 6 8.0 Task <1> had a dispatch error.

```

```

Task Summary
Submitted:          1
Done:               0
Exited:             1
  Execution Errors: 0
  Dispatch Errors:  1
  Other Errors:     0
Task Error Summary
Dispatch Error
Task ID:            1
Submit Time:        Tue Nov 20 14:37:06 2011
Failure Reason:     Pre-execution command failed
Command:            sleep 0
Pre-Exec:           exit 1
Start time:         Tue Nov 20 14:37:07 2011
Execution host:     ibm03
Action:             Task will be retried
Dispatch Error
Task ID:            1
Submit Time:        Tue Nov 20 14:37:08 2011
Failure Reason:     Pre-execution command failed
Command:            sleep 0
Pre-Exec:           exit 1
Start time:         Tue Nov 20 14:37:08 2011
Execution host:     ibm03
Action:             Task retry limit reached; task will not be retried

```

Note:

The "Task Summary" and "Summary of Errors" sections are sent to `stdout`. All other output is sent to `stderr`.

Send SIGUSR1 signal

After the tasks have been submitted to the Session Scheduler and started, users can enable additional debugging by Session Scheduler components by sending a SIGUSR1 signal.

To enable additional debugging by the `ssched` and `libvem` components, send a SIGUSR1 to the `ssched_real` process. This enables the following:

- `SSCHED_DEBUG_LOG_MASK=LOG_DEBUG`
- `SSCHED_DEBUG_CLASS=ALL`
- `SSCHED_DEBUG_MODULES=ALL`

The additional log messages are sent to `stderr`.

To enable additional debugging by the `sservice` and `sschld` components, send a SIGUSR1 on the remote host to the `sservice` process. This enables the following:

- `SSCHED_DEBUG_LOG_MASK=LOG_DEBUG`
- `SSCHED_DEBUG_CLASS=ALL`
- `SSCHED_DEBUG_MODULES=ALL`
- `SSCHED_DEBUG_REMOTE_HOSTS=ALL`
- `SSCHED_DEBUG_REMOTE_FILE=Y`

The debug messages are saved to a file in `/tmp/ssched/`. You are responsible for deleting this file when it is no longer needed.

Send SIGUSR2 signal

If a SIGUSR1 signal is sent, SIGUSR2 restores debugging to its original level.

Known issues and limitations

General issues

- The Session Scheduler caches host info from LIM. If the host factor of a host is changed after the Session Scheduler starts, the Session Scheduler will not see the updated host factor. The host factor is used in the task accounting log.
- Session Scheduler does not support per task memory or swap utilization tracking from `ssacct`. Run `bacct` to see aggregate memory and swap utilization.
- When specifying a multiline command line as a `ssched` command line parameter, you must enclose the command in quotes. A multiline command line is any command containing a semi-colon (;). For example:

```
ssched -o my.out "hostname; ls"
```

When specifying a multiline command line as a parameter in a task definition file, you must NOT use quotes. For example:

```
cat my.tasks
```

```
-o my.out hostname; ls
```

- If you submit a shell script containing multiple `ssched` commands, `bj obs -l` only shows the task summary for the currently running `ssched` instance. Enable task accounting and examine the accounting file to see information for tasks from all `ssched` instances in the shell script.
- Submitting a large number of tasks as part of one session may cause a slight delay between when the Session Scheduler starts and when tasks are dispatched to execution agents. The Session Scheduler must parse and submit each task before it begins dispatching any tasks. Parsing 50,000 tasks can take up to 2 minutes before dispatching starts.
- After all tasks have completed, the Session Scheduler will take some time to terminate all execution agents and to clean up temporary files. A minimum of 20 seconds is normal, longer for larger allocations.
- Session Scheduler handles the following signals: `SIGINT`, `SIGTERM`, `SIGUSR1`, `SIGSTOP`, `SIGTSTP`, and `SIGCONT`. All other signals cause `ssched` to exit immediately. No summary is output and task accounting information is not saved. The signals Session Scheduler handles will be expanded in future releases.



Using Ismake

Platform Make is a load-sharing, parallel version of GNU Make. It uses the same makefiles as GNU Make and behaves similarly, except that additional command line options control parallel execution.

The Platform Make executable, `lsmake`, is covered by the Free Software Foundation General Public License. Read the file `LSF_MISC/lsmake/COPYING` in the Platform LSF software distribution for details.

Platform LSF is a prerequisite for Platform Make. The Platform Make product is sold, licensed, distributed, and installed separately. For more information, contact Platform Computing.

Platform Make is only supported on UNIX.

About Platform Make

Platform Make allows you to use your Platform LSF cluster to run parts of your make in parallel. Tasks are started on multiple hosts simultaneously to reduce the execution time.

Tasks often consist of many subtasks, with some dependencies between the subtasks. For example, to compile a software package, you compile each file in the package, then link all the compiled files together.

In many cases, most of the subtasks do not depend on each other. For a software package, the individual files in the package can be compiled at the same time; only the linking step needs to wait for all the other tasks to complete.

Platform Make supports following standard LSF command debug options:

- `LSF_CMD_LOGDIR`
- `LSF_CMD_LOG_MASK`
- `LSF_DEBUG_CMD`
- `LSF_TIME_CMD`
- `LSF_NIOS_DEBUG`

Licensing

You must license each cluster or host with a Platform Make license on a per-core basis. Full and partial licensing modes are supported.

GNU Make compatibility

Platform Make is based on GNU Make and supports most GNU Make features. GNU Make is upwardly compatible with the make programs supplied by most UNIX vendors. Platform Make is compatible with makefiles for most versions of GNU Make.

Platform Make is fully compatible with GNU Make version 3.81. There are some incompatibilities between GNU Make and some other versions of make; these are beyond the scope of this document.

How Platform Make works

Platform Make is invoked using the `lsmake` command. For command syntax and complete information about command line options that control load sharing, see `lsmake` in the *Platform LSF Command Reference*.

lsmake command

Attention:

The submission host is always one of the hosts selected to run the job, unless you have used `-m` (choose hosts by name) or `-R` (choose hosts with special resource requirements) to define some host selection criteria that excludes it.

Furthermore, for this command only, the resource requirement string gives precedence to the submission host when choosing the best available hosts for the job. If you define resource requirements, and the submission host meets the criteria defined in the selection string, the submission host is always selected. The order string is only used to sort the other hosts.

The following examples show how to build your software in parallel and control the execution hosts used, the number of cores used, and the number of tasks run simultaneously on one core.

% lsmake -f mymakefile

`lsmake` uses one core on the submission host, and runs one task at a time (one task per core). This is the default behavior.

% lsmake -R "swp > 50 && mem > 100" -f mymakefile

`lsmake` uses one core on the submission host or best available host that satisfies the specified resource requirements, and runs one task at a time. If there are no eligible hosts, the job fails.

By default, Platform Make selects the same host type as the submitting host. This is necessary for most compilation jobs. All components must be compiled on the same host type and operating system version to run correctly. If your make task requires other resources, override the default resource requirements with `-R`.

% lsmake -V -j 3 -f mymakefile

```
[hostA] [hostD] [hostK]
<< Execute on local host >>
cc -O -c arg.c -o arg.o
<< Execute on remote host hostA >>
cc -O -c dev.c -o dev.o
<< Execute on remote host hostK >>
cc -O -c main.c -o main.o
<< Execute on remote host hostD >>
cc -O arg.o dev.o main.o
```

lsmake uses 3 cores, on hosts that are the same host type as the submission host. Use -V to return output as shown, including the names of the execution hosts. Use -j to specify a maximum number of cores.

If 5 cores are eligible, Platform Make automatically selects 3, the submission host and the best 2 of the remaining hosts.

If only 2 cores are eligible, Platform Make uses only 2 cores. At least one core is always eligible because the submission host always meets the default requirement.

```
% lsmake -R "swp > 50 && mem > 100" -j 3 -c 2 -f mymakefile
```

lsmake uses up to 3 cores, on hosts that satisfy the specified resource requirements, and starts 2 tasks on each core. If there are no eligible hosts, the job fails.

Use -c to take advantage of parallelism between the CPU and I/O on a powerful host and specify the number of concurrent jobs for each core.

```
% lsmake -m "hostA 2 hostB" -f mymakefile
```

lsmake uses 2 cores on hostA and one core on hostB, and runs one task per core. Use -m to specify exactly which hosts to use.

Use GNU make options

Platform Make supports all the GNU Make command line options. See the `gmake(1)` man page.

Reset environment variables

By default, Platform Make sets the environment variables on the execution hosts once, when you run lsmake. If your tasks overwrite files or environment variables during execution, use -E to automatically reset the environment variables for every task that executes on a remote host.

Run interactive tasks

When Platform Make is running processes on more than one host, it does not send standard input to the remote processes. Most makefiles do not require any user interaction through standard I/O.

Run lsmake under Platform LSF

Make jobs often require a lot of resources, but no user interaction. Such jobs can be submitted to LSF so that they are processed when the needed resources are available. The command lsmake includes extensions to run as a parallel batch job under LSF:

```
% bsub -n 10 lsmake
```

This command queues a Platform Make job that needs 10 job slots. When all 10 slots are available, LSF starts Platform Make on the first host, and passes the names of all hosts in an environment variable. Platform Make gets the host names from the environment variable and uses RES to run tasks.

You can also specify a minimum and maximum number of slots to dedicate to your make job:

```
% bsub -n 6,18 lsmake
```

Because Platform Make passes the suspend signal (SIGTSTP) to all its remote processes, the entire parallel make job can be suspended and resumed by the user or by LSF.

Output tagging

You can enable output tagging to prefix the sender's task ID to the parallel task data of the `lsmake` command. The following examples show the differences between the standard output and the tagged output of the `lsmake` command.

The following is the standard output from an `lsmake` session running in parallel:

```
% lsmake -j 3
```

```
echo sub1 ; sleep 1000
sub1
echo sub2 ; sleep 1000
echo sub3 ; sleep 1000
sub2
sub3
```

The following is the tagged output from an `lsmake` session running in parallel:

```
% lsmake -T -j 3
```

```
T1<local>: echo sub1 ; sleep 1000
T1<local>: sub1
T2<hostD>: echo sub2 ; sleep 1000
T3<hostA>: echo sub3 ; sleep 1000
T2<hostD>: sub2
T3<hostA>: sub3
```

The following is the tagged output from an `lsmake` session that includes the names of the hosts used:

```
% lsmake -T -V -j 3
```

```
<hostA> <hostD>
<< Execute T1 on host hostA >>
T1<local>: echo sub1 ; sleep 1000
T1<local>: sub1
<< Execute T2 on remote host hostD >>
T2<hostD>: echo sub2 ; sleep 1000
<< Execute T3 on host hostA >>
T3<hostA>: echo sub3 ; sleep 1000
T2<hostD>: sub2
T3<hostA>: sub3
```

Ismake performance

Ways to improve the performance of Platform Make:

- Tune your makefile and increase parallelism
- Process subdirectories in parallel
- Adjust the number of tasks run depending on the file server load
- Ensure tasks always run on the best cores available at the time
- Compensate for file system latency
- Analyze resource usage to improve performance and efficiency

Reorganize your makefile

You do not need to modify your makefile to use Platform Make, but reorganizing the contents of the makefile to increase the parallelism might reduce the running time.

The smallest unit that Platform Make runs in parallel is a single make rule. If your makefile has rules that include many steps, or rules that contain shell loops to build sub-parts of your project, Platform Make runs the steps serially.

Increase the parallelism in your makefile by breaking up complex rules into groups of simpler rules. Steps that must run in sequence can use make dependencies to enforce the order. Platform Make can then find more subtasks to run in parallel.

Compensate for file system latency

Whenever a command depends on results of a previous command, running the commands on different hosts may result in errors due to file system latency. The `-x` and `-a` options are two ways to prevent problems. Use `-x` to automatically rerun a command that has failed for any reason. Use `-a` when you have dependent targets that may run on different hosts, and you need to allow extra time in between for file synchronization. By default, the dependent target (if it runs on a different host) starts after a delay of 1 second.

For any target, the retry feature (`-x`) is useful to compensate for file system latency and minor errors. With this feature enabled, the system automatically reruns any command that fails. You control how many times the same command should be rerun (for example, if the number of retries is 1, the command is attempted twice before exiting).

For dependent targets, the `-a` option is most useful. Ideally, dependent targets run sequentially on the same execution host, and files generated or modified by the previous target are available immediately. However, the dependent target may run on a different host (if the first host is busy running another command, or the target has multiple dependencies). If you notice errors in these cases, use `-a` to define a larger buffer time to compensate for file system latency. By default, the buffer time is 1 second.

This feature allows time for the shared file system to synchronize client and server. When commands in a target finish, commands in a dependent target wait the specified time before starting on a different host. If the dependent target's commands start on the same execution host, there is no delay. Slower file systems require a longer delay, so configure this based on network performance at your site.

If retry is enabled, this buffer time also affects the timing of retry attempts. The interval between retries increases exponentially with each retry attempt. The time between the initial, failed attempt and the first retry is equal to the buffer time. For subsequent attempts, the interval between attempts is doubled each time.

For example, if the buffer time defined by `-a` is 3 seconds and the number of retries defined by `-x` is 4, the system will wait 3 seconds before the first retry, then wait 6 seconds for the second retry, then 12 seconds, then 24, and exit if the 4th retry fails. However, if the dependent target can start on the same execution host at any time before exiting, it does so immediately, because the delay between retries is only enforced when the dependent target runs on a different host.

Analyze resource usage

When you run `lsmake`, you can use the `summay (-y)` and `usage (-u)` options to learn if resources are being used efficiently and if resource availability may be limiting performance.

Use `-y` to display information about the job run time, hosts and slots allocated, and the highest number of tasks that ran in parallel. With this information, you can know if you requested more slots than the job actually needed.

Summary output:

Total Run Time - Total `lsmake` job run time, in the format *hh:mm:ss*

Most Concurrent Tasks - Maximum number of tasks that ran simultaneously; compare to Total Slots Allocated and Tasks Allowed per Slot to determine if parallel execution may have been limited by resource availability

Retries Allowed - Maximum number of retries allowed (set by `lsmake -x` option)

Hosts and Number of Slots Allowed - Execution hosts allocated, and the number of slots allocated on each. The output is a single line showing each name and number pair separated by spaces, in the format: *host_name number_of_slots*

Tasks Allowed per Slot - Maximum number of tasks allowed per slot (set by `lsmake -c` option)

Total Slots Allocated - Total number of slots actually allocated (may be limited by `lsmake -j` or `-m` options)

Use `-u` to generate a data file tracking the number of tasks running over time, which tells you how many slots were actually used and when they were needed. This file is useful if you want to export the data to third-party charting applications.

`lsmake.dat` file format:

The file is a simple text file, each line consists of just two values, separated by a comma. The first value is the time in the format *hh:mm:ss*, the second value is the number of tasks running at that time, for example:

```
23: 13: 39, 2
```

The file is updated with a new line of information every second.



Managing Platform LSF on EGO

About LSF on Platform EGO

LSF on Platform EGO allows EGO to serve as the central resource broker, enabling enterprise applications to benefit from sharing of resources across the enterprise grid.

- *Scalability*—EGO enhances LSF scalability. Currently, the LSF scheduler has to deal with a large number of jobs. EGO provides management functionality for multiple schedulers that co-exist in one EGO environment. In LSF Version 8, although only a single instance of LSF is available on EGO, the foundation is established for greater scalability in follow-on releases that will allow multiple instances of LSF on EGO.
- *Robustness*—In previous releases, LSF functioned as both scheduler and resource manager. EGO decouples these functions, making the entire system more robust. EGO reduces or eliminates downtime for LSF users while resources are added or removed.
- *Reliability*—In situations where service is degraded due to noncritical failures such as sbatchd or RES, by default, LSF does not automatically restart the daemons. The EGO Service Controller can monitor all LSF daemons and automatically restart them if they fail. Similarly, the EGO Service Controller can also monitor and restart other critical processes such as FlexNet and lmgrd.
- *Additional scheduling functionality*—EGO provides the foundation for EGO-enabled SLA, which provides LSF with additional and important scheduling functionality.
- *Centralized management and administration framework.*
- *Single reporting framework*—across various application heads built around EGO.

What is Platform EGO?

Platform Enterprise Grid Orchestrator (EGO) allows developers, administrators, and users to treat a collection of distributed software and hardware resources on a shared computing infrastructure (cluster) as parts of a single virtual computer.

EGO assesses the demands of competing business services (consumers) operating within a cluster and dynamically allocates resources so as to best meet a company's overriding business objectives. These objectives might include

- Reducing the time or the cost of providing key business services
- Maximizing the revenue generated by existing computing infrastructure
- Configuring, enforcing, and auditing service plans for multiple consumers

- Ensuring high availability and business continuity through disaster scenarios
- Simplifying IT management and reducing management costs
- Consolidating divergent and mixed computing resources into a single virtual infrastructure that can be shared transparently between many business users

Platform EGO also provides a full suite of services to support and manage resource orchestration. These include cluster management, configuration and auditing of service-level plans, resource facilitation to provide fail-over if a master host goes down, monitoring and data distribution.

EGO is only sensitive to the *resource requirements* of business services; EGO has no knowledge of any run-time dynamic parameters that exist for them. This means that EGO does not interfere with how a business service chooses to use the resources it has been allocated.

How Platform EGO works

Platform products work in various ways to match business service (consumer) demands for resources with an available supply of resources. While a specific clustered application manager or consumer (for example, an LSF cluster) identifies what its resource demands are, Platform EGO is responsible for supplying those resources. Platform EGO determines the number of resources each consumer is entitled to, takes into account a consumer's priority and overall objectives, and then allocates the number of required resources (for example, the number of slots, virtual machines, or physical machines).

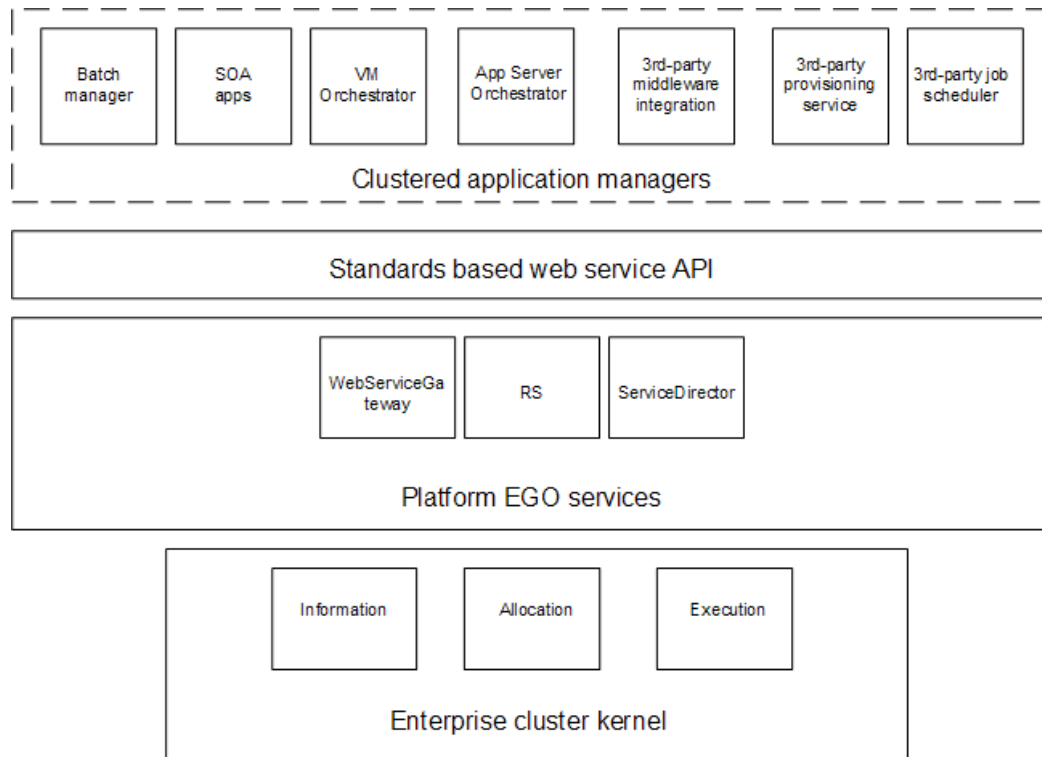
Once the consumer receives its allotted resources from Platform EGO, the consumer applies its own rules and policies. How the consumer decides to balance its workload across the fixed resources allotted to it is not the responsibility of EGO.

So how does Platform EGO know the demand? Administrators or developers use various EGO interfaces (such as the SDK or CLI) to tell EGO what constitutes a demand for more resources. When Platform LSF identifies that there is a demand, it then distributes the required resources based on the resource plans given to it by the administrator or developer.

For all of this to happen smoothly, various components are built into Platform EGO. Each EGO component performs a specific job.

Platform EGO components

Platform EGO comprises a collection of cluster orchestration software components. The following figure shows overall architecture and how these components fit within a larger system installation and interact with each other:



Key EGO concepts

Consumers	A consumer represents an entity that can demand resources from the cluster. A consumer might be a business service, a business process that is a complex collection of business services, an individual user, or an entire line of business.
EGO resources	<p>Resources are physical and logical entities that can be requested by a client. For example, an application (client) requests a processor (resource) in order to run.</p> <p>Resources also have attributes. For example, a host has attributes of memory, processor utilization, operating systems type, etc.</p>
Resource distribution tree	The resource distribution tree identifies consumers of the cluster resources, and organizes them into a manageable structure.
Resource groups	Resource groups are logical groups of hosts. Resource groups provide a simple way of organizing and grouping resources (hosts) for convenience; instead of creating policies for individual resources, you can create and apply them to an entire group. Groups can be made of resources that satisfy a specific requirement in terms of OS, memory, swap space, CPU factor and so on, or that are explicitly listed by name.
Resource distribution plans	<p>The resource distribution plan, or resource plan, defines how cluster resources are distributed among consumers. The plan takes into account the differences between consumers and their needs, resource properties, and various other policies concerning consumer rank and the allocation of resources.</p> <p>The distribution priority is to satisfy each consumer's reserved ownership, then distribute remaining resources to consumers that have demand.</p>

Services	<p>A service is a self-contained, continuously running process that accepts one or more requests and returns one or more responses. Services may have multiple concurrent service instances running on multiple hosts. All Platform EGO services are automatically enabled by default at installation.</p> <p>Run <code>egosh</code> to check service status.</p> <p>If EGO is disabled, the <code>egosh</code> command cannot find <code>ego.conf</code> or cannot contact <code>vemkd</code> (not started), and the following message is displayed:</p> <pre>You cannot run the egosh command because the administrator has chosen not to enable EGO in lsf.conf: LSF_ENABLE_EGO=N.</pre>
EGO user accounts	<p>A user account is a Platform system user who can be assigned to any role for any consumer in the tree. User accounts include optional contact information, a name, and a password.</p>

LSF and EGO directory structure

The following tables describe the purpose of each sub-directory and whether they are writable or non-writable by LSF.

LSF_TOP

Directory Path	Description	Attribute
LSF_TOP/8.0	LSF 8.0 binaries and other machine dependent files	Non-writable
LSF_TOP/conf	LSF 8.0 configuration files You must be LSF administrator or root to edit files in this directory	Writable by the LSF administrator, master host, and master candidate hosts
LSF_TOP/log	LSF 8.0 log files	Writable by all hosts in the cluster
LSF_TOP/work	LSF 8.0 working directory	Writable by the master host and master candidate hosts, and is accessible to slave hosts

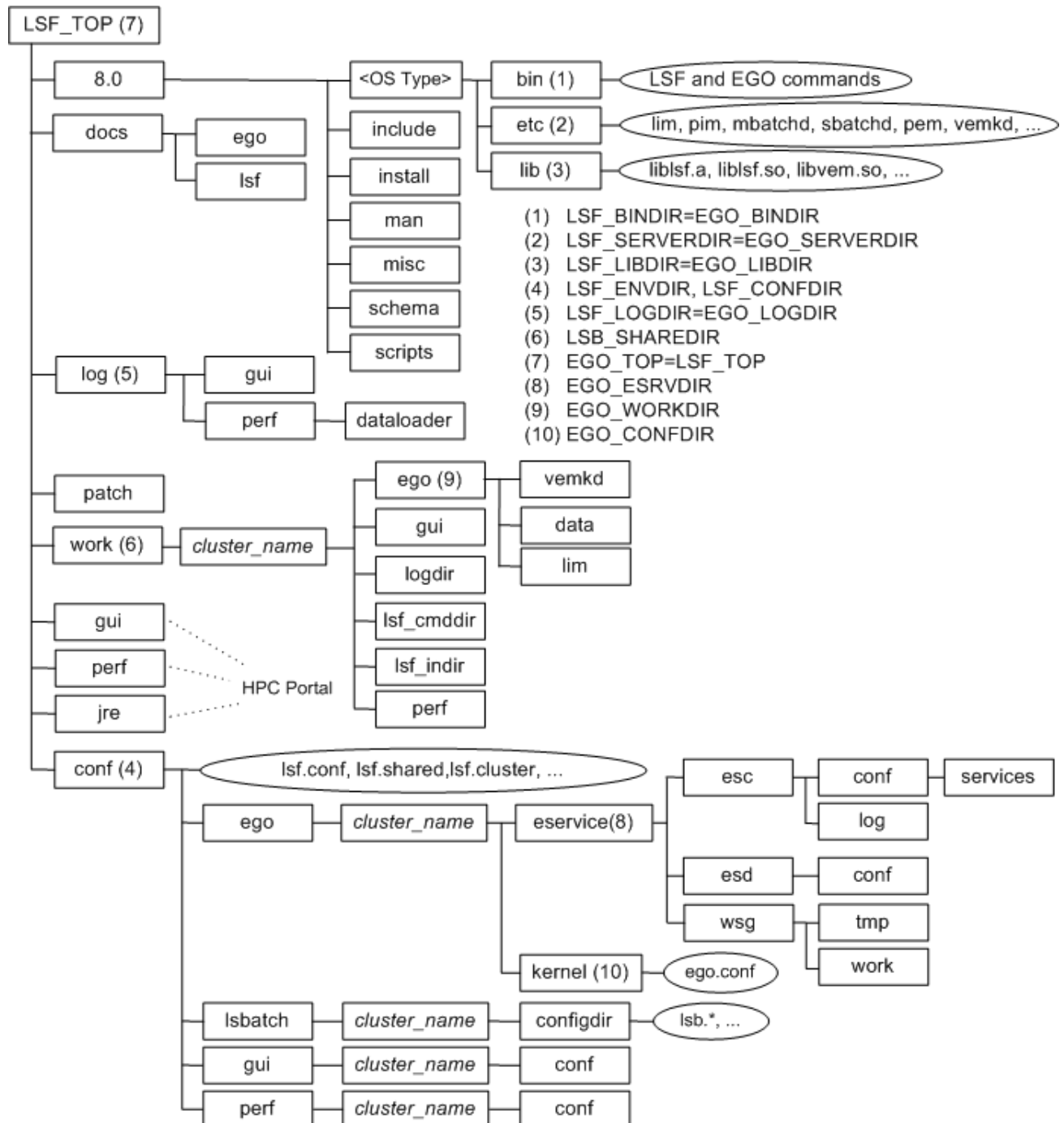
EGO directories

Directory Path	Description	Attribute
LSF_BINDIR	EGO binaries and other machine dependent files	Non-writable
LSF_CONFDIR/ego/ <i>cluster_name</i> /eservice (EGO_ESRVDIR)	EGO services configuration and log files.	Writable
LSF_CONFDIR/ego/ <i>cluster_name</i> /kernel (EGO_CONFDIR, LSF_EGO_ENVDIR)	EGO kernel configuration, log files and working directory, including conf/log/work	Writable
LSB_SHAREDIR/ <i>cluster_name</i> /ego (EGO_WORKDIR)	EGO working directory	Writable

Example directory structures

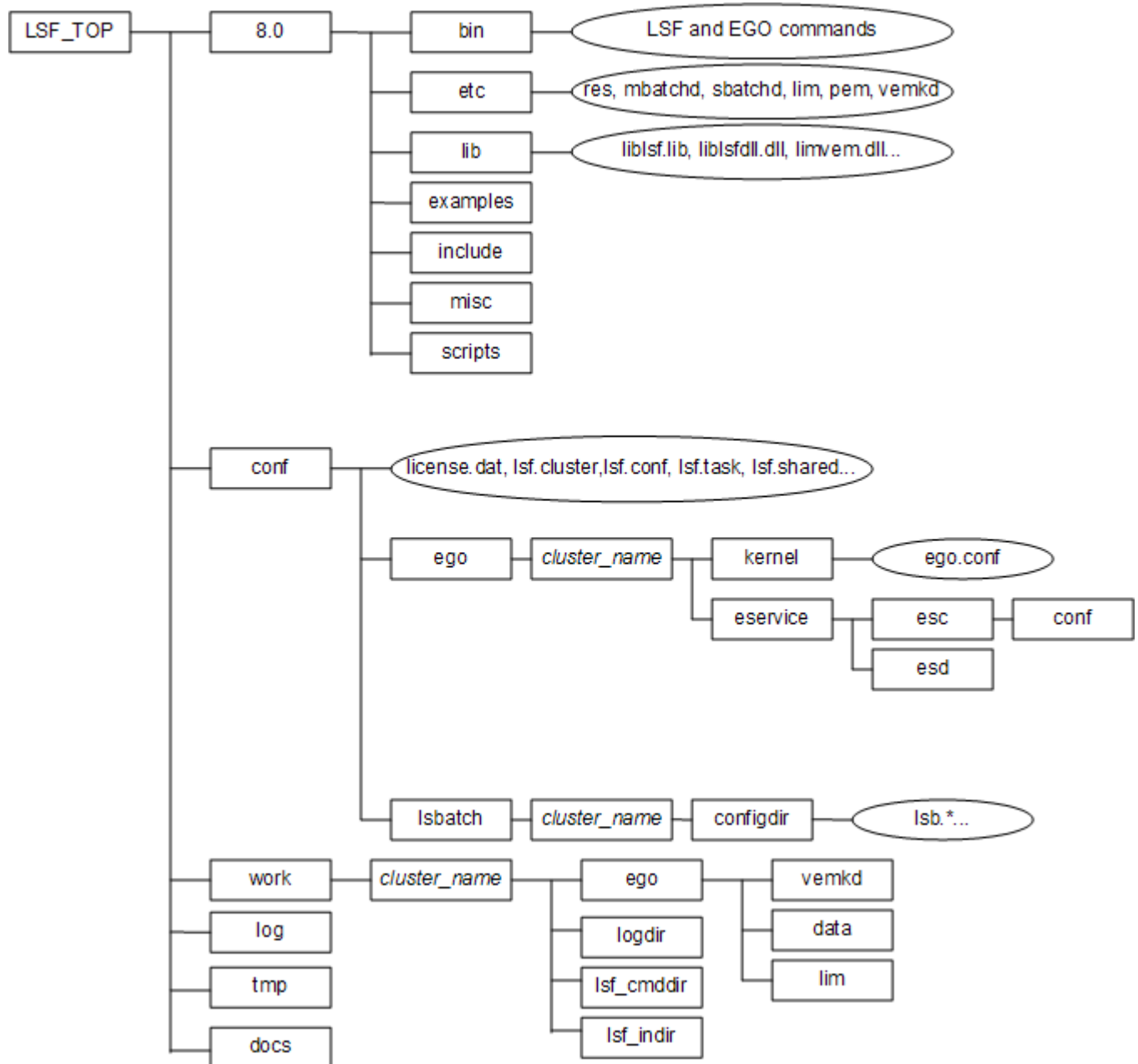
UNIX and Linux

The following figures show typical directory structures for a new UNIX or Linux installation with `lsfinstall`. Depending on which products you have installed and platforms you have selected, your directory structure may vary.



Microsoft Windows

The following diagram shows an example directory structure for a Windows installation.



Configure LSF and EGO

EGO configuration files for Platform LSF daemon management (res.xml and sbatchd.xml)

The following files are located in EGO_ESRVDIR/esc/conf/services/:

- `res.xml` —EGO service configuration file for `res`.
- `sbat chd.xml` —EGO service configuration file for `sbat chd`.

When LSF daemon control through EGO Service Controller is configured, `lsadm in` uses the reserved EGO service name `res` to control the LSF `res` daemon, and `badmi n` uses the reserved EGO service name `sbat chd` to control the LSF `sbat chd` daemon.

How to handle parameters in `lsf.conf` with corresponding parameters in `ego.conf`

When EGO is enabled, existing LSF parameters (parameter names beginning with `LSB_` or `LSF_`) that are set only in `lsf.conf` operate as usual because LSF daemons and commands read both `lsf.conf` and `ego.conf`.

Some existing LSF parameters have corresponding EGO parameter names in `ego.conf` (`LSF_CONFDIR/lsf.conf` is a separate file from `LSF_CONFDIR/ego/cluster_name/kernel/ego.conf`). You can keep your existing LSF parameters in `lsf.conf`, or you can set the corresponding EGO parameters in `ego.conf` that have not already been set in `lsf.conf`.

You cannot set LSF parameters in `ego.conf`, but you can set the following EGO parameters related to LIM, PIM, and ELIM in either `lsf.conf` or `ego.conf`:

- `EGO_DAEMONS_CPUS`
- `EGO_DEFINE_NCPUS`
- `EGO_SLAVE_CTRL_REMOTE_HOST`
- `EGO_WORKDIR`
- `EGO_PIM_SWAP_REPORT`

You cannot set any other EGO parameters (parameter names beginning with `EGO_`) in `lsf.conf`. If EGO is not enabled, you can only set these parameters in `lsf.conf`.

Note:

If you specify a parameter in `lsf.conf` and you also specify the corresponding parameter in `ego.conf`, the parameter value in `ego.conf` takes precedence over the conflicting parameter in `lsf.conf`.

If the parameter is not set in either `lsf.conf` or `ego.conf`, the default takes effect depends on whether EGO is enabled. If EGO is not enabled, then the LSF default takes effect. If EGO is enabled, the EGO default takes effect. In most cases, the default is the same.

Some parameters in `lsf.conf` do not have exactly the same behavior, valid values, syntax, or default value as the corresponding parameter in `ego.conf`, so in general, you should not set them in both files. If you need LSF parameters for backwards compatibility, you should set them only in `lsf.conf`.

If you have LSF 6.2 hosts in your cluster, they can only read `lsf.conf`, so you must set LSF parameters only in `lsf.conf`.

Platform LSF and Platform EGO corresponding parameters

The following table summarizes existing LSF parameters that have corresponding EGO parameter names. You must continue to set other LSF parameters in `lsf.conf`.

lsf.conf parameter	ego.conf parameter
LSF_API_CONNTIMEOUT	EGO_LIM_CONNTIMEOUT
LSF_API_RECVTIMEOUT	EGO_LIM_RECVTIMEOUT
LSF_CLUSTER_ID (Windows)	EGO_CLUSTER_ID (Windows)
LSF_CONF_RETRY_INT	EGO_CONF_RETRY_INT
LSF_CONF_RETRY_MAX	EGO_CONF_RETRY_MAX
LSF_DEBUG_LIM	EGO_DEBUG_LIM
LSF_DHPC_ENV	EGO_DHPC_ENV
LSF_DYNAMIC_HOST_TIMEOUT	EGO_DYNAMIC_HOST_TIMEOUT
LSF_DYNAMIC_HOST_WAIT_TIME	EGO_DYNAMIC_HOST_WAIT_TIME
LSF_ENABLE_DUALCORE	EGO_ENABLE_DUALCORE
LSF_GET_CONF	EGO_GET_CONF
LSF_GETCONF_MAX	EGO_GETCONF_MAX
LSF_LIM_DEBUG	EGO_LIM_DEBUG
LSF_LIM_PORT	EGO_LIM_PORT
LSF_LOCAL_RESOURCES	EGO_LOCAL_RESOURCES
LSF_LOG_MASK	EGO_LOG_MASK
LSF_MASTER_LIST	EGO_MASTER_LIST
LSF_PIM_INFODIR	EGO_PIM_INFODIR
LSF_PIM_SLEEPTIME	EGO_PIM_SLEEPTIME
LSF_PIM_SLEEPTIME_UPDATE	EGO_PIM_SLEEPTIME_UPDATE
LSF_RSH	EGO_RSH
LSF_STRIP_DOMAIN	EGO_STRIP_DOMAIN
LSF_TIME_LIM	EGO_TIME_LIM

Parameters that have changed in Platform LSF 8

The default for `LSF_LIM_PORT` has changed to accommodate EGO default port configuration. On EGO, default ports start with `lim` at 7869, and are numbered consecutively for `pem`, `vemkd`, and `egosc`.

This is different from previous LSF releases where the default `LSF_LIM_PORT` was 6879. `res`, `sbatchd`, and `mbatchd` continue to use the default pre-version 7 ports 6878, 6881, and 6882.

Upgrade installation preserves any existing port settings for `lim`, `res`, `sbatchd`, and `mbatchd`. EGO `pem`, `vemkd`, and `egosc` use default EGO ports starting at 7870, if they do not conflict with existing `lim`, `res`, `sbatchd`, and `mbatchd` ports.

EGO connection ports and base port

LSF and EGO require exclusive use of certain ports for communication. EGO uses the same four consecutive ports on every host in the cluster. The first of these is called the base port.

The default EGO base connection port is 7869. By default, EGO uses four consecutive ports starting from the base port. By default, EGO uses ports 7869-7872.

The ports can be customized by customizing the base port. For example, if the base port is 6880, EGO uses ports 6880-6883.

LSF and EGO needs the same ports on every host, so you must specify the same base port on every host.

Special resource groups for Platform LSF master hosts

By default, Platform LSF installation defines a special resource group named `ManagementHosts` for the Platform LSF master host. (In general, Platform LSF master hosts are dedicated hosts; the `ManagementHosts` EGO resource group serves this purpose.)

Platform LSF master hosts must not be subject to any lend, borrow, or reclaim policies. They must be exclusively owned by the Platform LSF consumer.

The default Platform EGO configuration is such that the `LSF_MASTER_LIST` hosts and the execution hosts are in different resource groups so that different resource plans can be applied to each group.

Manage LSF daemons through EGO

EGO daemons

Daemons in LSF_SERVERDIR	Description
vemkd	Started by lim on master host
pem	Started by lim on every host
egosc	Started by vemkd on master host

Platform LSF daemons

Daemons in LSF_SERVERDIR	Description
lim	lim runs on every host. On UNIX, lim is either started by lsadmin through rsh/ssh or started through rc file. On Windows, lim is started as a Windows service.
pim	Started by lim on every host
mbatchd	Started by sbatchd on master host
mbschd	Started by mbatchd on master host
sbatchd	Under OS startup mode, sbatchd is either started by lsadmin through rsh/ssh or started through rc file on UNIX. On Windows, sbatchd is started as a Windows service. Under EGO Service Controller mode, sbatchd is started by pem as an EGO service on every host.

Daemons in LSF_SERVERDIR	Description
res	<p>Under OS startup mode, res is either started by lsadmin through rsh/ssh or started through rc file on UNIX. On Windows, res is started as a Windows service.</p> <p>Under EGO Service Controller mode, res is started by pem as an EGO service on every host.</p>

Operating System daemon control

Operating system startup mode is the same as previous releases:

- On UNIX, administrators configure the autostart of sbatchd and res in the operating system (/etc/rc file or init tab) and use lsadmin and badmi n to start LSF daemons manually through rsh or ssh.
- On Windows, sbatchd and res are started as Windows services.

EGO Service Controller daemon control

Under EGO Service Control mode, administrators configure the EGO Service Controller to start res and sbatchd, and restart them if they fail.

You can still run lsadmin and badmi n to start LSF manually, but internally, lsadmin and badmi n communicates with the EGO Service Controller, which actually starts sbatchd and res as EGO services.

If EGO Service Controller management is configured and you run badmi n hshutdown and lsadmin resshutdown to manually shut down LSF, the LSF daemons are not restarted automatically by EGO. You must run lsadmin resstartup and badmi n hstartup to start the LSF daemons manually.

Permissions required for daemon control

To control all daemons in the cluster, you must

- Be logged on as root or as a user listed in the /etc/lsf.sudoers file. See the *LSF Configuration Reference* for configuration details of lsf.sudoers.
- Be able to run the rsh or ssh commands across all LSF hosts without having to enter a password. See your operating system documentation for information about configuring the rsh and ssh commands. The shell command specified by LSF_RSH in lsf.conf is used before rsh is tried.

Bypass EGO login at startup (lsf.sudoers)

You must be the LSF administrator (lsfadmin) or root to configure lsf.sudoers.

When LSF daemons control through EGO Service Controller is configured, users must have EGO credentials for EGO to start res and sbatchd services. By default, lsadmin and badmi n invoke the egosh *user login* command to prompt for the user name and password of the EGO administrator to get EGO credentials.

1. Configure lsf.sudoers to bypass EGO login to start res and sbatchd automatically.

Set the following parameters:

- LSF_EGO_ADMIN_USER—User name of the EGO administrator. The default administrator name is Admin.
- LSF_EGO_ADMIN_PASSWD—Password of the EGO administrator.

Administrative basics

See *Administering and Using Platform EGO* for detailed information about EGO administration.

Set the command-line environment

On Linux hosts, set the environment before you run any LSF or EGO commands. You need to do this once for each session you open. `root`, `lsfadmin`, and `egoadmin` accounts use LSF and EGO commands to configure and start the cluster.

You need to reset the environment if the environment changes during your session, for example, if you run `egoconfig mghost`, which changes the location of some configuration files.

- For `csh` or `tcsh`, use `csSRC.lsf`.
- For `sh`, `ksh`, or `bash`, use `profile.lsf`:

```
source LSF_TOP/conf/cshrc.lsf
```

```
. LSF_TOP/conf/profile.lsf
```

If Platform EGO is enabled in the LSF cluster (`LSF_ENABLE_EGO=Y` and `LSF_EGO_ENVDIR` are defined in `lsf.conf`), `csSRC.lsf` and `profile.lsf`, set the following environment variables:

- `EGO_BINDIR`
- `EGO_CONFDIR`
- `EGO_ESRVDIR`
- `EGO_LIBDIR`
- `EGO_LOCAL_CONFDIR`
- `EGO_SERVERDIR`
- `EGO_TOP`

See the *Platform EGO Reference* for more information about these variables.

See the *LSF Configuration Reference* for more information about `csSRC.lsf` and `profile.lsf`.

Logging and troubleshooting

Platform LSF log files

Platform LSF event and account log location

LSF uses directories for temporary work files, log files and transaction files and spooling.

LSF keeps track of all jobs in the system by maintaining a transaction log in the work subtree. The LSF log files are found in the directory `LSB_SHAREDIR/cluster_name/logdir`.

The following files maintain the state of the LSF system:

- `lsb.events` LSF uses the `lsb.events` file to keep track of the state of all jobs. Each job is a transaction from job submission to job completion. LSF system keeps track of everything associated with the job in the `lsb.events` file.
- `lsb.events.n` The events file is automatically trimmed and old job events are stored in `lsb.event.n` files. When `mbatchd` starts, it refers only to the `lsb.events` file, not the `lsb.events.n` files. The `bhist` command can refer to these files.

Platform LSF error log location

If the optional `LSF_LOGDIR` parameter is defined in `lsf.conf`, error messages from LSF servers are logged to files in this directory.

If `LSF_LOGDIR` is defined, but the daemons cannot write to files there, the error log files are created in `/tmp`.

If `LSF_LOGDIR` is not defined, errors are logged to the system error logs (`syslog`) using the `LOG_DAEMON` facility. `syslog` messages are highly configurable, and the default configuration varies widely from system to system. Start by looking for the file `/etc/syslog.conf`, and read the man pages for `syslog(3)` and `syslogd(1)`.

If the error log is managed by `syslog`, it is probably already being automatically cleared.

If LSF daemons cannot find `lsf.conf` when they start, they will not find the definition of `LSF_LOGDIR`. In this case, error messages go to `syslog`. If you cannot find any error messages in the log files, they are likely in the `syslog`.

Platform LSF daemon error logs

LSF log files are reopened each time a message is logged, so if you rename or remove a daemon log file, the daemons will automatically create a new log file.

The LSF daemons log messages when they detect problems or unusual situations.

The daemons can be configured to put these messages into files.

The error log file names for the LSF system daemons are:

- `res.log.host_name`
- `sbatchd.log.host_name`
- `mbatchd.log.host_name`
- `mbschd.log.host_name`

LSF daemons log error messages in different levels so that you can choose to log all messages, or only log messages that are deemed critical. Message logging for LSF daemons is controlled by the parameter `LSF_LOG_MASK` in `lsf.conf`. Possible values for this parameter can be any log priority symbol that is defined in `/usr/include/sys/syslog.h`. The default value for `LSF_LOG_MASK` is `LOG_WARNING`.

Platform LSF log directory permissions and ownership

Ensure that the permissions on the `LSF_LOGDIR` directory to be writable by `root`. The LSF administrator must own `LSF_LOGDIR`.

EGO log files

Log files contain important run-time information about the general health of EGO daemons, workload submissions, and other EGO system events. Log files are an essential troubleshooting tool during production and testing.

The naming convention for most EGO log files is the name of the daemon plus the host name the daemon is running on.

The following table outlines the daemons and their associated log file names. Log files on Windows hosts have a `.txt` extension.

Daemon	Log file name
ESC (EGO Service Controller)	<i>esc. log. hostname</i>
named	<i>named. log. hostname</i>
PEM (Process Execution Manager)	<i>pem. log. hostname</i>
VEMKD (Platform LSF Kernel Daemon)	<i>vemkd. log. hostname</i>
WSG (Web Service Gateway)	<i>wsg. log</i>

Most log entries are informational in nature. It is not uncommon to have a large (and growing) log file and still have a healthy cluster.

EGO log file locations

By default, most Platform LSF log files are found in *LSF_LOGDIR*.

- The service controller log files are found in *LSF_LOGDIR/ego/cluster_name/eservice/esc/log* (Linux) or *LSF_LOGDIR\ego\cluster_name\eservice\esc\log* (Windows).
- Web service gateway log files are found in
LSF_LOGDIR/ego/cluster_name/eservice/wsg/log (Linux)
LSF_LOGDIR\ego\cluster_name\eservice\wsg\log (Windows)
- The service directory log files, logged by BIND, are found in
LSF_LOGDIR/ego/cluster_name/eservice/esd/conf/named/namedb/named. log. hostname (Linux)
LSF_LOGDIR\ego\cluster_name\eservice\esd\conf\named\namedb\named. log. hostname (Windows)

EGO log entry format

Log file entries follow the format

date time_zone log_level [process_id:thread_id] action:description/message

where the date is expressed in YYYY-MM-DD hh-mm-ss.sss.

For example, 2006-03-14 11:02:44.000 Eastern Standard Time ERROR [2488:1036] vemkdexit: vemkd is halting.

EGO log classes

Every log entry belongs to a log class. You can use log class as a mechanism to filter log entries by area. Log classes in combination with log levels allow you to troubleshoot using log entries that only address, for example, configuration.

Log classes are adjusted at run time using `egosh debug`.

Valid logging classes are as follows:

Class	Description
LC_ALLOC	Logs messages related to the resource allocation engine

Class	Description
LC_AUTH	Logs messages related to users and authentication
LC_CLIENT	Logs messages related to clients
LC_COMM	Logs messages related to communications
LC_CONF	Logs messages related to configuration
LC_CONTAINER	Logs messages related to activities
LC_EVENT	Logs messages related to the event notification service
LC_MEM	Logs messages related to memory allocation
LC_PEM	Logs messages related to the process execution manager (pem)
LC_PERF	Logs messages related to performance
LC_QUERY	Logs messages related to client queries
LC_RECOVER	Logs messages related to recovery and data persistence
LC_RSRC	Logs messages related to resources, including host status changes
LC_SYS	Logs messages related to system calls
LC_TRACE	Logs the steps of the program

EGO log levels

There are nine log levels that allow administrators to control the level of event information that is logged.

When you are troubleshooting, increase the log level to obtain as much detailed information as you can. When you are finished troubleshooting, decrease the log level to prevent the log files from becoming too large.

Valid logging levels are as follows:

Number	Level	Description
0	LOG_EMERG	Log only those messages in which the system is unusable.
1	LOG_ALERT	Log only those messages for which action must be taken immediately.
2	LOG_CRIT	Log only those messages that are critical.
3	LOG_ERR	Log only those messages that indicate error conditions.
4	LOG_WARNING	Log only those messages that are warnings or more serious messages. This is the default level of debug information.
5	LOG_NOTICE	Log those messages that indicate normal but significant conditions or warnings and more serious messages.
6	LOG_INFO	Log all informational messages and more serious messages.
7	LOG_DEBUG	Log all debug-level messages.

Number	Level	Description
8	LOG_TRACE	Log all available messages.

EGO log level and class information retrieved from configuration files

When EGO is enabled, the pem and vemkd daemons read ego.conf to retrieve the following information (as corresponds to the particular daemon):

- EGO_LOG_MASK: The log level used to determine the amount of detail logged.
- EGO_DEBUG_PEM: The log class setting for pem.
- EGO_DEBUG_VEMKD: The log class setting for vemkd.

The wsg daemon reads wsg.conf to retrieve the following information:

- WSG_PORT: The port on which the Web service gateway (WebServiceGateway) should run
- WSG_SSL: Whether the daemon should use Secure Socket Layer (SSL) for communication.
- WSG_DEBUG_DETAIL: The log level used to determine the amount of detail logged for debugging purposes.
- WSG_LOGDIR: The directory location where wsg.log files are written.

The service director daemon (named) reads named.conf to retrieve the following information:

- logging, severity: The configured severity log class controlling the level of event information that is logged (critical, error, warning, notice, info, debug, or dynamic). In the case of a log class set to debug, a log level is required to determine the amount of detail logged for debugging purposes.

Why do log files grow so quickly?

Every time an EGO system event occurs, a log file entry is added to a log file. Most entries are informational in nature, except when there is an error condition. If your log levels provide entries for all information (for example, if you have set them to LOG_DEBUG), the files will grow quickly.

Suggested settings:

- During regular EGO operation, set your log levels to LOG_WARNING. With this setting, critical errors are logged but informational entries are not, keeping the log file size to a minimum.
- For troubleshooting purposes, set your log level to LOG_DEBUG. Because of the quantity of messages you will receive when subscribed to this log level, change the level back to LOG_WARNING as soon as you are finished troubleshooting.

Note:

If your log files are too long, you can always rename them for archive purposes. New, fresh log files will then be created and will log all new events.

How often should I maintain log files?

The growth rate of the log files is dependent on the log level and the complexity of your cluster. If you have a large cluster, daily log file maintenance may be required.

We recommend using a log file rotation utility to do unattended maintenance of your log files. Failure to do timely maintenance could result in a full file system which hinders system performance and operation.

Troubleshoot using multiple EGO log files

EGO log file locations and content

If a service does not start as expected, open the appropriate service log file and review the run-time information contained within it to discover the problem. Look for relevant entries such as insufficient disk space, lack of memory, or network problems that result in unavailable hosts.

Log file	Default location	What it contains
esc.log	Linux: LSF_LOGDIR/ego/ <i>cluster_name</i> /eservice/esc/log/esc.log, <i>hostname</i> Windows: LSF_LOGDIR\ego\ <i>cluster_name</i> \eservice\esc\log\esc.log, <i>hostname</i>	Logs service failures and service instance restarts based on availability plans.
named.log	Linux: LSF_LOGDIR/ego/ <i>cluster_name</i> /eservice/esd/conf/named/namedb/named.log, <i>hostname</i> Windows: LSF_LOGDIR\ego\ <i>cluster_name</i> \eservice\esd\conf\named\namedb\named.log, <i>hostname</i>	Logs information gathered during the updating and querying of service instance location; logged by BIND, a DNS server.
pem.log	Linux: LSF_LOGDIR/pem.log, <i>hostname</i> Windows: LSF_LOGDIR\pem.log, <i>hostname</i>	Logs remote operations (start, stop, control activities, failures). Logs tracked results for resource utilization of all processes associated with the host, and information for accounting or chargeback.
vemkd.log	Linux: LSF_LOGDIR/vemkd.log, <i>hostname</i> Windows: LSF_LOGDIR\vemkd.log, <i>hostname</i>	Logs aggregated host information about the state of individual resources, status of allocation requests, consumer hierarchy, resources assignment to consumers, and started operating system-level process.
wsg.log	Linux: LSF_LOGDIR/ego/ <i>cluster_name</i> /eservice/wsg/log/wsg.log, <i>hostname</i> Windows: LSF_LOGDIR\ego\ <i>cluster_name</i> \eservice\wsg\log\wsg.log, <i>hostname</i>	Logs service failures surrounding web services interfaces for web service clients (applications).

Match service error messages and corresponding log files

If you receive this message...	This may be the problem...	Review this log file
failed to create vem working directory	Cannot create work directory during startup	vemkd
failed to open lock file	Cannot get lock file during startup	vemkd

If you receive this message...	This may be the problem...	Review this log file
failed to open host event file	Cannot recover during startup because cannot open event file	vemkd
lim port is not defined	EGO_LIM_PORT in ego.conf is not defined	lim
master candidate can not set GET_CONF=lim	Wrong parameter defined for master candidate host (for example, EGO_GET_CONF=LIM)	lim
there is no valid host in EGO_MASTER_LIST	No valid host in master list	lim
ls_getmyhostname fails	Cannot get local host name during startup	pem
temp directory (%s) not exist or not accessible, exit	Tmp directory does not exist	pem
incorrect EGO_PEM_PORT value %s, exit	EGO_PEM_PORT is a negative number	pem
chdir(%s) fails	Tmp directory does not exist	esc
cannot initialize the listening TCP port %d	Socket error	esc
cannot log on	Log on to vemkd failed	esc
vem_register: error in invoking vem_register function	VEM service registration failed	wsg
you are not authorized to unregister a service	Either you are not authorized to unregister a service, or there is no registry client	wsg
request has invalid signature: TSIG service.ego: tsig verify failure (BADTIME)	Resource record updating failed	named

Frequently asked questions

Question *Does LSF 8 on EGO support a grace period when reclamation is configured in the resource plan?*

Answer No. Resources are immediately reclaimed even if you set a resource reclaim grace period.

Question *Does LSF 8 on EGO support upgrade of the master host only?*

Answer Yes

Question *Under EGO Service Controller daemon management mode on Windows, does PEM start sbatchd and res directly or does it ask Windows to start sbatchd and res as Windows Services?*

- Answer On Windows, LSF still installs sbatchd and res as Windows services. If EGO Service Controller daemon control is selected during installation, the Windows service will be set up as Manual. PEM will start up the sbatchd and res directly, not as Windows Services.
- Question *What's the benefit of LSF daemon management through the EGO Service Controller?*
- Answer EGO Service Controller provides High Availability services to sbatchd and res, and faster cluster startup than startup with lsadmin and badmin.
- Question *How does the hostsetup script work in LSF 8?*
- Answer LSF 8 hostset up script functions essentially the same as previous versions. It sets up a host to use the LSF cluster and configures LSF daemons to start automatically. In LSF 8, running `hostsetup --top=/path --boot="y"` will check the EGO service definition files `sbatchd.xml` and `res.xml`. If `res` and `sbatchd` startup is set to "Automatic", the `host rc` setting will only start `lim`. If set to "Manual", the `host rc` setting will start `lim`, `sbatchd`, and `res` as in previous versions.
- Question *Is non-shared mixed cluster installation supported, for example, adding UNIX hosts to a Windows cluster, or adding Windows hosts to a UNIX cluster?*
- Answer In LSF 8, non-shared installation is supported. For example, to add a UNIX host to a Windows cluster, set up the Windows cluster first, then run `lsfinstall -s -f slave.config`. In `slave.config`, put the Windows hosts in `LSF_MASTER_LIST`. After startup, the UNIX host will become an LSF host. Adding a Windows host is even simpler. Run the Windows installer, enter the current UNIX master host name. After installation, all daemons will automatically start and the host will join the cluster.
- Question *As EGO and LSF share base configuration files, how are other resources handled in EGO in addition to hosts and slots?*
- Answer Same as previous releases. LSF 8 `mbatchd` still communicates with LIM to get available resources. By default, LSF can schedule jobs to make use of all resources started in cluster. If EGO-enabled SLA scheduling is configured, LSF only schedules jobs to use resources on hosts allocated by EGO.
- Question *How about compatibility for external scripts and resources like elim, melim, esub and others?*
