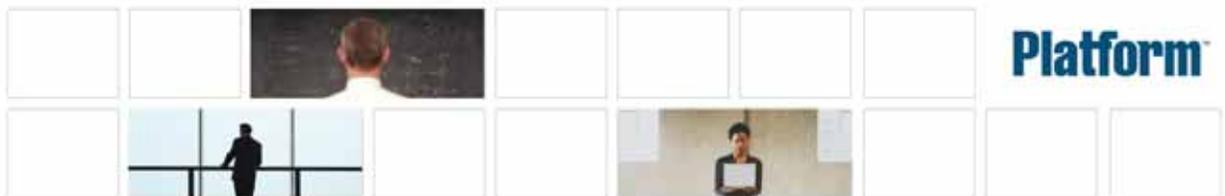

Administering Platform™ LSF™

Version 7 Update 5
Release date: March 2009
Last modified: March 16, 2009
Comments to: doc@platform.com
Support: support@platform.com



Copyright

© 1994-2009, Platform Computing Inc.

Although the information in this document has been carefully reviewed, Platform Computing Inc. (“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 Inc. in the United States and in other jurisdictions.

POWERING HIGH PERFORMANCE, PLATFORM COMPUTING, PLATFORM SYMPHONY, PLATFORM JOBSCHEDULER, PLATFORM ENTERPRISE GRID ORCHESTRATOR, PLATFORM EGO, and the PLATFORM and PLATFORM LSF logos are trademarks of Platform Computing Inc. 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.

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.

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/legal-notices/third-party-license-agreements>

Contents

1	About Platform LSF	15
	Contents	15
	Learn about Platform LSF	16
	Cluster Concepts	16
	Job Life Cycle	28
2	How the System Works	31
	Contents	31
	Job Submission	32
	Job Scheduling and Dispatch	34
	Host Selection	36
	Job Execution Environment	37
	Fault Tolerance	38

Part I: Managing Your Cluster

3	Working with Your Cluster	43
	Contents	43
	Viewing cluster information	44
	Example directory structures	49
	Cluster administrators	51
	Controlling daemons	52
	Controlling mbatchd	54
	Customize batch command messages	56
	Reconfiguring your cluster	57
4	Working with Hosts	59
	Contents	59
	Host status	60
	How LIM Determines Host Models and Types	62
	Viewing Host Information	64
	Controlling Hosts	70
	Adding a Host	73
	Remove a Host	75
	Remove a Host from Master Candidate List	76
	Adding Hosts Dynamically	77
	Automatically Detect Operating System Types and Versions	84
	Add Host Types and Host Models to Isf.shared	86
	Registering Service Ports	87

Contents

	Host Naming	89
	Hosts with Multiple Addresses	91
	Using IPv6 Addresses	93
	Specify host names with condensed notation	95
	Host Groups	96
	Compute Units	100
	Tuning CPU Factors	103
	Handling Host-level Job Exceptions	105
5	Working with Queues	107
	Contents	107
	Queue States	107
	Viewing Queue Information	108
	Control Queues	110
	Add and Remove Queues	113
	Manage Queues	115
	Handling Job Exceptions in Queues	116
6	Managing Jobs	119
	Contents	119
	Understanding Job States	120
	View Job Information	123
	Changing Job Order Within Queues	127
	Switch Jobs from One Queue to Another	128
	Forcing Job Execution	129
	Suspending and Resuming Jobs	130
	Killing Jobs	131
	Sending a Signal to a Job	133
	Using Job Groups	134
	Handling Job Exceptions	145
7	Managing Users and User Groups	149
	Contents	149
	Viewing User and User Group Information	149
	About User Groups	151
	Existing User Groups as LSF User Groups	151
	LSF User Groups	152
8	Platform LSF Licensing	155
	Contents	155
	The LSF License File	156
	How LSF Permanent Licensing Works	160
	Installing a Demo License	162
	Installing a Permanent License	164
	Updating a License	170
	FLEXlm Basics	172
	Multiple FLEXlm License Server Hosts	175
	Partial Licensing	177
	Floating Client Licenses	181
	Troubleshooting License Issues	187

9	Managing LSF on Platform EGO	191
	Contents	191
	About LSF on Platform EGO	192
	LSF and EGO directory structure	195
	Configuring LSF and EGO	199
	Managing LSF daemons through EGO	202
	EGO control of PMC and PERF services	204
	Administrative Basics	205
	Logging and troubleshooting	206
	Frequently asked questions	213
10	The Platform Management Console	215
	Contents	215
	Log on to the Platform Management Console	215
	Set the command-line environment	216
	Manage services	216
	Manage hosts	218
	Customize job submission interfaces	219
	Remove a job submission interface	219
11	Cluster Version Management and Patching on UNIX and Linux	221
	Contents	221
	Scope	222
	Patch installation interaction diagram	223
	Patch rollback interaction diagram	224
	Version management components	225
	Version management concepts	227
	Cluster patching behavior table	229
	Cluster rollback behavior table	230
	Version management files	230
	Version management commands	231
	Installing update releases on UNIX and Linux	232
	Installing fixes on UNIX and Linux	233
	Rolling back patches on UNIX and Linux	233
	Patching the Oracle database	234
	Patching the Derby database	235
12	Upgrading Platform LSF HPC	237
	Contents	237
	Upgrade Platform LSF HPC	237
	What lsinstall does	237

Part II: Working with Resources

13	Understanding Resources	245
	Contents	245
	About LSF Resources	246
	How Resources are Classified	248

Contents

How LSF Uses Resources	251
Load Indices	253
Static Resources	257
Automatic Detection of Hardware Reconfiguration	264
Set the external static LIM	265
14 Adding Resources	267
Contents	267
About Configured Resources	268
Add New Resources to Your Cluster	269
Configuring Isf.shared Resource Section	270
Configuring Isf.cluster.cluster_name Host Section	272
Configuring Isf.cluster.cluster_name ResourceMap Section	273
Static Shared Resource Reservation	275
External Load Indices	276
Modifying a Built-In Load Index	276
15 Managing Software Licenses with LSF	277
Contents	277
Using Licensed Software with LSF	277
Host-locked Licenses	277
Counted Host-Locked Licenses	277
Network Floating Licenses	278

Part III: Job Scheduling Policies

16 Time Syntax and Configuration	285
Contents	285
Specifying Time Values	285
Specifying Time Windows	285
Specifying Time Expressions	286
Using Automatic Time-based Configuration	287
17 Deadline Constraint and Exclusive Scheduling	291
Contents	291
Using Deadline Constraint Scheduling	291
Using Exclusive Scheduling	292
18 Preemptive Scheduling	295
Contents	295
About Preemptive Scheduling	295
19 Specifying Resource Requirements	297
Contents	297
About Resource Requirements	298
Queue-level Resource Requirements	300
Job-level Resource Requirements	302
About Resource Requirement Strings	304
Selection String	310
Order String	318

Usage String	320
Span String	327
Same String	329
Compute Unit String	331
20 Fairshare Scheduling	335
Contents	335
Understanding Fairshare Scheduling	336
User Share Assignments	337
Dynamic User Priority	338
How Fairshare Affects Job Dispatch Order	340
Host Partition User-based Fairshare	341
Queue-level User-based Fairshare	343
Cross-queue User-based Fairshare	343
Hierarchical User-based Fairshare	347
Queue-based Fairshare	350
Configuring Slot Allocation per Queue	352
View Queue-based Fairshare Allocations	354
Typical Slot Allocation Scenarios	355
Using Historical and Committed Run Time	360
Users Affected by Multiple Fairshare Policies	363
Ways to Configure Fairshare	364
Resizable jobs and fairshare	367
21 Resource Preemption	369
Contents	369
About Resource Preemption	370
Requirements for Resource Preemption	371
Custom Job Controls for Resource Preemption	371
Resource Preemption Steps	373
Configure Resource Preemption	375
License Preemption Example	377
Memory Preemption Example	379
22 Goal-Oriented SLA-Driven Scheduling	383
Contents	383
Using Goal-Oriented SLA Scheduling	383
Configuring Service Classes for SLA Scheduling	386
View Information about SLAs and Service Classes	388
Understanding Service Class Behavior	392

Part IV: Job Scheduling and Dispatch

23 Working with Application Profiles	401
Contents	401
Manage application profiles	402
Use application profiles	405
View application profile information	407

Contents

	How application profiles interact with queue and job parameters	411
24	Resource Allocation Limits	419
	Contents	419
	About Resource Allocation Limits	420
	Configuring Resource Allocation Limits	425
	Viewing Information about Resource Allocation Limits	434
25	Reserving Resources	437
	Contents	437
	About Resource Reservation	437
	Using Resource Reservation	438
	Memory Reservation for Pending Jobs	440
	Time-based Slot Reservation	442
	Viewing Resource Reservation Information	450
26	Advance Reservation	453
	Contents	453
	Understanding Advance Reservations	454
	Configure Advance Reservation	456
	Using Advance Reservation	458
27	Dispatch and Run Windows	477
	Contents	477
	Dispatch and Run Windows	477
	Run Windows	477
	Dispatch Windows	478
28	Job Dependencies	481
	Contents	481
	Job Dependency Terminology	481
	Job Dependency Scheduling	482
	Dependency Conditions	484
	View Job Dependencies	486
29	Job Priorities	489
	Contents	489
	User-Assigned Job Priority	490
	Automatic Job Priority Escalation	492
	Absolute Job Priority Scheduling	493
30	Job Requeue and Job Rerun	503
	Contents	503
	About Job Requeue	504
	Automatic Job Requeue	505
	Job-level automatic requeue	507
	Reverse Requeue	508
	Exclusive Job Requeue	509
	User-Specified Job Requeue	510
	Automatic Job Rerun	511
31	Job Checkpoint, Restart, and Migration	513
	Contents	513

	Checkpoint and restart options	514
	Checkpoint directory and files	514
	Checkpoint and restart executables	516
	Job restart	516
	Job migration	517
32	Chunk Job Dispatch	519
	Contents	519
	About Job Chunking	519
	Configure Chunk Job Dispatch	520
	Submitting and Controlling Chunk Jobs	522
33	Job Arrays	525
	Contents	525
	Create a Job Array	525
	Handling Input and Output Files	527
	Redirecting Standard Input and Output	527
	Passing Arguments on the Command Line	528
	Job Array Dependencies	529
	Monitoring Job Arrays	529
	Individual job status	530
	Specific job status	531
	Controlling Job Arrays	531
	Job Array Chunking	532
	Requeuing a Job Array	534
	Job Array Job Slot Limit	535
34	Running Parallel Jobs	537
	Contents	537
	How LSF Runs Parallel Jobs	538
	Preparing Your Environment to Submit Parallel Jobs to LSF	539
	Submitting Parallel Jobs	540
	Starting Parallel Tasks with LSF Utilities	541
	Job Slot Limits For Parallel Jobs	543
	Specifying a Minimum and Maximum Number of Processors	544
	Specifying a First Execution Host	545
	Controlling Job Locality using Compute Units	547
	Controlling Processor Allocation Across Hosts	554
	Running Parallel Processes on Homogeneous Hosts	557
	Limiting the Number of Processors Allocated	559
	Reserving Processors	562
	Reserving Memory for Pending Parallel Jobs	564
	Backfill Scheduling: Allowing Jobs to Use Reserved Job Slots	565
	Parallel Fairshare	574
	How Deadline Constraint Scheduling Works For Parallel Jobs	575
	Optimized Preemption of Parallel Jobs	576
	Processor Binding for Parallel Jobs	576
	Making Job Allocations Resizable	578
35	Submitting Jobs Using JSDL	581

Contents

Contents	581
Why Use JSDL?	581
Using JSDL Files with LSF	581
Collecting resource values using elim.jsdl	590

Part V: Controlling Job Execution

36	Runtime Resource Usage Limits	595
	Contents	595
	About Resource Usage Limits	595
	Specifying Resource Usage Limits	599
	Supported Resource Usage Limits and Syntax	601
	Examples	606
	CPU Time and Run Time Normalization	607
	PAM resource limits	608
37	Load Thresholds	609
	Contents	609
	Automatic Job Suspension	610
	Suspending Conditions	612
38	Pre-Execution and Post-Execution Commands	615
	Contents	615
	About Pre-Execution and Post-Execution Commands	616
	Configuring Pre- and Post-Execution Commands	618
39	Job Starters	625
	Contents	625
	About Job Starters	625
	Command-Level Job Starters	626
	Queue-Level Job Starters	628
	Controlling Execution Environment Using Job Starters	629
40	External Job Submission and Execution Controls	631
	Contents	631
	Understanding External Executables	631
	Using esub	632
	Existing esub	640
	Working with eexec	641
41	Configuring Job Controls	643
	Contents	643
	Default Job Control Actions	643
	Configuring Job Control Actions	645
	Customizing Cross-Platform Signal Conversion	648

Part VI: Interactive Jobs

42	Interactive Jobs with bsub	651
-----------	-----------------------------------	------------

Contents	651
About Interactive Jobs	651
Submitting Interactive Jobs	652
Performance Tuning for Interactive Batch Jobs	654
Interactive Batch Job Messaging	657
Running X Applications with bsub	658
Writing Job Scripts	659
Registering utmp File Entries for Interactive Batch Jobs	661
43 Running Interactive and Remote Tasks	663
Contents	663
Running Remote Tasks	663
Interactive Tasks	666
Load Sharing Interactive Sessions	668
Load Sharing X Applications	668

Part VII: Monitoring Your Cluster

44 Achieving Performance and Scalability	673
Contents	673
Optimizing Performance in Large Sites	673
Tuning UNIX for Large Clusters	674
Tuning LSF for Large Clusters	675
Monitoring Performance Metrics in Real Time	686
45 Reporting	691
Contents	691
Introduction to Reporting	691
Getting Started with Standard Reports	692
Custom Reports	694
System Description	698
Reports Administration	700
Test the Reporting Feature	712
Disable the Reporting Feature	713
Move to a Production Database	714
46 Event Generation	719
Contents	719
Event Generation	719
Enabling event generation	719
Events list	720
Arguments passed to the LSF event program	720
47 Tuning the Cluster	723
Contents	723
Tuning LIM	724
Improving performance of mbatchd query requests on UNIX	731
48 Authentication and Authorization	737
Contents	737

Contents

	Authentication options	737
	Authorization options	739
49	Job Email and Job File Spooling	745
	Contents	745
	Mail Notification When a Job Starts	745
	File Spooling for Job Input, Output, and Command Files	748
	Specifying a job command file (bsub -Zs)	749
	About the job spooling directory (JOB_SPOOL_DIR)	749
	Modifying the job input file	750
	Modifying the job command file	750
	For more information	750
50	Non-Shared File Systems	751
	Contents	751
	About Directories and Files	751
	Using LSF with Non-Shared File Systems	752
	Remote File Access	752
	File Transfer Mechanism (lsrctp)	754
51	Error and Event Logging	757
	Contents	757
	System Directories and Log Files	757
	Managing Error Logs	758
	System Event Log	760
	Duplicate Logging of Event Logs	761
	LSF Job Termination Reason Logging	762
	Understanding LSF job exit codes	766
52	Troubleshooting and Error Messages	769
	Contents	769
	Shared File Access	770
	Common LSF Problems	771
	Error Messages	778
	Setting Daemon Message Log to Debug Level	785
	Setting Daemon Timing Levels	788

Part VIII: LSF Utilities

53	Using Istcsh	793
	Contents	793
	About Istcsh	793
	Differences from Other Shells	795
	Limitations	795
	Starting Istcsh	796
	Using Istcsh as Your Login Shell	796
	Host Redirection	797
	Task Control	798
	Built-in Commands	798

Writing Shell Scripts in lscsh	800
Index	803

Contents

About Platform LSF

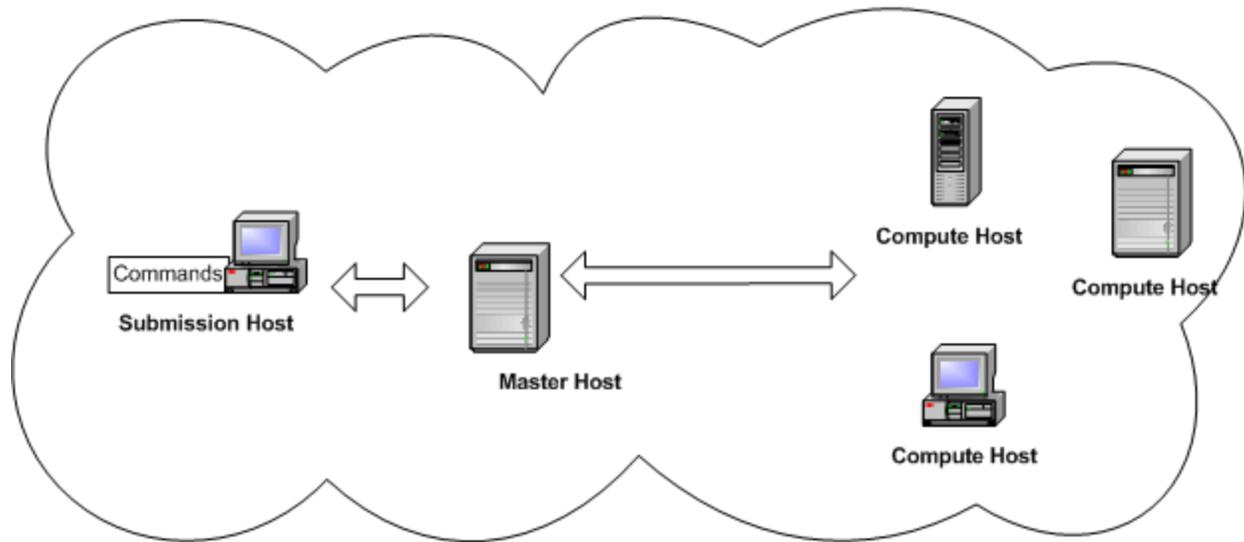
Contents

- ◆ [Learn about Platform LSF](#) on page 16
- ◆ [Cluster Concepts](#) on page 16
- ◆ [Job Life Cycle](#) on page 28

Learn about Platform LSF

Before using Platform LSF for the first time, you should download and read LSF Version 7 Release Notes for the latest information about what's new in the current release and other important information.

Cluster Concepts



Clusters, jobs, and queues

Cluster

A group of computers (hosts) running LSF that work together as a single unit, combining computing power and sharing workload and resources. A cluster provides a single-system image for disparate computing resources.

Hosts can be grouped into clusters in a number of ways. A cluster could contain:

- ◆ All the hosts in a single administrative group
- ◆ All the hosts on one file server or sub-network
- ◆ Hosts that perform similar functions

Commands:

- ◆ `lshosts`—View static resource information about hosts in the cluster
- ◆ `bhosts`—View resource and job information about server hosts in the cluster
- ◆ `lsid`—View the cluster name
- ◆ `lsclusters`—View cluster status and size

Configuration:

- ◆ Define hosts in your cluster in `lsf.cluster.cluster_name`

TIP: The name of your cluster should be unique. It should not be the same as any host or queue.

Job

A unit of work run in the LSF system. A job is a command submitted to LSF for execution. 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.

Commands:

- ◆ `bjobs`—View jobs in the system
- ◆ `bsub`—Submit jobs

Job slot

A job slot is a bucket into which a single unit of work is assigned in the LSF system.

If hosts are configured with a number of job slots, you can dispatch jobs from queues until all the job slots are filled.

Commands:

- ◆ `bhosts`—View job slot limits for hosts and host groups
- ◆ `bqueues`—View job slot limits for queues
- ◆ `busers`—View job slot limits for users and user groups

Configuration:

- ◆ Define job slot limits in `lsb.resources`.

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`—`mbatchd` has lost contact with `sbatchd` 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

Queue

A clusterwide container for jobs. All jobs wait in queues until they are scheduled and dispatched to hosts.

Queues do not correspond to individual hosts; each queue can use all server hosts in the cluster, or a configured subset of the server hosts.

When you submit a job to a queue, you do not need to specify an execution host. LSF dispatches the job to the best available execution host in the cluster to run that job.

Queues implement different job scheduling and control policies.

Commands:

- ◆ `bqueues` — View available queues
- ◆ `bsub -q` — Submit a job to a specific queue
- ◆ `bparams` — View default queues

Configuration:

- ◆ Define queues in `lsb.queues`

TIP: The names of your queues should be unique. They should not be the same as the cluster name or any host in the cluster.

First-come, first-served (FCFS) scheduling

The default type of scheduling in LSF. Jobs are considered for dispatch based on their order in the queue.

Hosts

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.

Commands:

- ◆ `lslload` — View load on hosts
- ◆ `lshosts` — View configuration information about hosts in the cluster including number of CPUs, model, type, and whether the host is a client or server
- ◆ `bhosts` — View batch server hosts in the cluster

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.

Submission host

The host where jobs are submitted to the cluster.

Jobs are submitted using the `bsub` command or from an application that uses the LSF API.

Client hosts and server hosts can act as submission hosts.

Commands:

- ◆ `bsub` — Submit a job
- ◆ `bjobs` — View jobs that are submitted

Execution host

The host where a job runs. Can be the same as the submission host. All execution hosts are server hosts.

Commands:

- ◆ `bjobs` — View where a job runs

Server host

Hosts that are capable of submitting and executing jobs. A server host runs `sbatchd` to execute server requests and apply local policies.

An LSF cluster may consist of static and dynamic hosts. Dynamic host configuration allows you to add and remove hosts without manual reconfiguration. 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.

Commands:

- ◆ `lshosts` — View hosts that are servers (`server=Yes`)

Configuration:

- ◆ Server hosts are defined in the `lsf.cluster.cluster_name` file by setting the value of `server` to 1

Client host

Hosts that are only capable of submitting jobs to the cluster. Client hosts run LSF commands and act only as submission hosts. Client hosts do not execute jobs or run LSF daemons.

Commands:

- ◆ `lshosts` — View hosts that are clients (`server=No`)

Configuration:

- ◆ Client hosts are defined in the `lsf.cluster.cluster_name` file by setting the value of `server` to 0

Floating client host

In LSF, you can have both client hosts and floating client hosts. The difference is in the type of license purchased.

If you purchased a regular (fixed) client license, LSF client hosts are static. The 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.

If you purchased a floating client license, LSF floating client hosts are dynamic. They are not listed in `lsf.cluster.cluster_name`. Since LSF does not take into account the host name but the number of floating licenses, clients can change dynamically and licenses will be distributed to clients that request to use LSF.

When you submit a job from any unlicensed host, and if there are any floating licenses free, the host will check out a license and submit your job to LSF. However, once a host checks out a floating 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 floating client license becomes available to any other host that needs it.

Master host

Where the master LIM and `mbatchd` run. An LSF server host that acts as the overall coordinator for that cluster. Each cluster has one master host to do all job scheduling and dispatch. If the master host goes down, another master candidate LSF server in the cluster becomes the master host.

All LSF daemons run on the master host. The LIM on the master host is the master LIM.

Commands:

- ◆ `lsid`—View the master host name

Configuration:

- ◆ The master host is defined along with other candidate master hosts by `LSF_MASTER_LIST` in `lsf.conf`.

LSF daemons

LSF daemon	Role
<code>mbatchd</code>	Job requests and dispatch
<code>mbschd</code>	Job scheduling
<code>sbatchd</code>	Job execution
<code>res</code>	
<code>lim</code>	Host information
<code>pim</code>	Job process information
<code>elim</code>	Collect and track custom dynamic load indices

`mbatchd`

Master Batch Daemon running on the master host. Started by `sbatchd`. Responsible for the overall state of jobs in the system.

Receives job submission, and information query requests. Manages jobs held in queues. Dispatches jobs to hosts as determined by `mbschd`.

Commands:

- ◆ `badmin mbdrestart`—Starts `sbatchd`

Configuration:

- ◆ Port number defined in `lsf.conf`.

`mbschd`

Master Batch Scheduler Daemon running on the master host. Works with `mbatchd`. Started by `mbatchd`.

Makes scheduling decisions based on job requirements and policies.

`sbatchd`

Slave Batch Daemon running on each server host. Receives the request to run the job from `mbatchd` and manages local execution of the job. Responsible for enforcing local policies and maintaining the state of jobs on the host.

`sbatchd` forks a child `sbatchd` for every job. The child `sbatchd` runs an instance of `res` to create the execution environment in which the job runs. The child `sbatchd` exits when the job is complete.

Commands:

- ◆ `badmin hstartup`—Starts `sbatchd`

- ◆ `badmin hshutdown`—Shuts down `sbatchd`
- ◆ `badmin hrestart`—Restarts `sbatchd`

Configuration:

- ◆ Port number defined in `lsf.conf`

res

Remote Execution Server (RES) running on each server host. Accepts remote execution requests to provide transparent and secure remote execution of jobs and tasks.