- Answer LSF 8 supports full compatibility for these external executables. `elim.xxx` is started under `LSF_SERVERDIR` as usual. By default, LIM is located under `LSF_SERVERDIR`.
- Question *Can Platform LSF MultiCluster share one EGO base?*
- Answer No, each LSF cluster must run on top of one EGO cluster.
- Question *Can EGO consumer policies replace MultiCluster lease mode?*
- Answer Conceptually, both define resource borrowing and lending policies. However, current EGO consumer policies can only work with slot resources within one EGO cluster. MultiCluster lease mode supports other load indices and external resources between multiple clusters. If you are using MultiCluster lease mode to share only slot resources between clusters, and you are able to merge those clusters into a single cluster, you should be able to use EGO consumer policy and submit jobs to EGO-enabled SLA scheduling to achieve the same goal.

Index

- ok host status
 - lsload command 44
 - status load index 134
- R res_req command argument 422
- ! (NOT) operator
 - job dependencies 590
- .cshrc file and lscsh 829
- .rhosts file
 - disadvantages 294
 - file transfer with lsrcp 332
 - host authentication 293
- @ (at sign) in lscsh 830
- /etc/hosts file
 - example host entries 73
 - host naming 70
 - name lookup 71
 - troubleshooting 352
- /etc/hosts.equiv file
 - host authentication 293
 - troubleshooting 352
 - using rcp 332
- /etc/shells file, and lscsh 829
- /etc/syslog.conf file 339, 870
- /usr/include/sys/syslog.h file 338
- && (AND) operator
 - job dependencies 590
- %I substitution string in job arrays 661
- %J substitution string in job arrays 661
- %USRCMD string in job starters 760
- || (OR) operator
 - job dependencies 590
- ~ (tilde)
 - not operator
 - host partition fairshare 466
 - host-based resources 149
- .cshrc file and lscsh 829
- /etc/services file
 - adding LSF entries to 68

- \$HOME/.rhosts file
 - disadvantages 294
 - file transfer with lsrcp command 332
 - host authentication 293
- /tmp directory
 - default LSF_LOGDIR 339, 870
- /tmp_mnt directory 350
- %USRCMD string in job starters 760

A

- abnormal job termination 91
- ABS_RUNLIMIT parameter in lsb.params 728
- absolute job priority scheduling
 - admin value 600
 - description 597
 - LSF feature interactions 604
 - modifying calculated APS value 600
 - priority factors 597
 - resizable jobs 605
- absolute run time limit 728
- access permissions for interactive tasks 808
- ACCESS_CONTROL parameter in lsb.serviceclasses 176
- accounting information for advance reservations 394
- adaptive dispatch. *See* chunk jobs
- admin APS value 600
- administrator comments
 - logging in lsb.events
 - for host open and close 53
 - for mbatchd restart 22
 - for queue events 122
- administrators
 - cluster administrator description 19
 - primary LSF administrator 19
- ADMINISTRATORS
 - lsb.queues file 599
- ADMINISTRATORS parameter in lsb.queues 176
- ADMINISTRATORS parameter in lsf.cluster.cluster_name 19
- advance reservation

- accounting information 394
- adding and removing 383
- commands 383
- configuring user policies 381
- description 380, 381
- preemption 396
- reservation ID 394
- resource-based SLA scheduling 508
- schmod_advrsv plugin 381
- submitting jobs 395
- user policies 381
- viewing 391
- viewing accounting information 394
- weekly planner (brsvs -p) 392
- advance reservations
 - compute units 398
 - resizable jobs 397
- advanced dependency conditions 593
- advanced reservation
 - compute units 83
- AFS (Andrew File System)
 - overview 328
- aliases
 - for resource names 428
 - host name ranges 70
 - using as host names 70
- AND operator (&&)
 - job dependencies 590
- Andrew File System. *See* AFS
- application profiles
 - adding and removing 540
 - configuring
 - for chunk jobs 657
 - controlling jobs 543
 - default application profile 541
 - description 540
 - job success exit values 541
 - modifying jobs (bmod -app) 543
 - submitting jobs (bsub -app) 543
 - viewing
 - detailed information (bapp -l) 545
 - jobs (bjobs -app) 546
 - summary information (bacct -app) 546
 - summary information (bapp) 545
- application-level job checkpoint and restart
 - description 629
- application-specific job checkpoint and restart
 - configuring 636
 - enabling 636
- APS_PRIORITY parameter in lsb.queues 599
- APS. *See* absolute job priority scheduling
- architecture
 - EGO 860
- architecture, viewing for hosts 49
- arguments
 - passed to the LSF event program 280
 - passing to job arrays 662
- arrays
 - chunking 665
- at sign (@) in lscsh 830
- augmentstarter job starter 762
- authentication
 - security 292
- automatic
 - duplicate event logging 342
 - event log archiving 342
 - job requeue 608
 - job rerun
 - description 612
 - resizable jobs 612
 - priority escalation 596
 - remote execution in lscsh 828
 - time-based configuration 373
- automatic job priority escalation
 - resizable jobs 596
- automount command
 - NFS (Network File System) 328, 350
- automount option
 - /net 330
- autoresizable jobs
 - checkpoint and restart 623
- available
 - meaning 135

B

- bacct -app 546
- bacct -U
 - advance reservations 394
- bacct command
 - CPU time display 731
 - SLA scheduling 531
- BACKFILL parameter in lsb.queues 703
- backfill scheduling
 - default run limit 723
 - description 701

- interruptible backfill 704
 - resizable jobs 702
 - resource allocation limits 557
- background jobs, bringing to foreground 831
- badmin command
 - hopen 53
 - hrestart 20
 - hshutdown 20
 - hstartup 20
 - logging administrator comments
 - for host open and close 53
 - for mbatchd restart 22
 - for queue events 122
 - LSF event logs 340
 - mbdrestart 20, 25
 - qact 122
 - qclose 122
 - qinact 122
 - qopen 122
- balance keyword
 - cu string 684
- bapp 545
- batch jobs
 - accessing files 328, 330
 - allocating processors 676
 - email about jobs
 - disabling 320
 - options 320
 - file access 330
 - input and output 320
 - killing 99
 - requeue 608
 - rerunning and restarting 612
 - signalling 99
- batch log files. *See* log files
- bbot command 95
 - user-assigned job priority 595
- bconf
 - about 26
 - history files 30
- benchmarks for setting CPU factors 84
- Berkeley Internet Name Domain (BIND)
 - host naming 70
- between-host user account mapping
 - description 186
 - local user account mapping
 - configuring 188
 - example 189
 - scope 187
 - Windows workgroup
 - configuring 188
 - Windows workgroup account mapping
 - example 189
- bgadd command
 - job group limits 106
- bgdel command 109
- bgmod command
 - job group limits 110
- bhist -l 109
- bhist command
 - job exit codes 347
 - LSF event logs 340
- bhosts -l 44
- bhosts -x
 - viewing host exception status 51
- bhosts command
 - checking time-based configuration 375
- BIND (Berkeley Internet Name Domain)
 - host naming 70
- binding processors
 - resizable jobs 713
- bjgroup command 107
 - SLA scheduling 527
- bjobs -app 546
- bjobs -aps
 - order of absolute job priority scheduling 602
- bjobs -g 107
- bjobs -x
 - viewing job exception status 94
- bjobs command
 - reservation ID for advance reservation 394
 - SLA scheduling 531
- bkill -app 543
- bkill -g 109
- bkill command
 - kill the Session Scheduler session 844
- black hole hosts 86, 112
- bladmin chkconfig command
 - checking time-based configuration 375
- blimits -c command
 - checking time-based configuration 375
- blimits command 568
- blinfo command
 - checking time-based configuration 375
- blstat command
 - checking time-based configuration 375

- bmod
 - absolute job priority scheduling 600
- bmod -app 543
- bmod -g 108
- bmod -is 323
- Boolean resources 131
- bparams command
 - checking time-based configuration 375
 - viewing configuration parameters 14
- bqueues -l
 - absolute job priority scheduling 603
- bqueues command
 - checking time-based configuration 375
 - cross-queue fairshare information 468
- bresize cancel command 650
- bresize release command 650
- bresources command
 - checking time-based configuration 375
- brestart command
 - resizable jobs 623
- bresume -app 543
- bresume -g 108
- brsvadd -b
 - specifying begin times 384
- brsvadd -e
 - specifying end times 384
- brsvadd -m
 - specifying a host list 384
- brsvadd -R
 - specifying a resource requirement string 384
- brsvadd -t
 - specifying recurring reservations 385
- brsvadd command 383
- brsvdel command 391
- brsvmod command 387
- brsvs command 391
- brun command
 - advance reservation 397
 - forcing a job to run 97
- bsla command 529
- bstop -app 543
- bstop -g 108
- bsub -app 543
- bsub -is 323
- bsub -sla 529
- bsub -Zs 325
- bsub command
 - email job notification 320

- input and output 320
- remote file access 330
 - submitting a job
 - associated to a job group 106
 - associated to a service class 529
- bswitch command
 - resizable jobs 653
- btopy command
 - user-assigned job priority 595
- built-in load indices
 - overriding 150
- built-in resources 131
- busers command
 - checking time-based configuration 375
- busy host status
 - lsload command 44
 - status load index 135
- busy thresholds, tuning 282

C

- calculating license key check sums (lmcksum) 224
- calculation of required licenses 216
- ceiling resource usage limit 722
- chargeback fairshare 489
- check ssched parameters 844
- check_license script, for counted software licenses 151
- checking
 - license server status (lmstat) 223
- checkout of licenses 216
- checkpoint and restart
 - description 628
 - resizable jobs 623
- checkpointable jobs
 - chunk jobs 659
- checksum
 - calculating for license key (lmcksum) 224
- chsh and lscsh 829
- chunk jobs
 - absolute job priority scheduling 605
 - checkpointing 658, 659
 - CHUNK_JOB_DURATION parameter in lsb.params 657
 - configuring application profile for 657
 - configuring queue for 656
 - description 655
 - fairshare scheduling 659
 - job controls 658
 - killing 658

- limitations on queues 656
 - migrating 658
 - modifying 659
 - rerunnable 659
 - resizable jobs 653
 - resource usage limits 719
 - resource-based SLA scheduling 508
 - resuming 658
 - submitting and controlling 657
 - time-based SLA scheduling 522
 - WAIT status and pending reason 658
- CHUNK_JOB_DURATION
 - parameter in lsb.params 657
- chunking
 - job array 665
- CLEAN_PERIOD parameter in lsb.params 99
- closed host status 44
 - bhosts -l 44, 49
 - bhosts command 44
- cluster administrators
 - description 19
- clusters
 - configuration file quick reference 24
 - protecting with strict checking 63
 - reconfiguring
 - commands 24
 - how reconfiguration affects licenses 25
- command file spooling
 - See also* job file spooling
 - default directory 324
 - description 323
 - JOB_SPOOL_DIR parameter in lsb.params 323
- commands
 - built-in 831
 - checking time-based configuration 375
 - FlexNet utilities 223
 - job starters 756
 - lmcksum (FlexNet) 224
 - lmdown (FlexNet) 224
 - lmhostid (FlexNet) 224
 - lmremove (FlexNet) 224
 - lmreread (FlexNet) 224
 - lmstat (FlexNet)
 - description 224
 - displaying license server status 223
 - using 223
 - lmver (FlexNet) 224
 - lshosts -l 227
 - using in job control actions 766
- components
 - EGO 860
- compound resource requirements
 - multi-level 424
 - overview 416
 - syntax 423
- Comprehensive System Accounting (IRIX CSA)
 - configuring 336
- compute unit resource allocation 422
- compute units
 - advanced reservation 83
 - configuring external compute units 83
 - cu string
 - syntax 684
 - exclusive 687
 - external 83
 - host level job allocation 688
 - reservation 688
 - resource requirements 449
 - resource-based SLA scheduling 508
- concurrent threads 729
- CONDENSE keyword in lsb.hosts 79, 83
- CONDENSE_PENDING_REASONS parameter in lsb.params 264
- condensed host groups
 - defining 79, 83
 - viewing 48
- condensed notation
 - host names 77
- configuration
 - adding and removing
 - application profiles 540
 - queues 124
 - application profiles
 - job success exit values 541
 - commands for checking 375
 - default application profile 541
 - removing
 - hosts 57
 - tuning
 - busy thresholds 282
 - LIM policies 282
 - load indices 282
 - load thresholds 283
 - mbatchd on UNIX 286
 - run windows 282
 - viewing

- errors 25
- configuration files
 - location 174
 - reconfiguration quick reference 24
- configuration parameters. *See* individual parameter names
- CONSUMABLE
 - lsf.shared file 147
- consumers
 - about 861
- CONTROL_ACTION parameter in lsf.serviceclasses 528
- core file size limit 725
- CORELIMIT parameter in lsf.queues 725
- counted software licenses
 - configuring 151
 - description 151
- CPU
 - factors
 - static resource 139
 - time normalization 731
 - tuning in lsf.shared 84
 - limits
 - per job 725
 - per process 725
 - normalization 731
 - run queue length, description 795
 - time
 - cumulative and decayed 458
 - in dynamic user priority calculation 458
 - time limit
 - job-level resource limit 725
 - tuning CPU factors in lsf.shared 84
 - utilization, ut load index 135
 - viewing run queue length 84
- CPU factor
 - non-normalized run time limit 728
- CPU factor (cpuf) static resource 138
- CPU time
 - idle job exceptions 112, 127
- CPU time normalization 725
- CPU_TIME_FACTOR parameter in lsf.params
 - fairshare dynamic user priority 458
- cpuf static resource 139
- CPULIMIT parameter in lsf.queues 725
- Cray
 - UNICOS accounting log files 336
- cross-cluster user account mapping
 - configuring 194
 - description 192

- enabling 194
- scope 187, 192
 - system level
 - configuring 194
 - example 195
 - user level
 - configuring 194
 - examples 195
- cross-queue fairshare 468
- CSA (IRIX Comprehensive System Accounting)
 - configuring 336
- cu resource requirement string
 - resizable jobs 450
- cu string 449
 - keyword balance 684
 - keyword excl 685
 - keyword maxcus 684
 - keyword pref 684
 - keyword type 684
 - syntax 684
- cumulative CPU time 458
- custom event handlers 278
- custom file transfer
 - configuring 332
- custom resources
 - adding 146
 - configuring 146
 - description 146
 - resource types 131

D

- DAEMON line
 - license.dat file 215
- DAEMON line, editing 217
- daemons
 - commands 20
 - debug commands 363
 - error logs 338, 870
 - restarting
 - mbatchd 22
 - sbatchd 21
 - shutting down
 - mbatchd 22
 - sbatchd 21
 - TCP service ports 68
 - ypbind 71
- data segment size limit 726
- DATALIMIT parameter in lsf.queues 726

- date of expiry (demo) 212
- DCE/DFS (Distributed File System)
 - overview 328
- deadline constraint scheduling
 - description 378
 - resizable jobs 378
- deadlock, avoiding signal and job action 767
- debug information
 - logging classes 871
 - logging levels 872
- debug level
 - commands for daemons 363
 - setting temporarily 363
- debug log classes
 - description 871
- debug log levels
 - description 872
- decayed
 - CPU time 458
 - run time 460, 461
- dedicated resources. *See* exclusive resources
- default
 - input file spooling 324
 - job control actions 764
 - JOB_SPOOL_DIR 324
 - LSF log file location 336, 869
 - LSF_LOGDIR 339, 870
 - output file spooling 324
 - resource usage limits 722
 - run limit
 - backfill scheduling 723
 - UNIX directory structure 17
 - Windows directory structure 18
- DEFAULT
 - model or type with lshosts command 355
- default normalization host 725
- default user group, fairshare scheduling 467
- DEFAULT_APPLICATION parameter in lsb.params 541
- DEFAULT_HOST_SPEC parameter
 - in lsb.params 725
 - in lsb.queues 731
- DEFAULT_JOBGROUP parameter in lsb.params 103
- DEFAULT_USER_GROUP in lsb.params 467
- defined keyword 430
- definitions
 - EGO 859
- delayed SLA scheduling goals
 - control action 528
- deletion
 - automatic job group cleanup 110
- demand
 - defining in SDK 860
- demo license 212
- dependency conditions
 - job arrays
 - operators 663
 - relational operators 591
- dependency conditions. *See* job dependency conditions
- dependency expressions
 - multiple conditions 590
- DFS (Distributed File System). *See* DCE/DFS 328
- directories
 - default UNIX directory structure 17
 - default Windows directory structure 18
 - log
 - permissions and ownership 336, 870
 - remote access 330
- directory for license (demo) 217
- directory for license (permanent) 218
- disks
 - I/O rate 136
- dispatch order, fairshare 465
- dispatch windows
 - description 376
 - hosts 53
 - queues 123
 - tuning for LIM 282
- DISPATCH_WINDOW
 - queues 123
- dispatch, adaptive. *See* chunk jobs
- displaying
 - FlexNet version information (lmver) 224
 - hardware host ID (lmhostid) 224
 - license server status (lmstat) 223, 224
 - licensed products 227