Commands:

- ◆ `lsadmin resstartup`—Starts `res`
- ◆ `lsadmin resshutdown`—Shuts down `res`
- ◆ `lsadmin resrestart`—Restarts `res`

Configuration:

- ◆ Port number defined in `lsf.conf`

lim

Load Information Manager (LIM) running on each server host. Collects host load and configuration information and forwards it to the master LIM running on the master host. Reports the information displayed by `lsload` and `lshosts`.

Static indices are reported when the LIM starts up or when the number of CPUs (`ncpus`) change. Static indices are:

- ◆ Number of CPUs (`ncpus`)
- ◆ Number of disks (`ndisks`)
- ◆ Total available memory (`maxmem`)
- ◆ Total available swap (`maxswp`)
- ◆ Total available temp (`maxtmp`)
- ◆ The number of physical processors per CPU configured on a host (`nprocs`).
- ◆ The number of cores per processor configured on a host (`ncores`).
- ◆ The number of threads per core configured on a host (`nthreads`).

Dynamic indices for host load collected at regular intervals are:

- ◆ Hosts status (`status`)
- ◆ 15 second, 1 minute, and 15 minute run queue lengths (`r15s`, `r1m`, and `r15m`)
- ◆ CPU utilization (`ut`)
- ◆ Paging rate (`pg`)
- ◆ Number of login sessions (`ls`)
- ◆ Interactive idle time (`it`)
- ◆ Available swap space (`swp`)
- ◆ Available memory (`mem`)
- ◆ Available temp space (`tmp`)
- ◆ Disk IO rate (`io`)

Commands:

Cluster Concepts

- ◆ `lsadmin limstartup`— Starts LIM
- ◆ `lsadmin limshutdown`— Shuts down LIM
- ◆ `lsadmin limrestart`— Restarts LIM
- ◆ `lsload`— View dynamic load values
- ◆ `lshosts`— View static host load values

Configuration:

- ◆ Port number defined in `lsf.conf`.

Master LIM

The LIM running on the master host. Receives load information from the LIMs running on hosts in the cluster.

Forwards load information to `mbatchd`, which forwards this information to `mbschd` to support scheduling decisions. If the master LIM becomes unavailable, a LIM on another host automatically takes over.

Commands:

- ◆ `lsadmin limstartup`— Starts LIM
- ◆ `lsadmin limshutdown`— Shuts down LIM
- ◆ `lsadmin limrestart`— Restarts LIM
- ◆ `lsload`— View dynamic load values
- ◆ `lshosts`— View static host load values

Configuration:

- ◆ Port number defined in `lsf.conf`.

ELIM

External LIM (ELIM) is a site-definable executable that collects and tracks custom dynamic load indices. An ELIM can be a shell script or a compiled binary program, which returns the values of the dynamic resources you define. The ELIM executable must be named `elim.anything` and located in `LSF_SERVERDIR`.

pim

Process Information Manager (PIM) running on each server host. Started by LIM, which periodically checks on PIM and restarts it if it dies.

Collects information about job processes running on the host such as CPU and memory used by the job, and reports the information to `sbatchd`.

Commands:

- ◆ `bjobs`— View job information

Batch jobs and tasks

You can either run jobs through the batch system where jobs are held in queues, or you can interactively run tasks without going through the batch system, such as tests for example.

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.

Commands:

- ◆ `bjobs`—View jobs in the system
- ◆ `bsub`—Submit jobs

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.

The `bsub` command stops display of output from the shell until the job completes, and no mail is sent to you by default. Use `Ctrl-C` at any time to terminate the job.

Commands:

- ◆ `bsub -I`—Submit an interactive job

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.

Commands:

- ◆ `lrun`—Submit an interactive task
- ◆ `lsgun`—Submit an interactive task to a group of hosts
- ◆ See also LSF utilities such as `ch`, `lsacct`, `lsacctmrg`, `lslogin`, `lspplace`, `lsload`, `lsloadadj`, `lselectible`, `lsmom`, `lstcsh`

Local task

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

Commands:

- ◆ `lsltasks`—View and add tasks

Configuration:

- ◆ `lsf.task`—Configure systemwide resource requirements for tasks
- ◆ `lsf.task.cluster`—Configure clusterwide resource requirements for tasks
- ◆ `.lsftask`—Configure user-specific tasks

Remote task

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

Commands:

- ◆ `lsrtasks`—View and add tasks

Configuration:

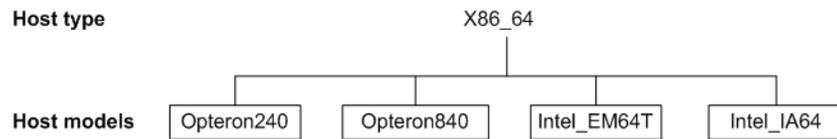
- ◆ `lsf.task`—Configure systemwide resource requirements for tasks

- ◆ `lsf.task.cluster`—Configure clusterwide resource requirements for tasks
- ◆ `.lsftask`—Configure user-specific tasks

Host types and host models

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

The following example host 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.

Commands:

- ◆ `lsinfo -t`—View all host types defined in `lsf.shared`

Configuration:

- ◆ Defined in `lsf.shared`
- ◆ Mapped to hosts in `lsf.cluster.cluster_name`

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.

Commands:

- ◆ `lsinfo -m`—View a list of currently running models
- ◆ `lsinfo -M`—View all models defined in `lsf.shared`

Configuration:

- ◆ Defined in `lsf.shared`
- ◆ Mapped to hosts in `lsf.cluster.cluster_name`

Users and administrators

LSF user

A user account that has permission to submit jobs to the LSF cluster.

LSF administrator

In general, you must be an LSF administrator to perform operations that will affect other LSF users. Each cluster has one primary LSF administrator, specified during LSF installation. You can also configure additional administrators at the cluster level and at the queue level.

Primary LSF administrator	The first cluster administrator specified during installation and first administrator listed in <code>lsf.cluster.cluster_name</code> . 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.
Cluster administrator	<p>May be specified during LSF installation or configured 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 necessarily have permission to change LSF configuration files.</p> <p>For example, a cluster administrator can create an LSF host group, submit a job to any queue, or terminate another user's job.</p>
Queue administrator	An LSF administrator user account that has administrative permissions limited to a specified queue. For example, an LSF queue administrator can perform administrative operations on the specified queue, or on jobs running in the specified queue, but cannot change LSF configuration or operate on LSF daemons.
Resources	
Resource usage	<p>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.</p> <p>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.</p> <p>LSF collects information such as:</p> <ul style="list-style-type: none"> ◆ 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 <p>On UNIX, job-level resource usage is collected through PIM.</p> <p>Commands:</p> <ul style="list-style-type: none"> ◆ <code>lsinfo</code>: View the resources available in your cluster ◆ <code>bjobs -l</code>: View current resource usage of a job <p>Configuration:</p> <ul style="list-style-type: none"> ◆ <code>SBD_SLEEP_TIME</code> in <code>lsb.params</code>: Configures how often resource usage information is sampled by PIM, collected by <code>sbatchd</code>, and sent to <code>mbatchd</code> ◆ <code>LSF_PIM_LINUX_ENHANCE</code> in <code>lsf.conf</code>: Enables the enhanced PIM, obtaining exact memory usage of processes using shared memory on Linux operating systems. By default memory shared between jobs may be counted more than once.

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.

Commands:

- ◆ `lslload -l`— View all load indices
- ◆ `bhosts -l`— View load levels on a host

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.

Commands:

- ◆ `lsinfo`— View external load indices

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.

Commands:

- ◆ `bhosts -l`— View suspending conditions for hosts
- ◆ `bqueues -l`— View suspending conditions for queues
- ◆ `bjobs -l`— View suspending conditions for a particular job and the scheduling thresholds that control when a job is resumed

Configuration:

- ◆ `lsb.hosts`— Configure thresholds for hosts
- ◆ `lsb.queues`— Configure thresholds for queues

Runtime resource usage limits

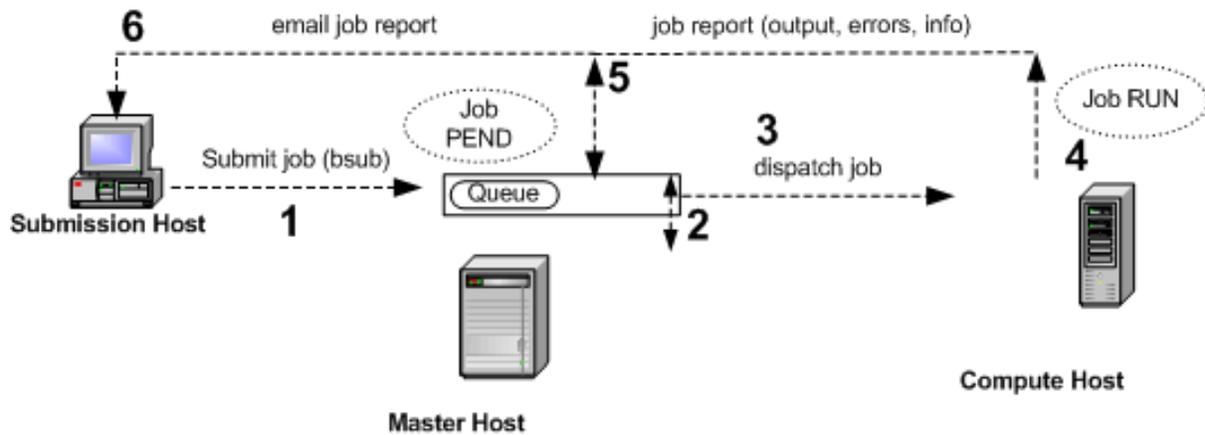
Limit the use of resources while a job is running. Jobs that consume more than the specified amount of a resource are signalled.

Configuration:

- ◆ `lsb.queues`— Configure resource usage limits for queues

- 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.
- Configuration:
- ◆ `lsb.resources` — Configure queue-level resource allocation limits for hosts, users, queues, and projects
- 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.
- Commands:
- ◆ `bsub -R` — Specify resource requirement string for a job
- Configuration:
- ◆ `lsb.queues` — Configure resource requirements for queues

Job Life Cycle



1 Submit a job

You submit a job from an LSF client or server with the `bsub` command.

If you do not specify a queue when submitting the job, the job is submitted to the default queue.

Jobs are held in a queue waiting to be scheduled and have the PENDING state. The job is held in a job file in the `LSF_SHAREDIR/cluster_name/logdir/info/` directory, or in one of its subdirectories if `MAX_INFO_DIRS` is defined in `lsb.params`.

Job ID

LSF assigns each job a unique job ID when you submit the job.

Job name

You can also assign a name to the job with the `-J` option of `bsub`. Unlike the job ID, the job name is not necessarily unique.

2 Schedule job

- `mbatchd` looks at jobs in the queue and sends the jobs for scheduling to `mbschd` at a preset time interval (defined by the parameter `JOB_SCHEDULING_INTERVAL` in `lsb.params`).
- `mbschd` evaluates jobs and makes scheduling decisions based on:
 - ❖ Job priority
 - ❖ Scheduling policies
 - ❖ Available resources
- `mbschd` selects the best hosts where the job can run and sends its decisions back to `mbatchd`.

Resource information is collected at preset time intervals by the master LIM from LIMs on server hosts. The master LIM communicates this information to `mbatchd`, which in turn communicates it to `mbschd` to support scheduling decisions.

3 Dispatch job

As soon as `mbatchd` receives scheduling decisions, it immediately dispatches the jobs to hosts.

4 Run job

`sbatchd` handles job execution. It:

- 1 Receives the request from `mbatchd`
- 2 Creates a child `sbatchd` for the job
- 3 Creates the execution environment
- 4 Starts the job using `res`

The execution environment is copied from the submission host to the execution host and includes the following:

- ❖ Environment variables needed by the job
- ❖ Working directory where the job begins running
- ❖ Other system-dependent environment settings, for example:
 - ❖ On UNIX, resource limits and `umask`
 - ❖ On Windows, desktop and Windows root directory

The job runs under the user account that submitted the job and has the status `RUN`.

5 Return output

When a job is completed, it is assigned the `DONE` status if the job was completed without any problems. The job is assigned the `EXIT` status if errors prevented the job from completing.

`sbatchd` communicates job information including errors and output to `mbatchd`.

6 Send email to client

`mbatchd` returns the job output, job error, and job information to the submission host through email. Use the `-o` and `-e` options of `bsub` to send job output and errors to a file.

Job report

A job report is sent by email to the LSF job owner and includes:

- ◆ Job information such as:
 - ❖ CPU use
 - ❖ Memory use
 - ❖ Name of the account that submitted the job
- ◆ Job output
- ◆ Errors

Job Life Cycle

How the System Works

LSF can be configured in different ways that affect the scheduling of jobs. By default, this is how LSF handles a new job:

- 1 Receive the job. Create a job file. Return the job ID to the user.
- 2 Schedule the job and select the best available host.
- 3 Dispatch the job to a selected host.
- 4 Set the environment on the host.
- 5 Start the job.

Contents

- ◆ [Job Submission](#) on page 32
- ◆ [Job Scheduling and Dispatch](#) on page 34
- ◆ [Host Selection](#) on page 36
- ◆ [Job Execution Environment](#) on page 37
- ◆ [Fault Tolerance](#) on page 38

Job Submission

The life cycle of a job starts when you submit the job to LSF. On the command line, `bsub` is used to submit jobs, and you can specify many options to `bsub` to modify the default behavior, including the use of a JSDL file. Jobs must be submitted to a queue.

Queues

Queues represent a set of pending jobs, lined up in a defined order and waiting for their opportunity to use resources. Queues implement different job scheduling and control policies. All jobs submitted to the same queue share the same scheduling and control policy. Queues do not correspond to individual hosts; each queue can use all server hosts in the cluster, or a configured subset of the server hosts.

A queue is a network-wide holding place for jobs. Jobs enter the queue via the `bsub` command. LSF can be configured to have one or more default queues. Jobs that are not submitted to a specific queue will be assigned to the first default queue that accepts them. Queues have the following attributes associated with them:

- ◆ Priority, where a larger integer is a higher priority
- ◆ Name, which uniquely identifies the queue
- ◆ Queue limits, that restrict hosts, number of jobs, users, groups, processors, etc.
- ◆ Standard UNIX limits: memory, swap, process, CPU, etc.
- ◆ Scheduling policies: FCFS, fairshare, preemptive, exclusive
- ◆ Administrators
- ◆ Run conditions
- ◆ Load-sharing threshold conditions, which apply load sharing to the queue
- ◆ UNIX `nice(1)` value, which sets the UNIX scheduler priority

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
```

Queue priority

Defines the order in which queues are searched to determine which job will be processed. Queues are assigned a priority by the LSF administrator, where a higher number has a higher priority. Queues are serviced by LSF in order of priority from the highest to the lowest. If multiple queues have the same priority, LSF schedules all the jobs from these queues in first-come, first-served order.

Automatic queue selection

Typically, a cluster has multiple queues. When you submit a job to LSF you might define which queue the job will enter. If you submit a job without specifying a queue name, LSF considers the requirements of the job and automatically chooses a suitable queue from a list of candidate default queues. If you did not define any candidate default queues, LSF will create a new queue using all the default settings, and submit the job to that queue.

Viewing default queues

Use `bparams` to display default queues:

```
bparams
Default Queues: normal
...
```

The user can override this list by defining the environment variable `LSB_DEFAULTQUEUE`.

How automatic queue selection works

LSF selects a suitable queue according to:

- ◆ User access restriction—Queues that do not allow this user to submit jobs are not considered.
- ◆ Host restriction—If the job explicitly specifies a list of hosts on which the job can be run, then the selected queue must be configured to send jobs to all hosts in the list.
- ◆ Queue status—Closed queues are not considered.
- ◆ Exclusive execution restriction—If the job requires exclusive execution, then queues that are not configured to accept exclusive jobs are not considered.
- ◆ Job's requested resources—These must be within the resource allocation limits of the selected queue.

If multiple queues satisfy the above requirements, then the first queue listed in the candidate queues (as defined by the `DEFAULT_QUEUE` parameter or the `LSB_DEFAULTQUEUE` environment variable) that satisfies the requirements is selected.

Job files

When a batch job is submitted to a queue, LSF Batch 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.

Job Scheduling and Dispatch

Submitted jobs sit in queues until they are scheduled and dispatched to a host for execution. When a job is submitted to LSF, many factors control when and where the job starts to run:

- ◆ Active time window of the queue or hosts
- ◆ Resource requirements of the job
- ◆ Availability of eligible hosts
- ◆ Various job slot limits
- ◆ Job dependency conditions
- ◆ Fairshare constraints
- ◆ Load conditions

Scheduling policies

First-Come, First-Served (FCFS) scheduling

By default, jobs in a queue are dispatched in first-come, first-served (FCFS) order. This means that jobs are dispatched according to their order in the queue. Since jobs are ordered according to job priority, this does not necessarily mean that jobs will be dispatched in the order of submission. The order of jobs in the queue can also be modified by the user or administrator.

Service level agreement (SLA) scheduling

An SLA in LSF is a “just-in-time” scheduling policy that defines an agreement between LSF administrators and LSF users. The SLA scheduling policy defines how many jobs should be run from each SLA to meet the configured goals.

Fairshare scheduling and other policies

If a fairshare scheduling policy has been specified for the queue or if host partitions have been configured, jobs are dispatched in accordance with these policies instead. To solve diverse problems, LSF allows multiple scheduling policies in the same cluster. LSF has several queue scheduling policies such as exclusive, preemptive, fairshare, and hierarchical fairshare.

Scheduling and dispatch

Jobs are scheduled at regular intervals (5 seconds by default, configured by the parameter `JOB_SCHEDULING_INTERVAL` in `lsb.params`). Once jobs are scheduled, they can be immediately dispatched to hosts.

To prevent overloading any host, LSF waits a short time between dispatching jobs to the same host. The delay is configured by the `JOB_ACCEPT_INTERVAL` parameter in `lsb.params` or `lsb.queues`. `JOB_ACCEPT_INTERVAL` controls the number of seconds to wait after dispatching a job to a host before dispatching a second job to the same host. The default is 60 seconds. If `JOB_ACCEPT_INTERVAL` is set to zero, more than one job can be started on a host at a time.

For large clusters, define `LSF_SERVER_HOSTS` in `lsf.conf` to decrease the load on the master LIM.

Some operating systems, such as Linux and AIX, let you increase the number of file descriptors that can be allocated to 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 the parameter `LSB_MAX_JOB_DISPATCH_PER_SESSION` in `lsf.conf` to a value greater than 300 and less than or equal to one-half the value of `MAX_SBD_CONNS` defined in `lsb.params`. `LSB_MAX_JOB_DISPATCH_PER_SESSION` defines the maximum number of jobs that `mbatchd` can dispatch during one job scheduling session. You must restart `mbatchd` and `sbatchd` when you change the value of this parameter for the change to take effect.

Dispatch order

Jobs are not necessarily dispatched in order of submission.

Each queue has a priority number set by an LSF Administrator when the queue is defined. LSF tries to start jobs from the highest priority queue first.

By default, LSF considers jobs for dispatch in the following order:

- ◆ For each queue, from highest to lowest priority. If multiple queues have the same priority, LSF schedules all the jobs from these queues in first-come, first-served order.
- ◆ For each job in the queue, according to FCFS order
- ◆ If any host is eligible to run this job, start the job on the best eligible host, and mark that host ineligible to start any other job until `JOB_ACCEPT_INTERVAL` has passed

Jobs can be dispatched out of turn if pre-execution conditions are not met, specific hosts or resources are busy or unavailable, or a user has reached the user job slot limit.

Viewing job order in queue

Use `bjobs` to see the order in which jobs in a queue will actually be dispatched for the FCFS policy.

Changing job order in queue (`btop` and `bbot`)

Use the `btop` and `bbot` commands to change the job order in the queue.

See [Changing Job Order Within Queues](#) on page 127 for more information.

Host Selection

Each time LSF attempts to dispatch a job, it checks to see which hosts are eligible to run the job. A number of conditions determine whether a host is eligible:

- ◆ Host dispatch windows
- ◆ Resource requirements of the job
- ◆ Resource requirements of the queue
- ◆ Host list of the queue
- ◆ Host load levels
- ◆ Job slot limits of the host

A host is only eligible to run a job if all the conditions are met. If a job is queued and there is an eligible host for that job, the job is placed on that host. If more than one host is eligible, the job is started on the best host based on both the job and the queue resource requirements.

Host load levels

A host is available if the values of the load indices (such as `r1m`, `pg`, `mem`) of the host are within the configured scheduling thresholds. There are two sets of scheduling thresholds: host and queue. If any load index on the host exceeds the corresponding host threshold or queue threshold, the host is not eligible to run any job.

Viewing host load levels

- ◆ Use the `bhosts -l` command to display the host thresholds.
- ◆ Use the `bqueues -l` command to display the queue thresholds.

Eligible hosts

When LSF tries to place a job, it obtains current load information for all hosts.

The load levels on each host are compared to the scheduling thresholds configured for that host in the `Host` section of `lsb.hosts`, as well as the per-queue scheduling thresholds configured in `lsb.queues`.

If any load index exceeds either its per-queue or its per-host scheduling threshold, no new job is started on that host.

Viewing eligible hosts

The `bjobs -lp` command displays the names of hosts that cannot accept a job at the moment together with the reasons the job cannot be accepted.

Resource requirements

Resource requirements at the queue level can also be used to specify scheduling conditions (for example, `r1m<0.4 && pg<3`).

A higher priority or earlier batch job is only bypassed if no hosts are available that meet the requirements of that job.

If a host is available but is not eligible to run a particular job, LSF looks for a later job to start on that host. LSF starts the first job found for which that host is eligible.

Job Execution Environment

When LSF runs your jobs, it tries to make it as transparent to the user as possible. By default, the execution environment is maintained to be as close to the submission environment as possible. LSF will copy the environment from the submission host to the execution host. The execution environment includes the following:

- ◆ Environment variables needed by the job
- ◆ Working directory where the job begins running
- ◆ Other system-dependent environment settings; for example, resource usage limits and `umask`:

Since a network can be heterogeneous, it is often impossible or undesirable to reproduce the submission host's execution environment on the execution host. For example, if home directory is not shared between submission and execution host, LSF runs the job in the `/tmp` on the execution host. If the `DISPLAY` environment variable is something like `Unix:0.0`, or `:0.0`, then it must be processed before using on the execution host. These are automatically handled by LSF.

To change the default execution environment, use:

- ◆ A job starter
- ◆ `bsub -L`

For resource control, LSF also changes some of the execution environment of jobs. These include nice values, resource usage limits, or any other environment by configuring a job starter.

Shared user directories

LSF works best when user home directories are shared across all hosts in the cluster. To provide transparent remote execution, you should share user home directories on all LSF hosts.

To provide transparent remote execution, LSF commands determine the user's current working directory and use that directory on the remote host.

For example, if the command `cc file.c` is executed remotely, `cc` only finds the correct `file.c` if the remote command runs in the same directory.

LSF automatically creates an `.lsbatch` subdirectory in the user's home directory on the execution host. This directory is used to store temporary input and output files for jobs.

Executables and the PATH environment variable

Search paths for executables (the `PATH` environment variable) are passed to the remote execution host unchanged. In mixed clusters, LSF works best when the user binary directories (for example, `/usr/bin`, `/usr/local/bin`) have the same path names on different host types. This makes the `PATH` variable valid on all hosts.

LSF configuration files are normally stored in a shared directory. This makes administration easier. There is little performance penalty for this, because the configuration files are not frequently read.

Fault Tolerance

LSF is designed to continue operating even if some of the hosts in the cluster are unavailable. One host in the cluster acts as the master, but if the master host becomes unavailable another host takes over. LSF is available as long as there is one available host in the cluster.

LSF can tolerate the failure of any host or group of hosts in the cluster. When a host crashes, all jobs running on that host are lost. No other pending or running jobs are affected. Important jobs can be submitted to LSF with an option to automatically restart if the job is lost because of a host failure.

Dynamic master host

The LSF master host is chosen dynamically. If the current master host becomes unavailable, another host takes over automatically. The failover master host is selected from the list defined in `LSF_MASTER_LIST` in `lsf.conf` (specified in `install.config` at installation). The first available host in the list acts as the master. LSF might be unavailable for a few minutes while hosts are waiting to be contacted by the new master.

Running jobs are managed by `sbatchd` on each server host. When the new `mbatchd` starts, it polls the `sbatchd` on each host and finds the current status of its jobs. If `sbatchd` fails but the host is still running, jobs running on the host are not lost. When `sbatchd` is restarted it regains control of all jobs running on the host.

Network failure

If the cluster is partitioned by a network failure, a master LIM takes over on each side of the partition. Interactive load-sharing remains available, as long as each host still has access to the LSF executables.

Event log file (`lsb.events`)

Fault tolerance in LSF depends on the event log file, `lsb.events`, which is kept on the primary file server. Every event in the system is logged in this file, including all job submissions and job and host status changes. If the master host becomes unavailable, a new master is chosen by `lim`. `sbatchd` on the new master starts a new `mbatchd`. The new `mbatchd` reads the `lsb.events` file to recover the state of the system.

For sites not wanting to rely solely on a central file server for recovery information, LSF can be configured to maintain a duplicate event log by keeping a replica of `lsb.events`. The replica is stored on the file server, and used if the primary copy is unavailable. When using LSF's duplicate event log function, the primary event log is stored on the first master host, and re-synchronized with the replicated copy when the host recovers.

Partitioned network

If the network is partitioned, only one of the partitions can access `lsb.events`, so batch services are only available on one side of the partition. A lock file is used to make sure that only one `mbatchd` is running in the cluster.

Host failure

If an LSF server host fails, jobs running on that host are lost. No other jobs are affected. Jobs can be submitted as *rerunnable*, so that they automatically run again from the beginning or as *checkpointable*, so that they start again from a checkpoint on another host if they are lost because of a host failure.

If all of the hosts in a cluster go down, all running jobs are lost. When a host comes back up and takes over as master, it reads the `lsb.events` file to get the state of all batch jobs. Jobs that were running when the systems went down are assumed to have exited, and email is sent to the submitting user. Pending jobs remain in their queues, and are scheduled as hosts become available.

Job exception handling

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 the corresponding actions. By default, LSF does not detect any exceptions.

See [Handling Host-level Job Exceptions](#) on page 105 and [Handling Job Exceptions in Queues](#) on page 116 for more information about job-level exception management.

Fault Tolerance



Managing Your Cluster

- ◆ [Working with Your Cluster](#) on page 43
- ◆ [Working with Hosts](#) on page 59
- ◆ [Working with Queues](#) on page 107
- ◆ [Managing Jobs](#) on page 119
- ◆ [Managing Users and User Groups](#) on page 149
- ◆ [Platform LSF Licensing](#) on page 155
- ◆ [Managing LSF on Platform EGO](#) on page 191
- ◆ [Cluster Version Management and Patching on UNIX and Linux](#) on page 221
- ◆ [Upgrading Platform LSF HPC](#) on page 237

Working with Your Cluster

Contents

- ◆ [Viewing cluster information](#) on page 44
- ◆ [Example directory structures](#) on page 49
- ◆ [Cluster administrators](#) on page 51
- ◆ [Controlling daemons](#) on page 52
- ◆ [Controlling mbatchd](#) on page 54
- ◆ [Reconfiguring your cluster](#) on page 57

Viewing 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>lsid</code>
Cluster name	<code>lsid</code>
Current master host	<code>lsid</code>
Cluster administrators	<code>lsclusters</code>
Configuration parameters	<code>bparams</code>

View LSF version, cluster name, and current master host

- 1 Run `lsid` to display the version of LSF, the name of your cluster, and the current master host:

```
lsid
Platform LSF 7 Update 5 March 3 2009
Copyright 1992-2009 Platform Computing Corporation

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

View cluster administrators

- 1 Run `lsclusters` 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 `lsclusters`.

View configuration parameters

- 1 Run `bparams` 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
Job Dispatch Interval: 20 seconds
Job Checking Interval: 15 seconds
Job Accepting Interval: 20 seconds
```

- 2 Run `bparams -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 `lsb.params`.

```
bparams -l
```

System default queues for automatic queue selection:

```
DEFAULT_QUEUE = normal idle
```

The interval for dispatching jobs by master batch daemon:

```
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 2007
```

```
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
```

Viewing daemon parameter configuration

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

In a MultiCluster environment, `lsadmin showconf` and `badmin showconf` only display the parameters of daemons on the local cluster.

Running `lsadmin showconf` and `badmin showconf` from a master candidate host will reach all server hosts in the cluster. Running `lsadmin showconf` and `badmin showconf` from a slave-only host may not be able to reach other slave-only hosts.

You cannot run `lsadmin showconf` and `badmin showconf` from client hosts. `lsadmin` shows only server host configuration, not client host configuration.

`lsadmin showconf` and `badmin showconf` only displays the values used by LSF.

`lsadmin showconf` and `badmin showconf` display `EGO_MASTER_LIST` from wherever it is defined. You can define either `LSF_MASTER_LIST` in `lsf.conf` or `EGO_MASTER_LIST` in `ego.conf`. LIM reads `lsf.conf` first, and `ego.conf` if EGO is enabled in the LSF cluster. LIM only takes the value of `LSF_MASTER_LIST` if `EGO_MASTER_LIST` is not defined at all in `lsf.conf`.

For example, if EGO is enabled in the LSF cluster, and you define `LSF_MASTER_LIST` in `lsf.conf`, and `EGO_MASTER_LIST` in `ego.conf`, `lsadmin showconf` and `badmin showconf` display the value of `EGO_MASTER_LIST` in `ego.conf`.

If EGO is disabled, `ego.conf` not loaded, so whatever is defined in `lsf.conf` is displayed.

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

- 3 Display LIM configuration.

Use `lsadmin showconf lim` to display the parameters configured in `lsf.conf` or `ego.conf` that apply to root LIM.

By default, `lsadmin` displays the local LIM parameters. You can specify the host to display the LIM parameters.

Examples

- ◆ Show `mbatchd` configuration:

```
badmin showconf mbd
MBD configuration at Fri Jun 8 10:27:52 CST 2007
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 sbatchd configuration on a specific host:**

```
badmin showconf sbd hosta
SBD configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2007
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 sbatchd configuration for all hosts:**

```
badmin showconf sbd all
SBD configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2007
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
```

...

```
SBD configuration for host <hostb> at Fri Jun 8 10:27:52 CST 2007
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 lim configuration:**

```
lsadmin showconf lim
LIM configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2007
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 lim configuration for a specific host:**

```
lsadmin showconf lim hosta
LIM configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2007
LSB_SHAREDIR=/scratch/dev/lsf/user1/0604/work
```

Viewing cluster information

```
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 lim configuration for all hosts:**

```
lsadmin showconf lim all
```

```
LIM configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2007
```

```
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
```

...

```
LIM configuration for host <hostb> at Fri Jun 8 10:27:52 CST 2007
```

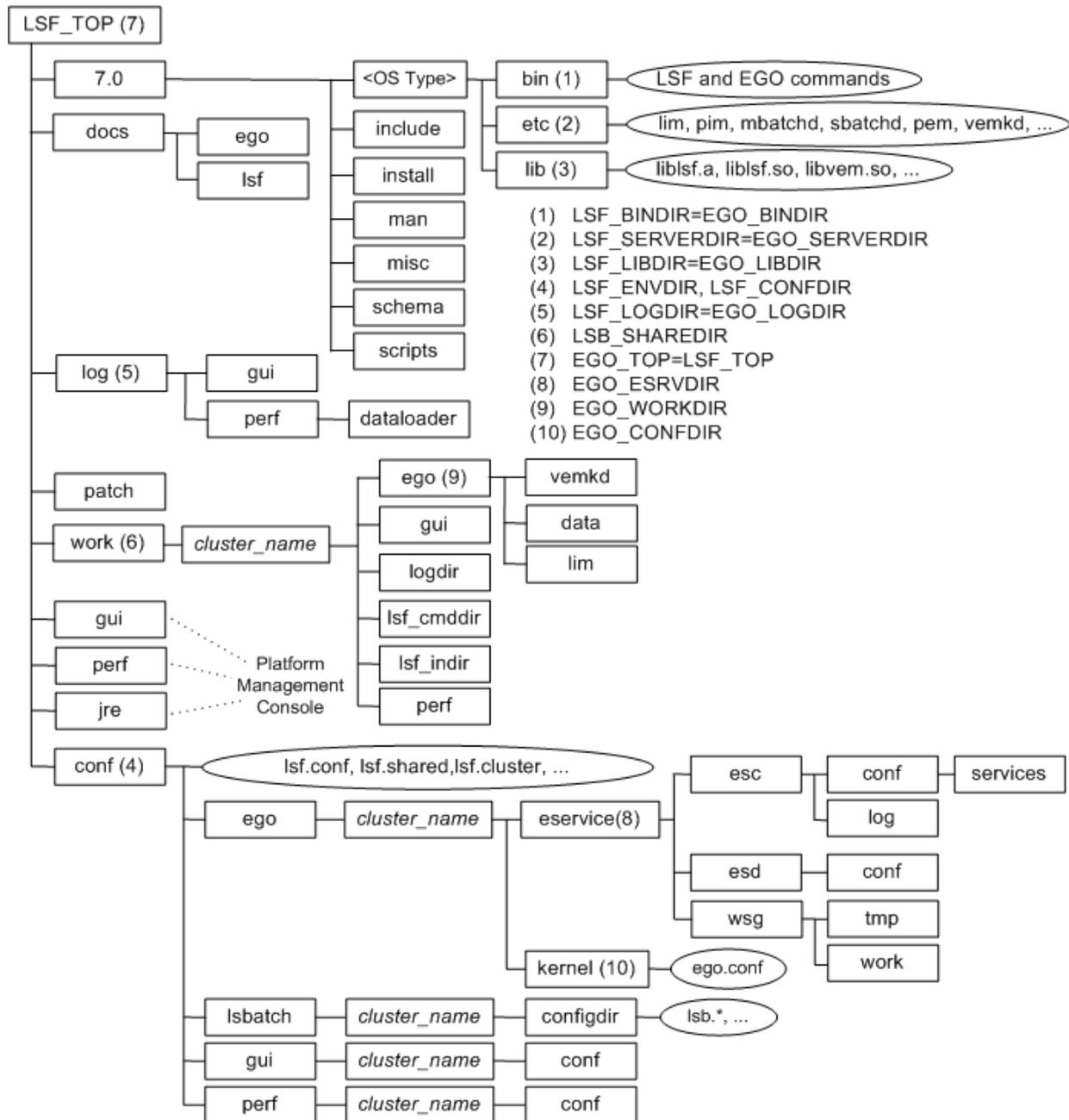
```
LSB_SHAREDIR=/scratch/dev/lsf/user1/0604/work
LSF_CONFDIR=/scratch/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/scratch/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
```

...

Example directory structures

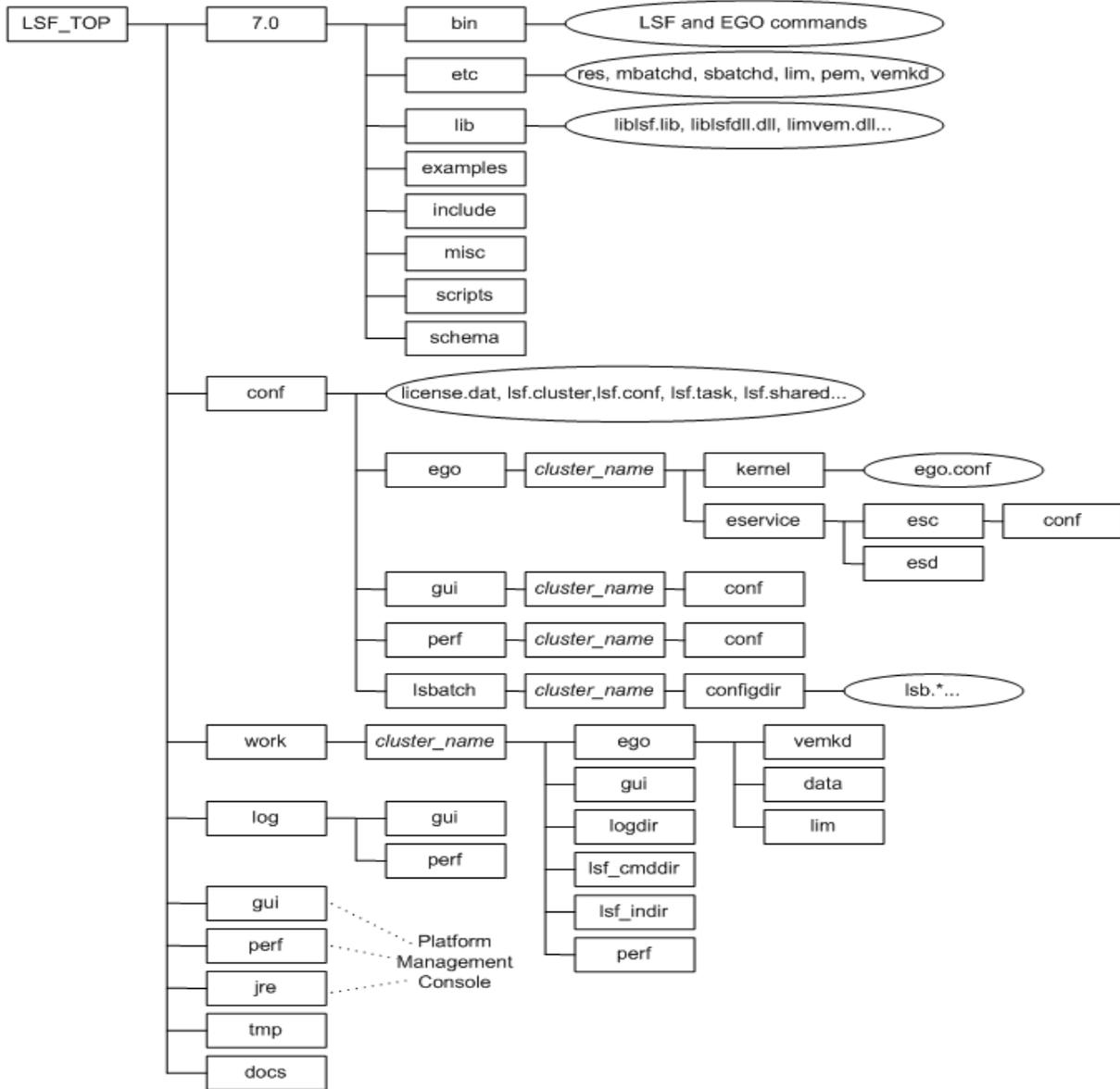
UNIX and Linux

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.