- distributed license server hosts 225
- distribution (partial licensing) 227
- distribution of licenses 216
- Domain Name Service (DNS)
 - host naming 70
- done job dependency condition 591
- DONE job state
 - description 90
- done jobs, viewing 90
- dual-stack hosts
 - defining official host name 73

- dns configuration 74
- duplicate event logging 342
 - after network partitioning 341
 - automatic 342
 - description 341
 - mbatchd restart with MAX_INFO_DIRS 266
- dynamic
 - hosts, protecting with strict checking 63
 - resources 131
 - user priority
 - description 457
 - formula 457
- dynamic priority
 - fairshare adjustment 459
 - memory based 459

E

- eadmin script
 - default exception actions 113
 - host and job exception handling 112
- EADMIN_TRIGGER_DURATION parameter in lsb.params 128
- echkpt
 - configuring 636
 - enabling 636
 - naming convention 632
 - syntax 633
- eexec
 - configuring 775, 785
 - definition 772
 - enabling 775, 785
 - example of monitoring execution environment 783
 - specifying a user account 785
 - typical uses 773
- effective run queue length
 - built-in resources 135
 - description 795
 - tuning LIM 284
- EGO
 - components 860
 - grace period
 - resources 875
 - how it works 860
 - what it is 859
- EGO administrator login bypass
 - enabling 38, 39
- EGO_LOG_MASK parameter in ego.conf 338
- EGO-enabled SLAs

- resource-based SLA scheduling 516
- ego.conf file
 - EGO_LOG_MASK parameter 338
 - managing error logs 338
- egroup
 - configuring 182
 - creating 182
 - description 180
 - enabling 182
 - scope 181
- elim
 - configuring 159
 - creating 161
 - description 156
 - enabling 159
 - example 163
 - overriding a built-in load index 163
 - scope 158
- email
 - disabling batch job notification 320
 - job options 320
 - limiting the size of job email 321
- embedded submission options for interactive jobs 802
- ENABLE_EXIT_RATE_PER_SLOT parameter in lsb.params 115
- ENABLE_ONE_UG_LIMITS
 - limits and user groups 559
- encryption
 - esub 780
 - X-Window 781
- ended job dependency condition 591
- ENFORCE_UG_TREE parameter
 - lsb.params 176
- environment
 - setting 869
- environment variables. *See* individual environment variable names
- equal share fairshare 489
- erestart
 - configuring 636
 - enabling 636
 - naming convention 632
 - syntax 633
- error logs
 - EGO_LOG_MASK parameter 338
 - log directory
 - LSF_LOGDIR 339, 870
 - log files 338, 870

- LSF daemons 338, 870
- LSF_LOG_MASK parameter 338
- managing log files 338
- on UNIX and Windows 339, 870
- errors
 - viewing in reconfiguration 25
- esub
 - configuring 775, 785
 - configuring a mandatory esub 785
 - definition 770
 - enabling 775, 785
 - example of changing job parameters 783
 - example of validating job parameters 783
 - naming convention 775
 - order in which multiple esubs run 772
 - typical uses 771
- evaluation license 212
- event generation 278
- event log archiving
 - automatic 342
- event log replication. *See* duplicate event logging
- event logging
 - mbatchd restart with MAX_INFO_DIRS 266
- event logs
 - automatic archiving 342
 - configuring duplicate logging 342
 - duplicate logging 342
 - logging administrator comments
 - for host open and close 53
 - for mbatchd restart 22
 - for queue events 122
 - LSF Batch log file in lsb.events file 340
 - update interval 342
- Event Viewer, Windows 278
- EVENT_UPDATE_INTERVAL in lsb.params 342
- events
 - custom programs to handle 278
 - generated by LSF 279
- example.services file 68
- examples
 - /etc/hosts file entries 73
- exception handling
 - configuring host exceptions 86
 - configuring in queues 127
- exception status
 - for hosts
 - viewing with bhosts 51
 - for jobs
 - viewing with bjobs 94
 - viewing with bqueues 120
- excl keyword
 - cu string 685
- exclusive jobs
 - requeue 610
 - resource-based SLA scheduling 508
- EXCLUSIVE parameter
 - in lsb.queues file 81
- exclusive resources host-based resources
 - exclusive resources 431
- exclusive scheduling
 - resizable jobs 534
- execution
 - forcing for jobs 97
 - priority 138
- execution host
 - mandatory for parallel jobs 681
- exit codes
 - job success exit values 541
 - returned by jobs 347
- exit dependency condition
 - relational operators 591
- exit job dependency condition 591
- EXIT job state
 - abnormal job termination 91
- exit rate for jobs 86, 112
- EXIT_RATE
 - bhosts -l 51
- EXIT_RATE parameter in lsb.hosts 86
- expiry date (demo) 212
- expiry time for mbatchd 287
- external
 - job dependency condition 592
- external authentication
 - configuration of 314
 - configuring 314
 - daemon authentication
 - enabling 314
 - daemon credentials
 - description 312
 - description 312
 - eauth user name
 - configuration of 316
 - enabling 314
 - encryption key
 - configuration of 316
 - host credentials

- description 312
- Kerberos
 - configuration of 317
 - eauth user name
 - configuration of 317
 - enabling 317
- Kerberos authentication
 - configuration 316
 - description 312
- Kerberos daemon authentication
 - enabling 317
 - non-Solaris 317
 - Solaris 317
- scope 313
 - user credentials
 - description 312
- external encryption key
 - configuring 316
- external host and user groups
 - configuring 182
 - defining 182
 - description 180
 - egroup
 - creating 182
 - enabling 182
 - scope 181
- external host groups
 - egroup
 - creating 182
- external load indices
 - behavior 164
 - benefits 156
 - commands 168
 - configuration to modify 167
 - configuring 159
 - description 156
 - elim
 - creating 161
 - example 163
 - host locations
 - environment variables 165
 - resource mapping 164
 - multiple executables 164
 - overriding a built-in load index 163
 - enabling 159
 - resource mapping 161
 - scope 158
- external resource

- defining 159
 - defining a dynamic resource 159
- external resources 131
- external user groups
 - egroup
 - creating 182

F

- factor grace period
 - absolute job priority scheduling 598
- factor limit
 - absolute job priority scheduling 598
- fairshare adjustment plugin 459
- FAIRSHARE parameter in lsb.queues 469
- fairshare scheduling
 - absolute job priority scheduling 604
 - across queues 468
 - chargeback 489
 - chunk jobs 659
 - default user group 467
 - defining policies that apply to several queues 468
 - description 454
 - dynamic user priority
 - description 457
 - formula 457
 - equal share 489
 - global 489
 - hierarchical share tree 473
 - overview 453
 - parallel jobs 710
 - policies 454
 - priority user 490
 - resizable jobs 492
 - resource usage measurement 457
 - static priority 490
 - user share assignment 455
 - viewing cross-queue fairshare information 468
- FAIRSHARE_QUEUES parameter
 - in bqueues 468
 - in lsb.queues
 - OBSOLETE 601
- fast job dispatching 262
- fault tolerance
 - non-shared file systems 329
- FEATURE line
 - license.dat file (demo) 214
 - license.dat file (permanent) 215
- features (LSF) 221

- file (for permanent license) 215
- file access, interactive tasks 808
- file preparation, job arrays 661
- file size usage limit 726
- file spooling. *See* command file spooling, job file spooling 323
- file systems
 - AFS (Andrew File System) 328
 - DCE/DFS (Distributed File System) 328
 - NFS (Network File System) 328
 - supported by LSF 328
- file transfer
 - lsr command 332
- FILELIMIT parameter in lsbf.queries 726
- files
 - /etc/hosts
 - example host entries 73
 - host naming 70
 - name lookup 71
 - /etc/services
 - adding LSF entries to 68
 - automatic time-based configuration 373
 - copying across hosts 809
 - enabling utmp registration 804
 - hosts
 - configuring 72
 - if-else constructs 373
 - license.dat
 - location 219
 - lsbf.params
 - CHUNK_JOB_DURATION parameter 657
 - lsbf.cluster.cluster_name
 - FLOAT_CLIENTS_ADDR_RANGE parameter 230
 - lsbf.conf
 - configuring TCP service ports 68
 - daemon service ports 68
 - lsbf.shared
 - adding a custom host types and models 67
 - redirecting 792
 - redirecting stdout and stderr 809
 - resolv.conf 71
 - spooling command and job files 801
- finger command in lscsh 829
- first execution host
 - parallel jobs 681
 - resizable jobs 681
- Flexera Software 216

- FlexNet 224
 - calculating license key check sums (lmcksum) 224
 - displaying
 - hardware host ID (lmhostid) 224
 - license server status (lmstat) 224
 - version information (lmver) 224
 - removing a licensed feature (lmremove) 224
 - rereading license file (lmreread) 224
 - shutting down license server (lmdown) 224
 - utility commands 223
- FlexNet log file 223
- FlexNet-based license 212
- FLOAT_CLIENT 229
- floating client
 - description 229
 - specifying host or model type 230
- floating client licenses
 - FLOAT_CLIENTS_ADDR_RANGE parameter in
 - lsbf.cluster.cluster_name 230
- floating LSF client
 - description 229
 - specifying host or model type 230
- floating LSF client licenses
 - FLOAT_CLIENTS_ADDR_RANGE parameter in
 - lsbf.cluster.cluster_name 231
- floating software licenses
 - configuring dedicated queue for 153
 - managing with LSF 151
- forcing job execution 97
- formula
 - fairshare dynamic user priority calculation 457
- free memory 136
- FS absolute job priority scheduling factor 599

G

- gethostbyname function (host naming) 71
- global fairshare 489
- GLOBAL_EXIT_RATE parameter in lsbf.params 115
- goal-oriented scheduling. *See* SLA scheduling
- goals
 - time-based SLA scheduling 521
- grace period
 - absolute job priority scheduling factor 598
 - EGO resources 875
 - licenses 216
- GROUP_ADMIN
 - lsbf.users 177

- groups
 - external host 80, 83
 - external user 178
 - hosts 78
 - users 174
- groups, specifying 487
- guarantee service class
 - configuring 516
- guaranteed resource pools
 - configuring 509
- H**
- hard resource limits
 - description 718
 - stack segment size 729
- hard resource usage limits
 - example 722
- hardware host ID
 - displaying (lmhostid) 224
- hardware host name 218
- hierarchical fairshare 472
- hierarchical share tree 473
- HIST_HOURS parameter in lsb.params
 - fairshare dynamic user priority 459
- historical run time 460
- history
 - job arrays 663, 665
- HJOB_LIMIT parameter in lsb.queues 562
- hname static resource 138
- home directories
 - remote file access 331
- hopen badmin command 53
- host affinity
 - same string 447
- host dispatch windows 376
- host entries
 - examples 73
- host exception handling
 - configuring 86
 - example 87
 - job exit rate exception 86, 112
- host groups 48
 - CONDENSE keyword 79, 83
 - condensed
 - viewing 48
 - configuring external host groups 80
 - defining 174

- defining condensed 79, 83
 - external
 - configuring 182
 - defining 182
 - description 180
 - overview 174
- host ID 218
 - displaying (lmhostid) 224
- host identifier 218
- host model static resource 138
- host models
 - adding custom names in lsf.shared 67
 - DEFAULT 355
 - select string 428
 - tuning CPU factors 84
- host name static resource 138
- host names
 - /etc/hosts file 70
 - aliases 70
 - matching with Internet addresses 70
 - ranges 77
 - ranges as aliases 70
 - resolv.conf file 71
 - resolver function 71
 - using DNS 71
 - wildcards and special characters 78, 82
- host partition fairshare 466
- host redirection 830
- host reservation. *See* advance reservation
- host selection 422
- host state. *See* host status
- host status
 - ok 44, 134
 - busy
 - load index 135
 - lsload 44
 - closed
 - bhosts 44
 - description 44
 - index 134
 - lockU and lockW 45, 135
 - ok
 - bhosts 44
 - load index 134
 - lsload 44
 - unavail
 - load index 135
 - lsload 45

- unlicensed 44, 45, 135
 - unreach 44
- host type static resource 138
- host types
 - adding custom names in lsf.shared 67
 - DEFAULT 355
 - resource requirements 416
 - select string 428
- host-based resources
 - description 131
- host-level
 - fairshare scheduling 466
 - resource information 587
- host-locked software licenses 150
- hostcache
 - modifying 65
- HOSTRESORDER environment variable 352
- hosts
 - adding with lsinstall 55
 - associating resources with 148
 - available 135
 - closing 53
 - connecting to remote 832
 - controlling 53
 - copying files across 809
 - dispatch windows 53
 - displaying 48
 - file 71
 - finding resource 810
 - for advance reservations 384
 - logging on the least loaded 810
 - multiple network interfaces 72
 - official name 70
 - opening 53
 - redirecting 830
 - removing 57
 - restricting use by queues 125
 - selecting for task 806
 - setting up 55, 56
 - spanning with parallel jobs 690
 - viewing
 - architecture information 49
 - detailed information 49
 - exceptions 51
 - history 50
 - job exit rate and load 51
 - load by host 49, 133
 - load by resource 130
 - model and type information 50
 - resource allocation limits (blimits) 568
 - shared resources 132
 - status of closed servers 49
 - suspending conditions 737
- hosts file (/etc/hosts)
 - example host entries 73
 - host naming 70
 - name lookup 71
 - troubleshooting 352
- hosts file (LSF)
 - configuring 72
- HOSTS parameter
 - in lsf.hosts 78
 - in lsf.queues file 78, 81
- hosts.equiv file
 - host authentication 293
 - using rcp 332
- hostsetup script 55, 56
- hrestart badmin command 20
- hshutdown badmin command 20
- hstartup badmin command 20

I

- idle job exceptions
 - configuring 127
 - description 112, 127
 - viewing with bjobs 94
 - viewing with bqueues 120
- idle time
 - built-in load index 136
 - description 795
 - suspending conditions 736
- if-else constructs
 - creating 374
 - files 373
- index list for job arrays 660
- input and output files
 - and interactive jobs 792
 - job arrays 661
 - splitting stdout and stderr 792
 - spooling directory 324
- installation directories
 - default UNIX structure 17
 - Windows default structure 18
- inter-queue priority 734
- interactive jobs

- competing for software licenses 153
- configuring queues to accept 790
- redirecting scripts to standard input 801
- resource reservation 443
- running X applications 799
- scheduling policies 790
- specifying embedded submission options 802
- specifying job options in a file 801
- specifying shell 802
- splitting stdout and stderr 792
- spooling job command files 801
- submitting 791
- submitting and redirecting streams to files 792
- submitting with pseudo-terminals 792
- viewing queues for 791
- writing job file one line at a time 801
- writing job scripts 801
- interactive sessions
 - starting 810
- interactive tasks
 - file access 808
- interfaces, network 72
- Internet addresses
 - matching with host names 70
- Internet Domain Name Service (DNS)
 - host naming 70
- interruptible backfill 704
 - resizable jobs 706
- INTERRUPTIBLE_BACKFILL parameter in lsib.queues 706
- io load index 136
- IPv6
 - configure hosts 75
 - supported platforms 75
 - using IPv6 addresses 75
- IRIX
 - Comprehensive System Accounting (CSA)
 - configuring 336
 - utmp file registration 804
- it load index
 - automatic job suspension 734
 - description 136, 795
 - suspending conditions 736

J

- JL/P parameter in lsib.users 562
- JL/U parameter in lsib.hosts 562
- job array dependency conditions

- operators 663
- job arrays
 - %I substitution string 661
 - %J substitution string 661
 - argument passing 662
 - controlling 665
 - creating 660
 - dependency condition operators 663
 - dependency conditions 662
 - file preparation 661
 - format 660
 - history 663, 665
 - index list 660
 - input and output files 661
 - maximum size 661
 - monitoring 663, 665
 - overview 655
 - passing arguments 662
 - redirection of input and output 661
 - specifying job slot limit 666
 - standard input and output 661
 - status 663, 665
 - submitting 660
 - syntax 660
- job checkpoint and restart
 - application level
 - configuring 632
 - description 629
 - enabling 632
 - application-level
 - echkpt requirements 632
 - erestart requirements 632
 - checkpoint directory 634
 - checkpoint files 634
 - commands 638
 - configuration to checkpoint jobs before suspension or termination 637
 - configuration to copy open job files to the checkpoint directory 637
 - configuration to save stderr and stdout 637
 - configuration to specify directory for application-level executables 636
 - configuration to specify mandatory application-level executables 636
 - configuring 631
 - description 628
 - echkpt 628
 - enabling 631

- erestart 628
 - kernel level
 - configuring 631
 - description 629
 - enabling 631
 - queue level
 - configuring 631
- resizable jobs 623
- scope 630
 - user level
 - configuring 632
 - description 629
 - enabling 632
- job chunking. *See* chunk jobs