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.

Add cluster administrators

-
- 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 reconfigure LIM.
 - 4 Run `badmin mbdrestart` to restart `mbatchd`.
-

Controlling 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/lsf.sudoers` file. See the *Platform 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.

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>lsf.sudoers</code> for all these commands
	Shut down	<code>lsfshutdown</code>	
sbatchd	Start	<code>badadmin hstartup [host_name ... all]</code>	Must be root or a user listed in <code>lsf.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>	
mbatchd mbschd	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>lsf.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>lsf.sudoers</code> for the startup command
	Shut down	<code>lsadmin limshutdown [host_name ... all]</code>	Must be the LSF administrator for other commands
	Restart	<code>lsadmin limrestart [host_name ... all]</code>	
	Restartall in cluster	<code>lsadmin reconfig</code>	

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.

Controlling mbatchd

You use the `badmin` 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`.

- 1 Run `badmin reconfig`.

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

Restart mbatchd

- 1 Run `badmin 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 `badmin reconfig`.

Log a comment when restarting mbatchd

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

For example:

```
badmin mbdrestart -C "Configuration change"
```

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

- 2 Run `badmin hist` or `badmin mbdhist` to display administrator comments for `mbatchd` restart.

Shut down mbatchd

- 1 Run `badmin hshutdown` to shut down `sbatchd` on the master host.

For example:

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

- 2 Run `badmin mbdrestart`:

```
badmin mbdrestart
Checking configuration files ...
No errors found.
```

This causes `mbatchd` and `mbschd` to exit. `mbatchd` cannot be restarted, because `sbatchd` is shut down. All LSF services are temporarily unavailable, but existing jobs are not affected. When `mbatchd` is later started by `sbatchd`, 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 `mbatchd`. Users see these messages when the batch command retries the connection to `mbatchd`.

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 `mbatchd`.

Reason for no response from <code>mbatchd</code>	Default message	Parameter used to customize the message
<code>mbatchd</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>mbatchd</code> fail	Cannot connect to LSF. Please wait...	<code>LSB_MBD_CONNECT_FAIL_MSG</code>
<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>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 `mbatchd` 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.

Reconfiguring 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>	reconfigures 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>	reconfigures <code>bld</code> , reconfigures 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 reconfigure LIM:

```
lsadmin reconfig
```

The `lsadmin reconfig` command checks for configuration errors.

Reconfiguring your cluster

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 `badadmin reconfig` to reconfigure `mbatchd`:

```
badadmin reconfig
```

The `badadmin 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 `badadmin mbdrestart` to restart `mbatchd`:

```
badadmin mbdrestart
```

The `badadmin 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 `badadmin 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.

Working with Hosts

Contents

- ◆ [Host status](#) on page 60
- ◆ [How LIM Determines Host Models and Types](#) on page 62
- ◆ [Viewing Host Information](#) on page 64
- ◆ [Controlling Hosts](#) on page 70
- ◆ [Adding a Host](#) on page 73
- ◆ [Remove a Host](#) on page 75
- ◆ [Adding Hosts Dynamically](#) on page 77
- ◆ [Automatically Detect Operating System Types and Versions](#) on page 84
- ◆ [Add Host Types and Host Models to lsf.shared](#) on page 86
- ◆ [Registering Service Ports](#) on page 87
- ◆ [Host Naming](#) on page 89
- ◆ [Hosts with Multiple Addresses](#) on page 91
- ◆ [Using IPv6 Addresses](#) on page 93
- ◆ [Specify host names with condensed notation](#) on page 95
- ◆ [Host Groups](#) on page 96
- ◆ [Compute Units](#) on page 100
- ◆ [Tuning CPU Factors](#) on page 103
- ◆ [Handling Host-level Job Exceptions](#) on page 105

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 <code>sbatchd</code> are unreachable.
unreach	LIM is running but <code>sbatchd</code> 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. A closed host does not accept new batch jobs:

Closed reason	Description
closed_Adm	An LSF administrator or root explicitly closed the host using <code>badmin hclose</code> . Running jobs are not affected.
closed_Busy	The value of a load index exceeded a threshold (configured in <code>lsb.hosts</code> , displayed by <code>bhosts -l</code>). Running jobs are not affected. Indices that exceed thresholds are identified with an asterisk (*).
closed_Exc1	An exclusive batch job (i.e., <code>bsub -x</code>) is running on the host.
closed_cu_Exc1	An exclusive compute unit job (i.e., <code>bsub -R "cu[excl]"</code>) is running within the compute unit containing this host.
closed_Full	The configured maximum number of running jobs has been reached. Running jobs will not be affected.
closed_LIM	<code>sbatchd</code> is running but LIM is unavailable.
closed_Lock	An LSF administrator or root explicitly locked the host using <code>lsadmin limlock</code> . Running jobs are suspended (SSUSP). Use <code>lsadmin limunlock</code> to unlock LIM on the local host.
closed_Wind	Host is closed by a dispatch window defined in <code>lsb.hosts</code> . Running jobs are not affected.
closed_EGO	For EGO-enabled SLA scheduling, <code>closed_EGO</code> indicates that the host is closed because it has not been allocated by EGO to run LSF jobs. Hosts allocated from EGO display status <code>ok</code> .

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	STATUS	CPUF	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV	DISPATCH_WINDOW		
hostB	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., lsruntime). 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 (*).
lockW	Does not affect batch jobs, only used for remote task placement (i.e., lsruntime). Host is locked by a run window (configured in <code>lsf.cluster.cluster_name</code> , displayed by <code>lshosts -l</code>).
lockU	Will not accept new batch jobs or remote tasks. An LSF administrator or root explicitly locked the host using <code>lsadmin limlock</code> , or an exclusive batch job (<code>bsub -x</code>) is running on the host. Running jobs are not affected. Use <code>lsadmin limunlock</code> 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_0pteronmProcessor142`, `IA64_2793`, or `SUNWUltrtra510_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 <code>lsf.shared</code> (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: <ul style="list-style-type: none"> a 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 b LIM then attempts to find the best match on the <i>bogomips</i> field ◆ For architecture strings with three fields: <ul style="list-style-type: none"> a 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 b After finding the best match for the model field, LIM attempts to find the best match on the <i>architecture</i> field c 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.

Viewing 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 `lsload` 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>lsload</code>
Host architecture information	<code>lshosts</code>
Host history	<code>badmin hhist</code>
Host model and type information	<code>lshosts</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 `HostGroups` section of `lsb.hosts`. To find out more about condensed host groups and to see the configuration for the above example, see [Defining condensed host groups](#) on page 98.

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     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 `lsload` 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 `lsload` to see load levels for each host:

```
lsload
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.

Viewing host architecture (type and model) information

An LSF cluster may consist of hosts of differing architectures and speeds. 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. For example:

```
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      SGI6   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 `SGI6` 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. See [UNKNOWN host type or model](#) on page 775 for instructions on measures to take.

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 `badadmin hhist` to view the history of a host such as when it is opened or closed:

```
badadmin 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_PentiumIIICoppermine
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	

- 3 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
```

Viewing Host Information

```
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

- 1 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

- 2 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

- 1 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, `hostA` is a static host while `hostB` is a dynamic host.

Controlling Hosts

Hosts are opened and closed by an LSF Administrator or root issuing a command or through configured dispatch windows.

Close a host

```
1 Run badmin hclose:
  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 badmin 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      r1m      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 `lsadmin reconfig` to reconfigure LIM.
 - b Run `badmin reconfig` to reconfigure mbatchd.
 - 4 Run `bhosts -l` to display the dispatch windows.
-

Log a comment when closing or opening a host

- 1 Use the `-C` option of `badadmin hclose` and `badadmin hopen` to log an administrator comment in `lsb.events`:

```
badadmin 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:

```
badadmin hclose -C "backup" hostA
```

followed by

```
badadmin 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 `badadmin hist` or `badadmin hhist` to display administrator comments for closing and opening hosts:

```
badadmin 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	-	-

Controlling Hosts

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`.

Adding a Host

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

Contents

- ◆ [Add a host of an existing type using `lsfinstall` on page 73](#)
- ◆ [Add a host of a new type using `lsfinstall` on page 74](#)

Add a host of an existing type using `lsfinstall`

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

- 1 Verify that the host type already exists 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/7.0` directory. The default is `/usr/share/lsf/7.0`. If the host type currently exists, there is a subdirectory with the name of the host type. If it does not exist, go to [Add a host of a new type using `lsfinstall` on page 74](#).
- 2 Add the host information to `lsf.cluster.cluster_name`:
 - a Log on to the LSF master host as root.
 - b 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  r1m  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
```

- c Save your changes.
- 3 Run `lsadmin reconfig` to reconfigure LIM.

- 4 Run `badmin mbdrestart` to restart `mbatchd`.
 - 5 Run `hostsetup` to set up the new host and configure the daemons to start automatically at boot from `/usr/share/lsf/7.0/install`:

```
./hostsetup --top="/usr/share/lsf" --boot="y"
```
 - 6 Start LSF on the new host:

```
lsadmin limstartup  
lsadmin resstartup  
badmin hstartup
```
 - 7 Run `bhosts` and `lshosts` to verify your changes.
 - ◆ If any host type or host model is UNKNOWN, follow the steps in [UNKNOWN host type or model](#) on page 775 to fix the problem.
 - ◆ If any host type or host model is DEFAULT, follow the steps in [DEFAULT host type or model](#) on page 775 to fix the problem.
-

Add a host of a new type using `lsfinstall`

RESTRICTION: `lsfinstall` is not compatible with clusters installed with `lsfsetup`. To add a host to a cluster originally installed with `lsfsetup`, 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/7.0` directory. The default is `/usr/share/lsf/7.0`. If the host type currently exists, there will be a subdirectory with the name of the host type. If the host type already exists, go to [Add a host of an existing type using `lsfinstall`](#) on page 73.
- 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/7.0/install
```
- 5 Edit `install.config`:
 - a For `LSF_TARDIR`, specify the path to the tar file. For example:

```
LSF_TARDIR="/usr/share/lsf_distrib/7.0"
```
 - 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/7.0/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.
 - ◆ If any host type or host model is UNKNOWN, follow the steps in [UNKNOWN host type or model](#) on page 775 to fix the problem.
 - ◆ If any host type or host model is DEFAULT, follow the steps in [DEFAULT host type or model](#) on page 775 to fix the problem.

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 `badmin hclose` 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/lsb.hosts`
 - ◆ `LSB_CONFDIR/cluster_name/configdir/lsb.queues`
 - ◆ `LSB_CONFDIR/cluster_name/configdir/lsb.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 `lsadmin reconfig` to reconfigure LIM.
- 8 Run `badmin 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.

Remove a Host from Master Candidate List

- 10 If any users of the host use `lscsh` as their login shell, change their login shell to `tcsh` or `csh`. Remove `lscsh` from the `/etc/shells` 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
```
-

Adding 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.

Adding Hosts Dynamically

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

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

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

Allowing 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 `HostGroup` 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 hghostadd`.
-

Changing a dynamic host to a static host

If you want to change a dynamic host to a static host, first use the command `badmin hghostdel` 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`.

Adding dynamic hosts

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.

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. 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:

Adding Hosts Dynamically

- ❖ 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 step 7.
 - ❖ If yes, run the `hostsetup` command. For example:
`cd /usr/share/lsf/7.0/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 (`cshrc.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

**Prerequisites:** 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=7.0
```

**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 step 4.
  - ❖ If yes, run the `hostsetup` command. For example:
 

```
cd /usr/share/lsf/7.0/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
```

## Removing dynamic hosts

To remove a dynamic host from the cluster, you can either set a timeout value, or you can 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 LSF waits for new operating system types and versions

**Prerequisites:** 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 Host Types and Host Models to lsf.shared

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.

### Add a custom host type or model

- 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 `HostType` section:

```
Begin HostType
TYPENAME # Keyword
DEFAULT
IBMAIX564
LINUX86
LINUX64
NTX64
NTIA64
SUNSOL
SOL732
SOL64
SGI658
SOLX86
HPPA11
HPUXIA64
MACOSX
End HostType
```

- b For a new host model, modify the `HostModel` section:

Add the new model and its CPU speed factor relative to other models. For more details on tuning CPU factors, see [Tuning CPU Factors](#) on page 103.

```
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 `badmin reconfig` to reconfigure `mbatchd`.

## Registering 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. To make sure that the port numbers you supply are not already used by applications registered in your service database check `/etc/services` or use the command `yycat services`

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 `lsf.conf`, port numbers defined in the service database are ignored.

### lsf.conf

- 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/instlib/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`.

## Registering Service Ports

- 3 Edit the `/etc/services` file by adding the contents of the `LSF_TOP/version/install/instlib/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 `lsfstartup` 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 `lsfshutdown` 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/instlib/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 `lsfstartup` to restart all daemons in the cluster.

## Host Naming

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.

## Using 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]
```

## Host Naming

For example:

```
atlasD0[0-3,4,5-6, ...]
```

is equivalent to:

```
atlasD0[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 `yycat 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 `yypbind` 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.

### Configuring the 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
```

## Hosts with Multiple Addresses

```
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: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.in-addr.arpa IN PTR host.domain
BB.BB.BB.BB.in-addr.arpa IN PTR host.domain
```

For a dual-stack host:

```
address class type name
AA.AA.AA.AA.in-addr.arpa IN PTR host.domain
BBBB:BBBB:BBBB:BBBB:BBBB:BBBB:BBBB:BBBB.in-addr.arpa IN PTR host.domain
```

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

## Using 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 Does the entry `/proc/net/if_inet6` exist?
      - ◆ If *yes*, the kernel is already configured for IPv6. Go to step 2.
      - ◆ If *no*, go to step b.
    - b To load the IPv6 module into the kernel, execute the following command as root:

```
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 step 1 through step 3 for other hosts in the cluster.
  - 5 To configure IPv6 networking, 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`
- ◆ `bmig`
- ◆ `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 `host1.corp.com`, `host2.corp.com`, `host3.corp.com`, all the way to `host100.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`, `host2.corp.com`, `host3.corp.com`, up to and including `host10.corp.com`. It is also submitted to `host12.corp.com` and the hosts between and including `host20.corp.com` and `host25.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 `HostPartition` 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 `badmin ckconfig` to check the group definition. If any errors are reported, fix the problem and check the configuration again.
- 8 Run `badmin mbdrestart` to apply the new configuration.

## Using 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:

- ◆ Use a tilde (~) to exclude 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)
```

- ◆ 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 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])
```

- ◆ Use square brackets with commas (`[integer1 , integer2 ...]`) to define individual non-negative integers at the end of a host 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 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
```

## Host Groups

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

### Defining 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 `HostGroup` 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`
- ◆ `bjobs`
- ◆ `bjobs -w`

For the `bhosts` output of this configuration, see [Viewing Host Information](#) on page 64.

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, `bjobs` 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
```

`bjobs` 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
```

## Importing 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 the *Platform LSF Configuration Reference*.

## 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.

For a complete description of compute units see [Controlling Job Locality using Compute Units](#) on page 547 in Chapter 34, “Running Parallel Jobs”.

### 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
enc11 (hostA hg1) enclosure
enc12 (hostC hostD) enclosure
enc13 (hostE hostF) enclosure
enc14 (hostG hg2) enclosure
rack1 (enc11 enc12) rack
rack2 (enc13 enc14) rack
cab1 (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 `badadmin ckconfig` to check the compute unit definition. If any errors are reported, fix the problem and check the configuration again.****12 Run `badadmin mbdrestart` to apply the new configuration.**

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

## Using 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*)
```

## Compute Units

- ◆ 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`, `allremote`, `all@cluster`, `other` and `default` cannot be used when defining compute units.

## Defining 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
enc1A Y (hostA hostB hostD) enclosure
enc1B N (hostC hostE) enclosure
End HostGroup
```

The following commands display condensed host information:

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

For the `bhosts` output of this configuration, see [Viewing Host Information](#) on page 64.

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

## Importing 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 the *Platform LSF Configuration Reference*.

## Using 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.

## Tuning 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: `hostA` and `hostB`. In this cluster, `hostA` takes 30 seconds to run a benchmark and `hostB` takes 15 seconds to run the same test. The CPU factor for `hostA` should be 1, and the CPU factor of `hostB` should be 2 because it is twice as fast as `hostA`.

## View normalized ratings

- 1 Run `lsload -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 *Platform 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 `badmin reconfig` to reconfigure `mbatchd`.

## Handling 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.

### Configuring 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
```

### Configuring thresholds for host exception handling

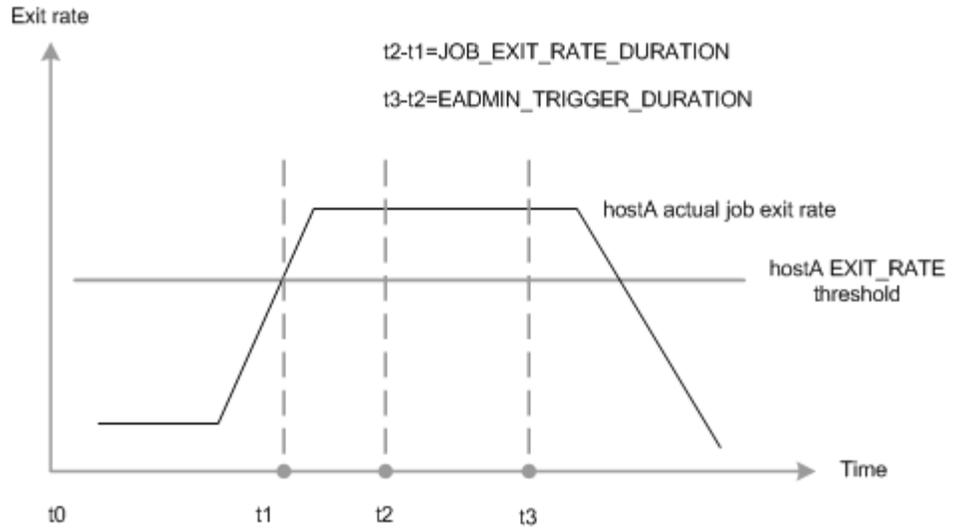
By default, LSF checks the number of exited jobs every 5 minutes. Use JOB\_EXIT\_RATE\_DURATION in `lsb.params` to change this default.

#### 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 `t1` to time `t2` (`t2=t1 + JOB_EXIT_RATE_DURATION` in `lsb.params`). At `t2`, the exit rate is still high, and a host exception is detected. At `t3` (`EADMIN_TRIGGER_DURATION` in `lsb.params`), LSF invokes `eadmin` and the host exception is handled. By default, LSF closes `hostA` and sends email to the LSF administrator. Since `hostA` is closed and cannot accept any new jobs, the exit rate drops quickly.



## Working with Queues

### Contents

- ◆ [Queue States](#) on page 107
- ◆ [Viewing Queue Information](#) on page 108
- ◆ [Control Queues](#) on page 110
- ◆ [Add and Remove Queues](#) on page 113
- ◆ [Manage Queues](#) on page 115
- ◆ [Handling Job Exceptions in Queues](#) on page 116

### 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 (`badmin qinact`), and `Inact_Win` when inactivated by a run or dispatch window.

## Viewing 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>badmin qhist</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 PRIO 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 - - - - -

 cpuspeed bandwidth
loadSched - -
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 `badadmin qhist` to display the times when queues are opened, closed, activated, and inactivated.

```
badadmin 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 or through configured dispatch and run windows.

## Close a queue

---

```
1 Run badmin qclose:
 badmin 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 badmin qopen:
 badmin qopen normal
 Queue <normal> is opened
```

---

## Inactivate a queue

---

```
1 Run badmin qinact:
 badmin qinact normal
 Queue <normal> is inactivated
```

---

## Activate a queue

---

```
1 Run badmin qact:
 badmin qact normal
 Queue <normal> is activated
```

---

## Log a comment when controlling a queue

---

```
1 Use the -C option of badmin queue commands qclose, qopen, qact, and
 qinact to log an administrator comment in lsb.events.
```

```
badmin 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:

```
badmin qclose -C "add user" normal
```

followed by

```
badmin 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"
```

## Control Queues

- 2 Use `badadmin hist` or `badadmin qhist` to display administrator comments for closing and opening hosts.

`badadmin 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
```

```
PRIO NICE STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SSUSP USUSP RSV
30 20 Closed:Active - - - - 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(min) UNDERRUN(min) 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 `badadmin 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 `badadmin reconfig`.
- 4 Run `bqueues -l` to display the run windows.

## Add and Remove Queues

### 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.

## Add and Remove Queues

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.
- 

## 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.  

```
badmin qclose night
Queue <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 SUBM
IT_TIME
5308 user5 RUN night hostA hostD job5 Nov 2
1 18:16
5310 user5 PEND night hostA hostC job10 Nov 2
1 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 `badmin reconfig` to reconfigure `mbatchd`.
-

## Manage Queues

### 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 `engineering1` and `engineering2` 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 `badmin ckconfig` to check the new queue definition. If any errors are reported, fix the problem and check the configuration again.
- 5 Run `badmin 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 `badmin 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
```

## Handling 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.

### Configuring job exception handling (`lsb.queues`)

You can configure your queues to detect job exceptions. Use the following parameters:

|                     |                                                                                                                                                                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>JOB_IDLE</b>     | 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 <code>eadmin</code> to trigger the action for a job idle exception. |
| <b>JOB_OVERRUN</b>  | Specify a threshold for job overrun. If a job runs longer than the specified run time, LSF invokes <code>eadmin</code> to trigger the action for a job overrun exception.                                                                                       |
| <b>JOB_UNDERRUN</b> | Specify a threshold for job underrun. If a job exits before the specified number of minutes, LSF invokes <code>eadmin</code> 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

## Configuring 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.



## Managing Jobs

### Contents

- ◆ [Understanding Job States](#) on page 120
- ◆ [View Job Information](#) on page 123
- ◆ [Changing Job Order Within Queues](#) on page 127
- ◆ [Switch Jobs from One Queue to Another](#) on page 128
- ◆ [Forcing Job Execution](#) on page 129
- ◆ [Suspending and Resuming Jobs](#) on page 130
- ◆ [Killing Jobs](#) on page 131
- ◆ [Sending a Signal to a Job](#) on page 133
- ◆ [Using Job Groups](#) on page 134
- ◆ [Handling Job Exceptions](#) on page 145

## Understanding Job States

The `bjobs` 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       |

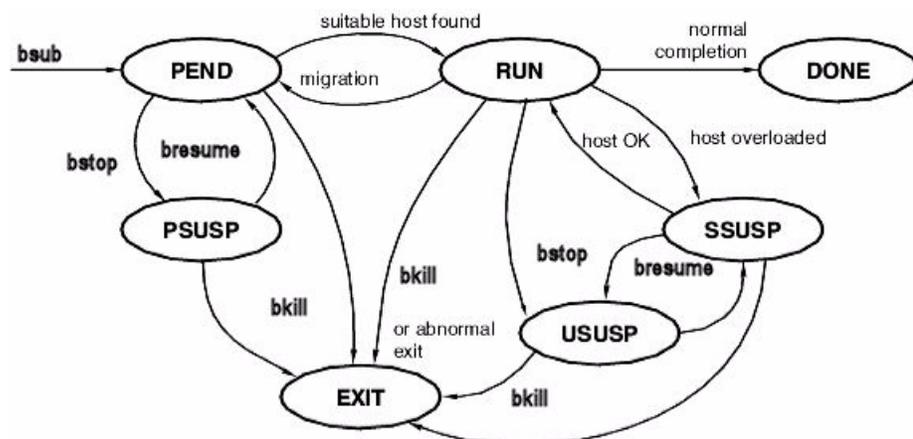
### Suspended job states

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.