- job control actions
 - CHKPNT 766
 - configuring 765
 - default actions 764
 - LS_EXEC_T 764
 - on Windows 765
 - overriding terminate interval 765
 - RESUME 764
 - SUSPEND 764
 - TERMINATE 765
 - terminating 767
 - using commands in 766
 - with lscsh 831
- job dependencies
 - logical operators 590
- job dependency conditions
 - advanced 593
 - description 591
 - done 591
 - ended 591
 - examples 593
 - exit 591
 - external 592
 - job arrays 662
 - job name 592
 - post_done 593
 - post_err 593
 - scheduling 590
 - specifying 590
 - specifying job ID 592
 - started 593
- job dispatch
 - fast 262
 - maximum per session 262
- job dispatch order, fairshare 465
- job email
 - bsub options 320
 - disabling batch job notification 320
 - limiting size with LSB_MAILSIZE_LIMIT 321
- job exception handling
 - configuring 127
 - default eadmin action 113
 - exception types 112, 127
 - viewing exception status with bjobs 94
 - viewing exceptions with bqueues 120
- job exit rate exceptions
 - configuring 86
 - description 86, 112
 - viewing with bhosts 51
- job file spooling
 - See also* command file spooling 323
 - default directory 324
 - description 323
 - JOB_SPOOL_DIR parameter in lsb.params 323
- job files 10
- job groups
 - add limits 106
 - automatic deletion 110
 - controlling jobs 108
 - default job group 103
 - description 102
 - displaying SLA service classes 527
 - example hierarchy 105
 - job limits 104
 - modify limits 110
 - viewing 107
- job idle factor
 - viewing with bjobs 94
- job limit 729
- job limits 557
 - job groups 104
- job migration
 - absolute job priority scheduling 604, 624
 - automatic
 - configure at host level 621
 - configure at queue level 620
 - configuring 620
 - enabling 620
 - configuration to modify 623
 - configuring 619
 - description 616
 - enabling 619

- scope 617
- job overrun exceptions
 - configuring 127
 - description 112, 127
 - viewing with bjobs 94
 - viewing with bqueues 120
- job packs 669
- job preemption
 - absolute job priority scheduling 604
 - description 402
 - job slot limits 407
 - time-based SLA scheduling 508, 516, 522
- job priority
 - automatic escalation 596
 - user assigned 595
- job requeue
 - absolute job priority scheduling 604
 - automatic 608
 - exclusive 610
 - resizable jobs 653
 - reverse requeue 610
 - user-specified 611
- job rerun
 - disabling post-execution commands 612
 - resizable jobs 612
- job restart
 - resizable jobs 623
- job scripts
 - writing for interactive jobs 801
- job slot limits 557
 - calculation of usage for preemption 407
 - for job arrays 666
 - for parallel jobs 679
 - viewing resource allocation limits (blimits) 568
- job spanning 422, 445
- job starters
 - augmentstarter 762
 - command-level 756
 - lic_starter script to manage software licenses 154
 - preservestarter 762
 - queue-level
 - configuring 760
 - description 756
 - specifying command or script 758, 760
 - user commands 760
- job states
 - description 90
 - DONE
 - description 90
 - EXIT
 - abnormal job termination 91
 - PEND 90
 - POST_DONE 92
 - POST_ERR 92
 - PSUSP 90
 - RUN 90
 - SSUSP 90
 - USUSP 90
 - WAIT for chunk jobs 658
- job submission
 - check ssched parameters 844
- job submission and execution controls
 - configuring 775, 785
 - description 770
 - enabling 775, 785
 - scope 774
- job success exit values
 - application profile configuration 541
 - JOB_CONTROLS parameter in lsb.queues 765
 - JOB_EXIT_RATE_DURATION parameter in lsb.params 86
 - JOB_GROUP_CLEAN parameter in lsb.params 110
 - JOB_IDLE parameter in lsb.queues 127
 - JOB_OVERRUN parameter in lsb.queues 127
 - JOB_POSITION_CONTROL_BY_ADMIN parameter in lsb.params 264
 - JOB_PRIORITY_OVER_TIME parameter in lsb.params 599
 - automatic job priority escalation 596
 - JOB_SCHEDULING_INTERVAL parameter in lsb.params 263
 - JOB_SPOOL_DIR parameter in lsb.params 323
 - JOB_STARTER
 - lsb.queues file 760
 - JOB_STARTER parameter in lsb.queues 760
 - JOB_TERMINATE_INTERVAL parameter in lsb.params 727, 765
 - JOB_UNDERRUN parameter in lsb.queues 127
- job-level
 - resource requirements 420
 - resource reservation 572
 - run limits 728
- job-level suspending conditions
 - viewing 737
- jobs
 - changing execution order 95
 - check pre-execution script 746
 - checkpointing
 - chunk jobs 659

- CHKPNT 766
 - controlling
 - in an application profile 543
 - email notification
 - disabling 320
 - options 320
 - enabling rerun 612
 - enforcing memory usage limits 727
 - exit codes
 - description 347
 - job success exit values 541
 - forcing execution 97
 - interactive. *See* interactive jobs
 - killing
 - in an application profile 543
 - limiting processors for parallel 695
 - modifying
 - in an application profile 543
 - optimum number running in time-based SLA 521
 - pending 90
 - preemption 734
 - requeueing 665
 - requeuing
 - description 611
 - rerunning 612
 - rerunning automatically 612
 - restarting
 - automatically 612
 - resuming
 - in an application profile 543
 - sending specific signals to 101
 - short running 656
 - specifying options for interactive 801
 - specifying shell for interactive 802
 - spooling command and job files 801
 - spooling input, output, and command files 323
 - stopping
 - in an application profile 543
 - submitting
 - to a job group 106
 - to a service class 529
 - to an application profile 543
 - suspended 737
 - suspending 98, 734
 - suspending at queue level 737
 - switching queues 96
 - viewing
 - by user 93

- configuration parameters in lsb.params 14
 - viewing resource allocation limits (blimits) 568
- jobs command in lscsh 831
- jobs requeue, description 608
- JRIORITY absolute job priority scheduling factor 599
- JSDL
 - configuration 149
 - elim for 163
 - load indices 163
 - required resources 147
- JSDL (Job Submission Description Language)
 - benefits 815
 - elim.jsdl 825
 - how to submit a job 824
 - LSF extension elements 821
 - schema files 815
 - supported elements 815
 - unsupported elements 823
 - using with LSF 815

K

- Kerberos authentication
 - configuration 316
 - configuration of 317
 - description 312
 - eauth user name
 - configuration of 317
 - enabling 317
- Kerberos daemon authentication
 - enabling 317
 - non-Solaris 317
 - Solaris 317
- kernel-level job checkpoint and restart
 - description 629

L

- libfairshareadjust 459
- lic_starter script, to manage software licenses 154
- license file (permanent) 215
- license host 216
- license key
 - calculating checksum (lmcksum) 224
- license management commands 223
- license server
 - calculating license key check sums (lmcksum) 224
 - checking status (lmstat) 223

- displaying
 - hardware host ID (Imhostid) 224
 - status (Imstat) 223
 - version information (Imver) 224
 - specifying TCP port 235
 - utility commands 223
- license server daemon (Imgrd) 216
- license server host
 - description 216
- license.dat (permanent) 215
- license.dat file
 - location 219
- licenses
 - cluster reconfiguration 25
 - displaying
 - FlexNet version information (Imver) 224
 - hardware host ID (Imhostid) 224
 - LSF products 227
 - server status (Imstat) 224
 - floating LSF client
 - specifying host or model type 230
 - removing feature (Imremove) 224
 - rereading license file (Imreread) 224
 - shutting down FlexNet server (Imdown) 224
 - software
 - counted 151
 - dedicated queue for 153
 - floating 151
 - host locked 150
 - interactive jobs and 153
- licensing
 - floating LSF client
 - description 229
- LIM (Load Information Manager)
 - tuning
 - load indices 282
 - load thresholds 283
 - policies 282
 - run windows 282
- LIM, master 216
- limdebug command 363
- limitations
 - lsrnp command 332
 - on chunk job queues 656
- limits
 - job 557
 - job group 104
 - job slot 557
- See* resource allocation limits or resource usage limits
- limtime command 365
- live reconfiguration
 - about 26
 - history files 30
 - liveconf.hist file 26
 - LSF_LIVE_CONFDIR 31
 - merge files 31
- lmcksum command 224
- lmdown command 224
- Imgrd daemon 216
- Imhostid 218
- Imhostid command 224
- Imremove command 224
- Imreread command 224
- Imstat command
 - description 224
 - using 223
- Imver command 224
- load average 135
- load indices
 - See also* resources
 - built-in
 - overriding 150
 - summary 134
 - io 136
 - it 136
 - ls 136
 - mem 136
 - pg 135
 - r15m 135
 - r15s 135
 - r1m 135
 - swp 136
 - tmp 136
 - tuning for LIM 282
 - types 794
 - ut 135
 - ut load index
 - select resource requirement string 428
 - viewing 136
- load levels
 - viewing by resource 130
 - viewing for hosts 49
- load sharing
 - displaying current setting 831
 - with lscsh 833
- load thresholds

- configuring 736
 - description 418
 - paging rate, tuning 283
 - queue level 736
 - resizable jobs 736
 - tuning 283
 - tuning for LIM 282, 283
- local event logging
 - mbatchd restart with MAX_INFO_DIRS 266
- local mode in lscsh 828
- local user account mapping 186
- locality
 - parallel jobs 445, 683, 690
- location of license (demo) 217
- location of license (permanent) 218
- lockU and lockW host status
 - lsload command 45
 - status load index 135
- log file (FlexNet) 223
- log files
 - change ownership 338
 - default location 336, 869
 - directory permissions and ownership 336, 870
 - ESC 871
 - logging events on Cray UNICOS 336
 - maintaining 338, 870
 - managing 338, 870
 - mbatchd.log.host_name 338, 870
 - mbschd.log.host_name 338, 870
 - named 871
 - PEM 871
 - res.log.host_name 338, 870
 - sbatchd.log.host_name 338, 870
 - VEMKD 871
 - WSG 871
- LOG_DAEMON facility, LSF error logging 339, 870
- logging classes
 - description 871
- logging levels 337
 - description 872
- logical operators
 - in time expressions 372
 - job dependencies 590
- login sessions 136
- login shell, using lscsh as 829
- lost_and_found queue 125
- ls load index 136
- LS_EXEC_T environment variable 764
- ls_postevent() arguments 280
- LSB_CHUNK_RUSAGE parameter in lsf.conf 719
- LSB_CONFDIR parameter in lsf.conf
 - default UNIX directory 17
- LSB_DEFAULT_JOBGROUP environment variable 103
- LSB_DISABLE_RERUN_POST_EXEC parameter in lsf.conf 612
- LSB_HOSTS environment variable 674
- LSB_JOB_CPULIMIT parameter in lsf.conf 725
- LSB_JOBINDEX environment variable 662
- LSB_JOBPGIDS environment variable 766
- LSB_JOBPIIDS environment variable 766
- LSB_LOCALDIR parameter in lsf.conf file 342
- LSB_MAILSIZE environment variable 321
- LSB_MAILSIZE_LIMIT parameter in lsf.conf 321
- LSB_MAILTO parameter in lsf.conf 320
- LSB_MAX_JOB_DISPATCH_PER_SESSION parameter in lsf.conf 262
- LSB_MBD_PORT parameter in lsf.conf 68
- LSB_NCPU_ENFORCE parameter in lsf.conf 710
- LSB_NTRIES environment variable 91
- LSB_QUERY_PORT parameter in lsf.conf 263, 287
- LSB_REQUEUE_TO_BOTTOM parameter in lsf.conf 608, 610
- LSB_SACCT_ONE_UG 467
- LSB_SBD_PORT parameter in lsf.conf 68
- LSB_SHAREDIR parameter in lsf.conf
 - default UNIX directory 17
 - duplicate event logging 341
- LSB_SHAREDIR/cluster_name/logdir
 - LSF log files 336, 869
- LSB_SIGSTOP parameter in lsf.conf 98, 767
- LSB_SUSP_REASON environment variable 766
- LSB_SUSP_SUBREASONS environment variable 766
- LSB_UTMP parameter in lsf.conf 804
- lsb.acct file
 - job exit information 343
 - job termination reason logging 343
 - killing jobs in a batch 99
- lsb.applications file
 - adding an application profile 540
 - NAME parameter 540
 - REQUEUE_EXIT_VALUES parameter 608
 - SUCCESS_EXIT_VALUES parameter 542
- lsb.events file
 - logging administrator comments
 - for host open and close 53
 - for mbatchd restart 22
 - for queue events 122
 - managing event log 340

- lsb.hosts file
 - CONDENSE keyword 79, 83
 - host exception handling 86
 - if-else constructs 373
 - time-based configuration 373
 - user groups 175
 - USER_SHARES parameter 175, 176
 - using host groups 78
 - using user groups 176
- lsb.modules file
 - advance reservation 381
 - schmod_advrsv plugin 381
- lsb.params file
 - absolute run time limit 728
 - CHUNK_JOB_DURATION parameter 657
 - CLEAN_PERIOD parameter 99
 - CONDENSE_PENDING_REASONS parameter 264
 - controlling lsb.events file rewrites 340
 - CPU time normalization 725
 - default application profile 541
 - default normalization host 725
 - DEFAULT_JOBGROUP parameter 103
 - EADMIN_TRIGGER_DURATION threshold for exception handling 128
 - ENABLE_EXIT_RATE_PER_SLOT parameter 115
 - GLOBAL_EXIT_RATE parameter 115
 - if-else constructs 373
 - job termination signal interval 727
 - JOB_EXIT_RATE_DURATION for exception handling 86
 - JOB_GROUP_CLEAN parameter 110
 - JOB_POSITION_CONTROL_BY_ADMIN parameter 264
 - JOB_PRIORITY_OVER_TIME parameter 596
 - JOB_SCHEDULING_INTERVAL parameter 263
 - JOB_SPOOL_DIR parameter 323
 - MAX_CONCURRENT_JOB_QUERY parameter 263
 - MAX_INFO_DIRS parameter 265
 - MAX_PEND_JOBS parameter 91
 - MAX_SBD_CONNS parameter 262
 - MAX_USER_PRIORITY parameter 595
 - MBD_QUERY_CPUS parameter 288
 - MBD_REFRESH_TIME parameter 286
 - MIN_REFRESH_TIME parameter 287
 - MIN_SWITCH_PERIOD parameter 265
 - NEWJOB_REFRESH parameter 288
 - non-normalized run time limit 728
 - PARALLEL_SCHED_BY_SLOT parameter 680
 - specifying job input files 323
 - SUB_TRY_INTERVAL parameter 91
 - time-based configuration 373
- lsb.queues file
 - adding a queue 124
 - ADMINISTRATOR\$parameter 176
 - EXCLUSIVE parameter 81
 - HOSTS parameter 78, 81
 - if-else constructs 373
 - job exception handling 127
 - JOB_IDLE parameter 127
 - JOB_OVERRUN parameter 127
 - JOB_UNDERRUN parameter 127
 - normalization host 731
 - QUEUE_NAME parameter 124
 - REQUEUE_EXIT_VALUES parameter 608
 - resource usage limits 722
 - restricting host use by queues 125
 - time-based configuration 373
 - user groups 175
 - USERS parameter 175, 176
 - using compute units 81
 - using host groups 78
 - using user groups 176
- lsb.queues files
 - DEFAULT_HOST_SPEC parameter 731
- lsb.resources file
 - advance reservation policies 381
 - if-else constructs 373
 - parameters 561
 - PER_USER parameter 176
 - time-based configuration 373
 - USERS parameter 176
 - using user groups 176
 - viewing limit configuration (blimits) 568
- lsb.serviceclasses file
 - ACCESS_CONTROL parameter 176
 - configuring guarantee SLA scheduling 516
 - configuring guaranteed resource pools 509
 - configuring SLA scheduling 522
 - CONTROL_ACTION 528
 - USER_GROUP parameter 176
 - using user groups 176
- lsb.users
 - GROUP_ADMIN 177
- lsb.users file
 - if-else constructs 373
 - MAX_PEND_JOBS parameter 91
 - time-based configuration 373
 - user groups 175

- USER_NAME parameter 175, 176
 - using user groups 176
- lsbapplications file
 - using compute units 81
- LSF Daemon Error Log 338, 870
- LSF daemon startup control
 - configuring 37
 - description 34
 - EGO administrator login bypass
 - configuring 38
 - description 35
 - enabling 37
 - scope 36
 - startup by users other than root
 - configuration of 37
 - configuring 37
 - description 34
 - enabling 37
- LSF events
 - generated by LSF 279
 - generation of 278
 - program arguments 280
- LSF features 221
- LSF license grace period 216
- LSF licenses
 - license file location 219
- LSF master LIM 216
- LSF parameters. *See* individual parameter names 14
- LSF products 221
 - displaying enabled license 227
- LSF vendor license daemon (lsf_ld) 216
- LSF_BINDIR parameter in lsf.conf 17, 332
- LSF_CONFDIR parameter in lsf.conf 17
- LSF_DYNAMIC_HOST_WAIT_TIME parameter in lsf.conf 59
- LSF_ENABLE_CSA parameter in lsf.conf 336
- LSF_INCLUDEDIR parameter in lsf.conf
 - default UNIX directory 17
- LSF_JOB_STARTER environment variable 758
- lsf_ld 216
- LSF_LICENSE_FILE parameter in lsf.conf 219
- LSF_LIM_PORT parameter in lsf.conf 68
- LSF_LIVE_CONFDIR
 - live reconfiguration directory 26
- LSF_LOG_MASK parameter in lsf.conf 338, 363
- LSF_LOGDIR parameter in lsf.conf 339, 870
- LSF_MANDIR parameter in lsf.conf 17
- LSF_MASTER_LIST parameter in lsf.conf 59
- LSF_MISC parameter in lsf.conf 17
- LSF_NT2UNIX_CLTRB environment variable 768
- LSF_NT2UNIX_CLTRC environment variable 768
- LSF_REMOTE_COPY_CMD 330, 332
- LSF_RES_PORT parameter in lsf.conf 68
- LSF_RSH parameter in lsf.conf
 - controlling daemons 20
- LSF_SERVERDIR 222
- LSF_SERVERDIR parameter in lsf.conf 17
- LSF_STRICT_CHECKING parameter in lsf.conf 63
- LSF_STRICT_RESREQ parameter in lsf.conf 431
- LSF_TOP directory
 - default UNIX directory structure 17
- lsf.cluster.cluster_name file
 - ADMINISTRATORS parameter 19
 - configuring cluster administrators 19
 - exclusive resources 431
 - license checkout 216
- lsf.conf file
 - comprehensive system account 336
 - configuring duplicate logging 342
 - configuring TCP service ports 68
 - custom file transfer 332
 - daemon service ports 68
 - default UNIX directory 17
 - duplicate event logging 341
 - dynamic host startup time 59
 - limiting the size of job email 321
 - LSB_CHUNK_RUSAGE parameter 719
 - LSB_DISABLE_RERUN_POST_EXEC parameter 612
 - LSB_JOB_CPULIMIT parameter 725
 - LSB_JOB_MEMLIMIT 726
 - LSB_MAILSIZE_LIMIT parameter 321
 - LSB_MAILTO parameter 320
 - LSB_MAX_JOB_DISPATCH_PER_SESSION parameter 262
 - LSB_MEMLIMIT_ENFORCE 726
 - LSB_QUERY_PORT parameter 263, 287
 - LSB_SIGSTOP parameter 98
 - LSF_BINDIR parameter 17, 332
 - LSF_DYNAMIC_HOST_WAIT_TIME parameter 59
 - LSF_ENABLE_CSA parameter 336
 - LSF_LOG_MASK parameter 338
 - LSF_LOGDIR parameter 339, 870
 - LSF_MANDIR parameter 17
 - LSF_MASTER_LIST parameter 59
 - LSF_MISC parameter 17
 - LSF_SERVERDIR parameter 17
 - LSF_STRICT_CHECKING parameter 63

- LSF_STRICT_RESREQ parameter 431
 - lsrnp command executable 332
 - managing error logs 338
 - per-job CPU limit 725
 - resource usage limits for chunk jobs 719
 - sending email to job submitter 320
 - setting message log to debug level 363
 - strict checking, enabling 63
 - lsf.conf parameter LSF_LICENSE_FILE 219
 - lsf.licensescheduler file
 - if-else constructs 373
 - time-based configuration 373
 - lsf.shared file
 - adding a custom host type and model 67
 - tuning CPU factors 84
 - lsfinstall
 - adding a host 55
 - lsfshutdown command
 - shutting down daemons on all hosts 20
 - lsfstartup command
 - starting daemons on all hosts 20
 - lshosts
 - viewing dynamic host information 52
 - lshosts -l
 - viewing licensed LSF products 227
 - lshosts command
 - DEFAULT host model or type 355
 - lsrnp command
 - description 330
 - executable file location 332
 - file transfer 332
 - restrictions 332
 - lstcsh
 - about 827
 - difference from other shells 829
 - exiting 829
 - limitations 829
 - local mode 828
 - remote mode 828
 - resource requirements 827
 - starting 829
 - task lists 827
 - using as login shell 829
 - writing shell scripts in 833
- M**
- mail
 - disabling batch job notification 320
 - job options 320
 - limiting the size of job email 321
 - mandatory first execution host
 - parallel jobs 681
 - resizable jobs 681
 - MAX_CONCURRENT_JOB_QUERY parameter in lsb.params 263
 - MAX_INFO_DIRS parameter in lsb.params 265
 - MAX_JOB_NUM parameter in lsb.params 340
 - MAX_JOBS parameter in lsb.users 562
 - MAX_PEND_JOBS parameter in lsb.params or lsb.users 91
 - MAX_RESERVE_TIME parameter in lsb.queues 576, 577
 - MAX_SBD_CONNS parameter in lsb.params 262
 - MAX_SLOTS_IN_POOL parameter
 - in lsb.queues 477
 - MAX_SLOTS_IN_POOL parameter in lsb.queues 477
 - MAX_USER_PRIORITY parameter in lsb.params
 - automatic job priority escalation 596
 - user-assigned job priority 595
 - maxcus keyword
 - cu string 684
 - maximum
 - number of processors for parallel jobs 680
 - resource usage limit 722
 - maxmem static resource 138
 - maxswp static resource 138
 - maxtmp static resource 138
 - mbatchd (master batch daemon)
 - expiry time 287
 - push new job information to a child mbatchd 287, 288
 - refresh time 287
 - restarting 22
 - shutting down 22
 - specifying query-dedicated port 287
 - specifying time interval for forking child 287
 - tuning on UNIX 286
 - mbatchd.log.host_name file 338, 870
 - MBD_QUERY_CPUS parameter in lsb.params 288
 - MBD_REFRESH_TIME parameter in lsb.params 286
 - MBD. *See* mbatchd
 - mbddebug command 363
 - mbdrestart badmin command 20
 - mbdtime command 365
 - mbschd.log.host_name file 338, 870
 - MEM absolute job priority scheduling factor 599
 - mem load index
 - description 136

- MEMLIMIT parameter in lsb.queues 726
- memory
 - available 136
 - usage limit 727
 - viewing resource allocation limits (blimits) 568
- mesub
 - definition 771
- migrated jobs
 - absolute job priority scheduling 604, 624
- MIN_REFRESH_TIME parameter in lsb.params 287
- MIN_SWITCH_PERIOD parameter in lsb.params 265
- minimum processors for parallel jobs 680
- missed SLA scheduling goals
 - control action 528
- model static resource 138
- modify
 - LSF_MASTER_LIST 58
- multi-core hosts 216
- multi-homed hosts 72
- MultiCluster
 - resource-based SLA scheduling 516
 - time-based SLA scheduling 522
- multiple condensed host groups 80
- multiple conditions
 - dependency expressions 590
- multiple license server hosts 225
- multiple queues
 - absolute job priority scheduling 601
- multiprocessor hosts
 - configuring queue-level load thresholds 737
 - tuning LIM 284
- multithreading, configuring mbatchd for 286
- MXJ parameter in lsb.hosts 562

N

- name lookup using /etc/hosts file 71
- NAME parameter in lsb.applications 540
- native language system, and lscsh 829
- ncores static resource 138
- ncpus static resource
 - dynamically changing processors 144
 - reported by LIM 138
- ndisks static resource 138
- network
 - interfaces 72
 - partitioning
 - and duplicate event logging 341

- port numbers
 - configuring for NIS or NIS+ databases 69
- Network File System. *See* NFS
- Network Information Service. *See* NIS
- NEWJOB_REFRESH parameter in lsb.params 288
- NFS (Network File System)
 - automount command 328, 350
 - nosuid option 294
 - overview 328
- NIS (Network Information Service)
 - configuring port numbers 69
 - host name lookup in LSF 70
 - ypcat hosts.byname 71
- non-normalized run time limit 728
- non-uniform user name space
 - between-host user account mapping
 - description 186
 - cross-cluster user account mapping
 - description 192
- normalization
 - CPU time limit 731
 - host 731
 - run time limit 731
- normalization host 725
- normalized run queue length
 - description 135
 - tuning LIM 284
- nosuid option, NFS mounting 294
- NOT operator (!)
 - job dependencies 590
- not operator (~)
 - host partition fairshare 466
 - host-based resources 149
- nprocs static resource 138
- NQS (Network Queueing System)
 - logging events on Cray UNICOS 336
- nqsacct file 336
- nthreads static resource 138
- number of processors for parallel jobs 680
- numdone dependency condition 662
- numended dependency condition 662
- numerical resources 131
- numexit dependency condition 662
- numhold dependency condition 663
- numpend dependency condition 662
- numrun dependency condition 662
- numstart dependency condition 663

O

- obsolete parameters
 - FAIRSHARE_QUEUES 601
 - USER_ADVANCE_RESERVATION in lsb.params 381
- official host name 70
- ok host status
 - bhosts command 44
 - lsload command 44
 - status load index 134
- one-time advance reservation 384
- operators
 - job array dependency conditions 663
 - logical in job dependencies 590
 - logical in time expressions 372
 - not (~)
 - host partition fairshare 466
 - host-based resources 149
 - relational
 - exit dependency condition 591
 - resource requirements 429
 - selection strings 429
- OR operator (||)
 - job dependencies 590
- order of job execution, changing 95
- order resource requirement string
 - resizable jobs 436
- order string 435
- OS memory limit 727
- output and input files, for job arrays 661
- output file spooling
 - default directory 324
- overrun job exceptions
 - configuring 127
 - description 112, 127
 - viewing with bjobs 94
 - viewing with bqueues 120
- ownership of log directory 336, 870

P

- paging rate
 - automatic job suspension 735
 - checking 736
 - description 135, 794
 - load index 135
 - suspending conditions 736
- parallel fairshare 710

- parallel jobs
 - allocating processors 676
 - backfill scheduling 701
 - fairshare 710
 - interruptible backfill scheduling 704
 - job slot limits 679
 - limiting processors 695
 - locality 445, 683, 690
 - mandatory first execution host 681
 - number of processors 680
 - preemption of 413
 - processor reservation 698
 - PROCLIMIT
 - resizable jobs 696
 - selecting hosts with same string 447
 - spanning hosts 690
 - submitting 676
- parallel programming
 - packages 675
- parallel tasks
 - running with lsgroup 807
 - starting 677
- PARALLEL_SCHED_BY_SLOT parameter in lsb.params 680
- partial licensing 227
- PATH environment variable
 - and lscsh 829
- paths
 - /etc/hosts file
 - example host entries 73
 - host naming 70
 - name lookup 71
 - /etc/hosts.equiv file
 - host authentication 293
 - using rcp 332
 - /etc/services file
 - adding LSF entries to 68
 - /net 330
- PEND
 - job state 90
- pending jobs
 - absolute job priority scheduling 597
 - order of absolute job priority scheduling 602
- pending reasons
 - queue-level resource reservation 574
 - viewing 91
- PER_USER parameter in lsb.resources 176
- per-core licensing 216
- per-job CPU limit 725

- per-process limits
 - CPU limit 725
 - data segment size 726
 - file size 726
 - memory limit 727
 - stack segment size 729
- per-resource reservation
 - configuring 575
 - viewing with bresources 588
- performance tuning
 - busy thresholds 282
 - LIM policies 282
 - load indices 282
 - load thresholds 283
 - mbatchd on UNIX 286
 - run windows for LIM 282
- periodic tasks 338, 870
- permanent license 212
- permanent LSF license
 - displaying server status (lmstat) 223
- permissions
 - log directory 336, 870
- pg load index
 - suspending conditions 736
- PIM (Process Information Manager)
 - resource use 133
- PJOB_LIMIT parameter in lsb.queues 562
- PluginModule section in lsb.modules
 - advance reservation 381
- policies
 - fairshare 454
 - tuning for LIM 282
- port
 - notation
 - license daemon 235
- port (in LSF_LICENSE_FILE)¶ 219
- port numbers
 - configuring for NIS or NIS+ databases 69
- ports
 - registering daemon services 68
 - specifying dedicated 287
- post_done job dependency condition 593
- POST_DONE post-execution job state 92
- post_err job dependency condition 593
- POST_ERR post-execution job state 92
- post-execution commands
 - disabling for rerunnable jobs 612
- pre- and post-execution processing
 - application level
 - configuration of 743
 - enabling 743
 - configuring 742
 - description 740
 - enabling 742
 - include post-processing in job finish status
 - configuration of 747
 - post-processing timeout
 - configuration of 748
 - queue level
 - configuration of 742
 - enabling 742
 - user account
 - configuration of 748
- pre-and post execution processing
 - scope 741
- pre-execution retry limit
 - application level
 - configuration of 749
 - enabling 749
 - cluster-wide
 - configuration of 749
 - enabling 749
 - queue level
 - configuration of 749
 - enabling 749
- pre-execution script
 - check job history 746
- PREEMPT_FOR parameter in lsb.params 712
- preemptable queues
 - definition 402
- preempted jobs
 - control action 413
 - limit preemption retry 413
- preemption
 - absolute job priority scheduling 604
- preemption. *See* preemptive scheduling
- preemptive
 - scheduling
 - description 402
- preemptive queues
 - definition 402
- preemptive scheduling
 - advance reservation 396
 - configuration of 409
 - control action for preempted jobs 413
 - description 402

- enabling 405
- job slot limits 407
- job slot usage 407
- limit preemption retry 413
- limitations 402
- order of preemption 406
- parallel jobs 413
- per-host job slot limit for users and user groups 412
- per-processor job slot limit for a user 412
- per-processor job slot limit for user groups 412
- time-based SLA scheduling 508, 516, 522
- total job slot limit for user groups 412
- pref keyword
 - cu string 684
- preservestarter job starter 762
- priority
 - automatic escalation 596
 - user assigned 595
- PRIORITY parameter in lsb Queues 470, 477
- priority user fairshare 490
- priority. *See* dynamic user priority
- PROC absolute job priority scheduling factor 599
- process allocation for parallel jobs 422, 447
- PROCESSLIMIT parameter in lsb Queues 727
- processor binding
 - resizable jobs 713
- processor reservation
 - configuring 698
- processors
 - limiting for parallel jobs 695
 - number for parallel jobs 680
 - reservation 698
- PROCLIMIT parameter in lsb Queues 599
- products 221
- PRODUCTS line, editing 221
- programs
 - handling LSF events 278
- project names
 - viewing resource allocation limits (blimits) 568
- pseudo-terminal
 - submitting interactive jobs with 792
 - using to run a task 807
- PSUSP job state
 - description 98
 - overview 90

Q

- qact badmin command 122

- qclose badmin command 122
- qinact badmin command 122
- QJOB_LIMIT parameter in lsb Queues 563
- qopen badmin command 122
- QRIORITY absolute job priority scheduling factor 599
- queue dispatch windows 376
- queue groups
 - absolute job priority scheduling 601
- QUEUE_GROUP parameter in lsb Queues 601
- QUEUE_NAME parameter in lsb Queues 124
- queue-based fairshare
 - resource usage measurement 457
 - resource-based SLA scheduling 508
- queue-level
 - fairshare across queues 468
 - fairshare scheduling 467
 - job starter 760
 - resource limits 722
 - resource requirements 418
 - resource reservation 574
 - run limits 723
- queue-level resource information
 - viewing 587
- queue-level resource limits, defaults 722
- queues
 - adding and removing 124
 - backfill queue 702
 - changing job order within 95
 - chunk job limitations 656
 - configuring
 - for chunk jobs 656
 - job control actions 765
 - suspending conditions 737
 - dispatch windows 123
 - fairshare across queues 468
 - interactive 790
 - interruptible backfill 706
 - lost_and_found 125
 - preemptive and preemptable 402
 - restricting host use 125
 - run windows 124
 - setting rerun level 612
 - specifying suspending conditions 737
 - user-assigned job priority 595
 - viewing
 - available 119
 - detailed queue information 119
 - for interactive jobs 791

- history 120
 - job exception status 120
 - resource allocation limits (blimits) 568
 - status 119
- R
- r15m load index
 - built-in resources 135
 - description 795
 - suspending conditions 736
- r15s load index
 - built-in resources 135
 - description 795
 - suspending conditions 736
- r1m load index
 - built-in resources 135
 - description 795
 - suspending conditions 736
- ranges
 - host name aliases 70
- rcp command 330
- recurring advance reservation 385
- redundant license server hosts 225
- relational operators
 - exit dependency condition 591
- remote execution
 - with lscsh 828
- remote jobs
 - bringing background jobs to foreground 831
 - execution priority 138
- remote mode in lscsh 828
- remove
 - master host 58
- removing
 - a licensed feature (lremove) 224
- REQUEUE_EXIT_VALUES parameter in lsb.applications 608
- REQUEUE_EXIT_VALUES parameter in lsb.queues 608, 610
- queued jobs
 - absolute job priority scheduling 604
 - automatic 608
 - description 608
 - exclusive 610
 - resizable jobs 653
 - reverse 610
 - user-specified 611
- rerunnable jobs 612
 - chunk jobs 659
 - disabling post-execution 612
- RERUNNABLE parameter in lsb.queues 612
- RES_REQ parameter
 - in lsb.applications 81
 - in lsb.queues 81
- res.log.host_name file 338, 870
- resdebug command 363
- reservation
 - advance 380, 381
- reservation ID
 - advance reservation 394
- reservation limits
 - resource requirements 572
- reserved memory
 - for pending jobs 587
- resizable jobs
 - absolute job priority scheduling 605
 - advance reservations 397
 - automatic job priority escalation 596
 - backfill scheduling 702
 - bresize cancel command 650
 - bresize release command 650
 - checkpoint and restart 623
 - chunk jobs 653
 - compute units 398
 - cu resource requirement string 450
 - deadline constraint scheduling 378
 - exclusive scheduling 534
 - fairshare scheduling 492
 - first execution host 681
 - interruptible backfill 706
 - job rerun 612
 - JOB_ACCEPT_INTERVAL parameter 653
 - limiting processors for parallel jobs 696
 - load thresholds 736
 - minimum and maximum processors for parallel jobs 680
 - order resource requirement string 436
 - processor binding 713
 - queued jobs 653
 - resource allocation limits 559
 - resource requirements 417
 - resource-based SLA scheduling 508
 - usage resource requirement string 443
 - same resource requirement string 447
 - select resource requirement string 434
 - slot reservation 572
 - span resource requirement string 446

- switched jobs 653
- time-based SLA scheduling 522
- time-based slot reservation 583
- resize
 - notification command 651
- resolv.conf file 71
- resolver function 71
- resource allocation limits
 - description 556
 - enforcement 557
 - job limits 557
 - job slot limits 557
 - resource requirements 556
 - resource reservation and backfill 557
 - switched jobs 558
 - viewing (blimits) 568
- resource configurations
 - viewing with blimits 568
- resource consumers 556
- resource granularity 575
- resource names
 - aliases 428
 - description 146
- resource reclaim
 - grace period 875
- resource requirement string
 - cu section
 - syntax 684
- resource requirements
 - and task lists in lscsh 827
 - compound
 - multi-level 424
 - syntax 423
 - compute units 422
 - description 416
 - exclusive resources 431
 - for advance reservations 384
 - host type 416
 - operators 429
 - ordering hosts 422, 435
 - parallel job locality 422, 445
 - parallel job processes 422, 447
 - parallel jobs
 - selecting hosts 447
 - reservation limits 572
 - resizable jobs 417
 - resource reservation 437
 - resource usage 422, 437
- select string 428
- selecting hosts 422, 428, 447
 - simple
 - multi-level 423
 - syntax 422
 - topology 449
- resource reservation
 - absolute job priority scheduling 604
 - description 572
 - resizable jobs 572
 - resource allocation limits 557
 - static shared resources 150
- resource types
 - external resources 131
- resource usage
 - fairshare scheduling 457
 - resource requirements 422, 437
 - viewing 133
- resource usage limits
 - ceiling 722
 - chunk job enforcement 719
 - configuring 722
 - conflicting 719
 - default 722
 - for deadline constraints 378
 - hard 722
 - maximum 722
 - priority 719
 - soft 722
 - specifying 722
- RESOURCE_RESERVE parameter in lsb.queues 576, 577, 584, 700
- RESOURCE_RESERVE_PER_SLOT parameter in lsb.params 443
- RESOURCE_RESERVE_PER_SLOT parameter in lsb.params 253, 438, 444, 575, 599
- resource-based service level goals
 - job preemption 508, 516
- resource-based SLA scheduling
 - advance reservation 508
 - chunk jobs 508
 - compute units 508
 - EGO-enabled SLAs 516
 - exclusive jobs 508
 - MultiCluster 516
 - queue-based fairshare 508
 - resizable jobs 508
- ResourceMap section in lsf.cluster.cluster_name 148

ResourceReservation section in lsb.resources 381

resources

See also load indices 794

adding custom 146

advance reservations 380

associating with hosts 148

Boolean 131

built-in 134

configuring custom 146

custom 146

host-level 587

per-resource configuration 588

queue-level 587

shared 132

types 131

viewing

available 130

host load 130

RESRSV_LIMIT, lsb.queues 572

restime command 365

restrictions

chunk job queues 656

lsrnp command 332

lstcsh 829

RESUME job control action 764

resume thresholds

viewing 738

RESUME_COND parameter in lsb.queues 765

reverse requeue 610

rexpri static resource 138

rhosts file

troubleshooting 352

rlogin command

interactive terminals 795

rsh command

lsfrestart 20

RUN job state

overview 90

run limits

configuring 719, 728

default 723

specifying 730

run queue

effective 135

normalized 135

suspending conditions 736

run time

decayed 460, 461

historical 460

normalization 731

run time decay 461

run time limit

non-normalized (absolute) 728

run windows

description 376

queues 124

tuning for LIM 282

RUN_JOB_FACTOR parameter in lsb.params

fairshare dynamic user priority 458, 459

RUN_TIME_FACTOR parameter in lsb.params

fairshare dynamic user priority 458

RUN_WINDOW

queues 124

RUNLIMIT parameter in lsb.queues 703, 728

running jobs

viewing 90

rusage

resource requirements section 422

resource reservation 572

usage string syntax 437

rusage resource requirement string

resizable jobs 443

S

same resource requirement string

resizable jobs 447

same string 447

sample /etc/hosts file entries 73

sanity-check ssched parameters 844

sbatchd (slave batch daemon)

remote file access 330

restarting 21

shutting down 21

sbatchd.log.host_name file 338, 870

sbddebug command 363

sbdtime command 365

schdddebug command 363

schddtime command 365

scheduling

exclusive 534

fairshare 454

hierarchical fairshare 472

preemptive

description 402

service level agreement (SLA) 514

- threshold
 - queue-level resource requirements 418
- scheduling policies
 - absolute job priority scheduling 597
 - automatic job priority escalation 596
 - user-assigned job priority 595
- scheduling priority factors
 - absolute job priority scheduling 597
- schmod_advrsv plugin for advance reservation 381
- scripts
 - check_license for counted software licenses 151
 - lic_starter to manage software licenses 154
 - redirecting to standard input for interactive jobs 801
 - writing for interactive jobs 801
 - writing in lscsh 833
- SDK
 - defining demand 860
- security
 - LSF authentication 292
- select resource requirement string
 - resizable jobs 434
 - ut load index 428
- selection strings
 - defined keyword 430
 - description 428
 - operators 429
- server hosts, viewing detailed information 49
- SERVER line
 - license.dat file 215
- server static resource 138, 139
- server status closed 49
- servers
 - displaying license status (lmstat) 224
- service class
 - goal-oriented scheduling 521
- service classes
 - bacct command 529, 531
 - bjgroup command 527
 - bjobs command 531
 - description 514
 - submitting jobs 529
- service database examples 68
- service level agreement. *See* SLA scheduling
- service level goals
 - time-based service classes 521
- service ports (TCP and UDP)
 - registering 68
- services

- about 862
 - cluster
 - service director 862
 - web service gateway 862
 - WEBGUI 862
- session jobs
 - kill the session (bkill) 844
- Session Scheduler session
 - kill the session (bkill) 844
- setuid permissions 352
- SGI IRIX. *See* IRIX
- share assignments 455
- share tree 473
- shared file systems
 - using LSF without 329
- shared files 350
- shared resources
 - defined keyword 430
 - description 132
 - exclusive resource selection strings
 - exclusive resources 431
 - static
 - reserving 150
 - viewing 132
- shares
 - fairshare assignment 455
 - viewing user share information 172
- shell mode, enabling 809
- shell scripts. *See* scripts
- shell variables and lscsh 829
- shells
 - default shell for interactive jobs 802
 - lscsh 829
 - specifying for interactive jobs 802
- short-running jobs, as chunk jobs 656
- shutting down
 - FlexNet server (lmdown) 224
- shutting down license server (lmdown) 224
- SIGCONT signal
 - default RESUME action 764
 - job control actions 101
- SIGINT signal
 - conversion to Windows 768
 - default TERMINATE action 765
 - job control actions 101
- SIGKILL signal
 - default TERMINATE action 765
 - job control actions 101

- sending a signal to a job 101
- signals
 - avoiding job action deadlock 767
 - configuring SIGSTOP 98, 764, 767
 - converting 768
 - customizing conversion 768
 - job exit codes 347
 - sending to a job 101
 - SIGINT 101
 - SIGTERM 101
- SIGQUIT signal
 - conversion to Windows 768
- SIGSTOP signal
 - bstop 98
 - configuring 98, 764, 767
 - default SUSPEND action 764
 - job control actions 101
- SIGTERM signal
 - default TERMINATE action 765
 - job control actions 101
- SIGTSTP signal
 - bstop 98
 - default SUSPEND action 764
- simple resource requirements
 - multi-level 423
 - syntax 422
- sitched jobs
 - resource allocation limits 558
- site-defined resources
 - resource types 131
- SLA guarantee scheduling
 - configuring 516
- SLA scheduling
 - bacct command 531
 - bjgroup command 527
 - bjobs command 531
 - bsla command 529
 - deadline goals 514
 - delayed goals 528
 - description 514
 - guarantee goals 514
 - missed goals 528
 - service classes
 - description 514
 - submitting jobs 529
 - throughput goals 514
 - velocity goals 514
 - violation period 528
- slot limits 557
- slot reservation
 - resizable jobs 572
- SLOT_POOL parameter
 - in lsb.queues 477
- SLOT_RESERVE parameter in lsb.queues 576, 584, 700
- SLOT_SHARE parameter in lsb.queues 477
- slots
 - viewing resource allocation limits (blimits) 568
- soft resource limits
 - data segment size 726
 - description 718
 - example 722
 - file size 726
 - memory usage 727
- software licenses
 - counted 151
 - floating
 - dedicated queue for 153
 - description 151
 - host locked 150
 - interactive jobs competing with batch jobs 153
 - managing 154
- span resource requirement string
 - resizable jobs 446
- span string 445
- special characters
 - defining host names 78, 82
- specifying
 - license server TCP port 235
- spooling. *See* command file spooling, job file spooling
- ssched command
 - check parameters 844
- SSH 294, 780, 781
- SSH X11 forwarding
 - setting up 800
- SSUSP job state
 - description 98
 - overview 90
- stack segment size limit 729
- STACKLIMIT parameter in lsb.queues 729
- standard input and error
 - splitting for interactive jobs 792
- standard input and output
 - job arrays 661
- standard output and error
 - redirecting to a file 809
- started job dependency condition 593

- static job priority
 - absolute job priority scheduling 600
- static priority fairshare 490
- static resources
 - See also* individual resource names
 - description 138
 - shared
 - reserving 150
- statistics file
 - time-based SLA scheduling 522
- status
 - closed in bhosts 49
 - displaying license server (lmstat) 224
 - job arrays 663, 665
 - load index 134
 - viewing
 - queues 119
 - WAIT for chunk jobs 658
- STATUS
 - bhosts 44
- stderr and stdout
 - redirecting to a file 809
 - splitting for interactive jobs 792
- STOP_COND parameter in lsb.queues 764
- stopping
 - FlexNet server (lmdown) 224
- STRICT_UG_CONTROL parameter
 - lsb.params file 177
- string resources 131
- SUB_TRY_INTERVAL parameter in lsb.params 91
- subfactors
 - absolute job priority scheduling 600
- submission options
 - embedding for interactive jobs 802
- success exit values
 - application profile configuration 541
- SUCCESS_EXIT_VALUES parameter in lsb.applications 542
- Sun Network Information Service/Yellow Pages. *See* NIS
- supported file systems 328
- SUSPEND job control action
 - default 764
- suspended jobs
 - resuming 738
 - states 91
 - viewing resource allocation limits (blimits) 568
- suspending conditions
 - configuring 737
 - viewing 737
- suspending reason
 - viewing 91, 737
- suspending thresholds 737
- swap space
 - load index 136
 - suspending conditions 736
 - viewing resource allocation limits (blimits) 568
- SWAPLIMIT parameter in lsb.queues 729
- switched jobs
 - resizable jobs 653
- SWP absolute job priority scheduling factor 599
- swp load index
 - description 136
 - suspending conditions 736
 - viewing resource allocation limits (blimits) 568
- syslog.h file 338
- system overview 860

T

- task control
 - with lscsh 831
- task lists
 - and lscsh 827
 - changing memberships 828
- task submission
 - check ssched parameters 844
- tasks
 - file access 808
 - running same on many hosts in sequence 807
 - selecting host to run on 806
 - starting parallel 677
- TCP service port numbers
 - configuring for NIS or NIS+ databases 69
 - registering for LSF 68
- tcsh
 - version and lscsh 829
- temp space
 - suspending conditions 736
 - viewing resource allocation limits (blimits) 568
- temporary license 212
- TERMINATE job control action 765
- TERMINATE_WHEN parameter in lsb.queues
 - changing default SUSPEND action 767
 - TERMINATE job control action 765
- TerminateProcess() system call (Windows)
 - job control actions 765
- THREADLIMIT parameter in lsb.queues 728

- threads
 - job limit 729
- thresholds
 - exited job exceptions 86
 - idle job exceptions 127
 - job exit rate for hosts 86, 112
 - job overrun exceptions 127
 - job underrun exceptions 127
 - scheduling and suspending 737
 - tuning for LIM 283
- tilde (~)
 - not operator
 - host partition fairshare 466
 - host-based resources 149
- time expressions
 - creating for automatic configuration 372
 - logical operators 372
- time normalization
 - CPU factors 731
- time values
 - specifying 370
- time windows
 - syntax 371
- time-based configuration
 - automatic 373
 - commands for checking 375
- time-based resource limits 378
- time-based service class
 - configuring 522
 - examples 523
- time-based service level goals
 - job preemption 522
 - optimum number of running jobs 521
- time-based SLA scheduling
 - chunk jobs 522
 - configuring 522
 - examples 523
 - job preemption 508, 516, 522
 - MultiCluster 522
 - optimum number of running jobs 521
 - resizable jobs 522
 - service level goals 521
 - statistics file 522
 - user groups 522
- time-based slot reservation
 - resizable jobs 583
- timing level
 - commands for daemons 365
- tmp load index
 - description 136
 - suspending conditions 736
 - viewing resource allocation limits (blimits) 568
- type keyword
 - cu string 684
- type static resource 50, 138

U

- UDP service port numbers
 - registering for LSF 68
- UJOB_LIMIT parameter in lsb.queues 562
- unavail host status
 - bhosts command 44
 - lsload command 45
 - status load index
 - status load index 135
- uncondensed host groups
 - viewing 48
- underrun job exceptions
 - configuring 127
 - description 112, 127
 - viewing with bjobs 94
 - viewing with bqueues 120
- UNICOS accounting 336
- UNIX directory structure
 - example 17
- UNIX/Windows user account mapping
 - configuring 203
 - description 200
 - enabling 203, 205
 - example 204
 - local machine name
 - enabling 203
 - multi-domain
 - enabling 203
 - scope 201, 237
 - single domain
 - enabling 203
- unlicensed cluster 217
- unlicensed host status
 - bhosts command 44
 - lsload command 45
 - status load index 135
- unreach host status
 - bhosts command 44
- update interval 342

- duplicate event logging 342
- usage limits. *See* resource usage limits
- usage string 437
- USE_PRIORITY_IN_POOL parameter
 - in lsb.queues 477
- USE_PRIORITY_IN_POOL parameter in lsb.queues 477
- user account mapping
 - between-host
 - description 186
 - local user account mapping 188, 189
 - Windows workgroup 188
 - Windows workgroup account mapping 189
 - cross-cluster
 - configuring 194
 - description 192
 - enabling 194
 - system level 194, 195
 - user level 194, 195
 - local user account mapping 186
 - UNIX/Windows
 - configuring 203
 - description 200
 - enabling 203
 - example 204
 - Windows workgroups 186
- user authentication
 - security 292
- user group administrators
 - about 176
 - configure 177
 - rights 177
 - viewing 173
- user groups
 - configuring external user groups 178
 - external
 - configuring 182
 - defining 182
 - description 180
 - overview 174
 - specifying 487
 - time-based SLA scheduling 522
 - viewing information about 172
- user groups and limits 559
- user priority
 - description 457
 - formula 457
- user share assignments 455
- USER_ADVANCE_RESERVATION parameter in lsb.params
 - obsolete parameter 381
- USER_GROUP parameter in lsb.serviceclasses 176
- USER_NAME parameter in lsb.users 176
- USER_NAME parameter in lsb.users file 175
- USER_SHARES parameter in lsb.hosts 176
- USER_SHARES parameter in lsb.hosts file 175
- user-assigned job priority 595
- user-based host partition fairshare
 - resource usage measurement 457
- user-based queue-level fairshare
 - resource usage measurement 457
- user-level job checkpoint and restart
 - description 629
- user-specified job requeue 611
- users
 - viewing information about 172
 - viewing jobs submitted by 93
 - viewing resource allocation limits (blimits) 568
 - viewing shares 172
- USERS parameter in lsb.queues 176
- USERS parameter in lsb.queues file 175
- USERS parameter in lsb.resources 176
- USUSP job state
 - description 98
 - overview 90
 - suspending and resuming jobs 98
- ut load index
 - built-in resource 135
 - select resource requirement string 428
- utmp file registration on IRIX
 - enabling 804

V

- variables. *See* individual environment variable names 321
- vendor daemon (lsf_ld) 216
- version information, displaying in FlexNet (lmver) 224
- viewing
 - configuration errors 25
- viewing condensed and uncondensed 48
- violation period
 - SLA scheduling 528
- virtual memory
 - load index 136
 - suspending conditions 736
- virtual memory limit 729
- vmstat 136

W

WAIT status of chunk jobs

- description 658

- viewing 94

wall-clock run time limit 728

weekly planner for advance reservation (brsvs -p) 392

wildcards

- defining host names 78, 82

windows

- dispatch 376

- run 376

- time 371

Windows

- default directory structure 18

- job control actions 765

 - TerminateProcess() system call

- job control actions 765

- workgroup account mapping 186

Windows Event Viewer 278

workarounds to lsrcp limitations 332

X

X applications

- running with bsub 799

X11 800

xterm

- starting in LSF Base 811

Y

ypbind daemon 71

ypcat hosts.byname 71

ypmake command 69