See Chapter 32, “[Chunk Job Dispatch](#)” for more information about chunk jobs.

## 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. See [Handling Host-level Job Exceptions](#) on page 105 for more information.

## 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.

See Chapter 38, “[Pre-Execution and Post-Execution Commands](#)” for more information.

## View Job Information

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

### View all jobs for all users

- 1 Run `bjobs -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 `bjobs -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 Job Information

### View running jobs

- 1 Run `bjobs -r` to display running jobs.

### View done jobs

- 1 Run `bjobs -d` to display recently completed jobs.

### View pending job information

- 1 Run `bjobs -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 `bjobs -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: Submitted from host <hostA>, CWD <${HOME}>, Output File
</dev/null>, Specified Hosts <hostB>;
Wed Aug 13 14:23:43: 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: Submitted from host <hostA>, CWD <${HOME}>,
Output File </dev/null>, Specified Hosts <
hostB>;
Wed Aug 13 14:18:10: Started on <hostB>, Execution Home </home/user1>, Execution
CWD </home/user1>;
Wed Aug 13 14:18:50: 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.**

View Job Information

## View resizable job information

- 1 Run `bjobs -d` to display recently completed jobs.

## Changing 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 `btop` 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.

## Moving 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 `user1`'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 | night  | hostA     |           | /s200    | Oct 23 11:04 |
| 5310  | user1 | PEND | night  | hostB     |           | /myjob   | Oct 23 13:45 |
| 5311  | user2 | PEND | night  | 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 | night  | hostA     |           | /s200    | Oct 23 18:17 |
| 5310  | user1 | PEND | night  | hostB     |           | /myjob   | Oct 23 13:45 |
| 5309  | user2 | PEND | night  | hostA     |           | /s700    | Oct 23 11:04 |

## Switch Jobs from One Queue to Another

You can use the command `bswitch` 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 `bswitch` to move pending and running jobs from queue to 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 /job500 Oct 23 10:16
5309 user2 RUN priority hostA hostB /job200 Oct 23 11:04
5311 user2 PEND night hostA /job700 Oct 23 18:17
5310 user1 PEND night hostB /myjob 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 `night` queue and switches them to the `idle` queue.

```
bswitch -q night idle 0
Job <5308> is switched to queue <idle>
Job <5310> is switched to queue <idle>
```

## Forcing 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 `hostA`:

```
brun -m hostA 104
```

---

## Suspending and Resuming Jobs

A job can be suspended by its owner or the LSF administrator. These jobs are considered user-suspended and are displayed by `bjobs` 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:

- ◆ `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`.

### 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.

---

## Killing 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 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.

---

## Sending 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 `SIGSTOP` to suspend a job, `SIGCONT` to resume a job, and `SIGKILL` 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 `SIGKILL` signal is sent to the specified jobs to kill them. Twenty seconds before `SIGKILL` is sent, `SIGTERM` and `SIGINT` are sent to give the job a chance to catch the signals and clean up.

On Windows, job control messages replace the `SIGINT` and `SIGTERM` 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 `SIGTSTP`. If the job is running on HP-UX and `SIGTSTP` 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.

---

## Using 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 -sla service_class_name -g job_group_name`). Job groups attached to an SLA service class with `bgadd` are explicit job groups (`bgadd -sla 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 `bjgroup` 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 user1
- ◆ Job group /A/B is created by user2
- ◆ Job group /A/B/C is created by user3

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:

- ◆ user1 can control jobs submitted by other users in all 3 job groups: /A, /A/B, and /A/B/C
- ◆ user2 can control jobs submitted by other users only in 2 job groups: /A/B and /A/B/C
- ◆ user3 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
```

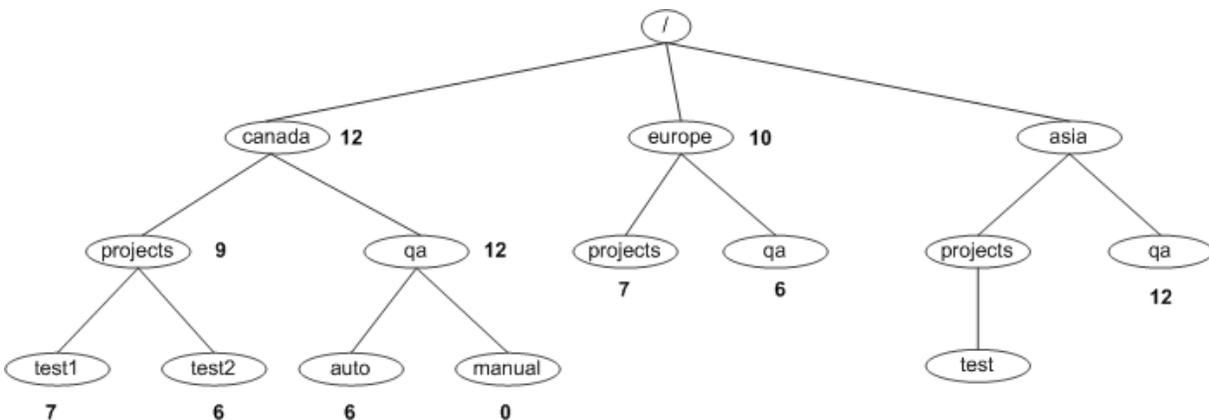
If `/canada` is existing job group, and `/canada/projects` and `/canada/projects/test` are new groups, only the job group `/canada/projects/test` is limited to 6 running and suspended jobs. Job group `/canada/projects` will have whatever limit is specified for its parent job group `/canada`. The limit 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 `myjob` 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 jobs in job groups

View job group information, and jobs running in specific job groups.

### View information about job groups (`bjgroup`)

- 1 Use the `bjgroup` 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 `bjgroup -s` to sort job groups by group hierarchy.

For example, for job groups named `/A`, `/A/B`, `/X` and `/X/Y`, `bjgroup -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 |



```
</home/user1;
Tue Jun 17 16:22:01: 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.

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: Submitted from host <hostA>, to Queue <normal>, CWD
<${HOME}/lsf51/5.1/sparc-sol7-64/bin>;
```

```
Wed May 14 15:24:10: Parameters of Job are changed:
Job group changes to: /risk_group/portfolio2/monthly;
```

```
Wed May 14 15:24:17: Dispatched to <hostA>;
```

```
Wed May 14 15:24:17: 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 subgroups. 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
```

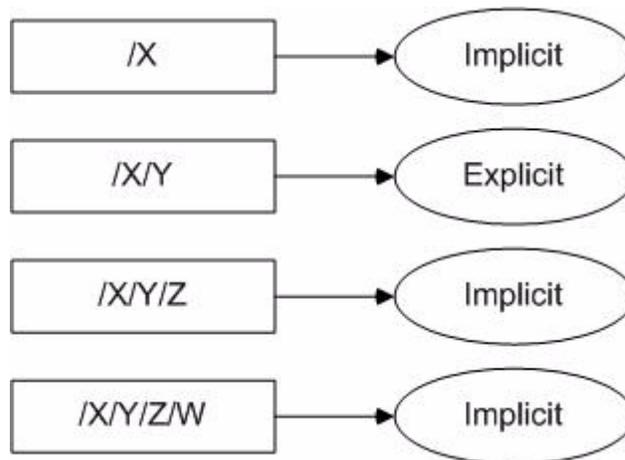
## Automatic job group cleanup

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.

## Handling 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 `bjobs -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 `lsb.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 `lsb.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 `lsb.hosts`).

If `EXIT_RATE` is not 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.

### Customize job exception actions with the `eadmin` script

When an exception is detected, LSF takes appropriate action by running the script `LSF_SERVERDIR/eadmin` on the master host.

You can customize `eadmin` to suit the requirements of your site. For example, `eadmin` could find out the owner of the problem jobs and use `bstop -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 `eadmin`.

### 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.

## Handling 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:

| Exit rate type ...                                                                             | Includes ...                                                                                                                                                                                                                                                                                                                           |
|------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>JOBEXIT</code>                                                                           | Local exited jobs<br>Remote job initialization failures<br>Parallel job initialization failures on hosts other than the first execution host<br>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.) |
| <code>JOBEXIT_NONLSF</code><br>This is the default when <code>EXIT_RATE_TYPE</code> is not set | Local exited jobs<br>Remote job initialization failures<br>Parallel job initialization failures on hosts other than the first execution host                                                                                                                                                                                           |

| Exit rate type ... | Includes ...                                                                                          |
|--------------------|-------------------------------------------------------------------------------------------------------|
| JOBINIT            | Local job initialization failures<br>Parallel job initialization failures on the first execution host |
| HPCINIT            | Job initialization failures for Platform LSF 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.

### Excluding 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`.

### Scaling and tuning job exit rate by number of slots

On large, multiprocessor hosts, use to `ENABLE_EXIT_RATE_PER_SLOT=Y` in `lsb.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 `lsb.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.

### For more information

- ◆ See [Handling Host-level Job Exceptions](#) on page 105 for information about configuring host-level job exceptions.
- ◆ See [Handling Job Exceptions in Queues](#) on page 116 for information about configuring job exceptions. in queues

# Managing Users and User Groups

## Contents

- ◆ [Viewing User and User Group Information](#) on page 149
- ◆ [About User Groups](#) on page 151
- ◆ [Existing User Groups as LSF User Groups](#) on page 151
- ◆ [LSF User Groups](#) on page 152

## Viewing 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."

---

## Viewing User and User Group Information

### View user information

- 1 Run `busers all`.

```
busers all
USER/GROUP JL/P MAX NJOBS PEND RUN SSUSP USUSP RSV
default 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 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
develop 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]
```

---

## 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 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 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 (`badmin`). 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 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 `lsb.queues` for authorized queue users
- ◆ `USER_NAME` in `lsb.users` for user job slot limits
- ◆ `USER_SHARES` (optional) in `lsb.hosts` for host partitions or in `lsb.queues` or `lsb.users` for queue fairshare policies

## LSF User Groups

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

Use `bugroup` to view user groups and members, use `busers` to view all users in the cluster.

## Where to use LSF user groups

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

- ◆ `USERS` in `lsb.queues` for authorized queue users
- ◆ `USER_NAME` in `lsb.users` for user job slot limits
- ◆ `USER_SHARES` (optional) in `lsb.hosts` for host partitions or in `lsb.queues` for queue fairshare policies

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 `lsb.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`.

- 6 Optional: To enable hierarchical fairshare, specify share assignments in the `USER_SHARES` column.

- 7 Save your changes.
  - 8 Run `badadmin ckconfig` to check the new user group definition. If any errors are reported, fix the problem and check the configuration again.
  - 9 Run `badadmin reconfig` to reconfigure the cluster.
- 

## Importing 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 the *Platform LSF Configuration Reference*.



## Platform LSF Licensing

### Contents

- ◆ [The LSF License File](#) on page 156
- ◆ [How LSF Permanent Licensing Works](#) on page 160
- ◆ [Installing a Demo License](#) on page 162
- ◆ [Installing a Permanent License](#) on page 164
- ◆ [Updating a License](#) on page 170
- ◆ [FLEXlm Basics](#) on page 172
- ◆ [Multiple FLEXlm License Server Hosts](#) on page 175
- ◆ [Partial Licensing](#) on page 177
- ◆ [Floating Client Licenses](#) on page 181
- ◆ [Troubleshooting License Issues](#) on page 187

## The LSF License File

You must have a valid license to run LSF. This section helps you to understand the types of LSF licenses and the contents of the LSF license file. It does not contain information required to install your license.

---

**TIP:** To learn about licensing a cluster that includes Windows hosts, see *Using Platform LSF on Windows*.

---

### 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, as described in [Installing a Permanent License](#) on page 164.

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 Macrovision® FLEXlm™ 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 FLEXlm™ 7.2 or later
- ◆ Is keyed to one or more specific FLEXlm license server hosts
- ◆ Does not expire

### Enforcement of multicore processor licenses on Linux and Windows

Multicore hosts running Linux or Windows must be licensed by the `lsf_dualcore_x86` license feature. Each physical processor requires one standard LSF license and  $num\_cores - 1$  `lsf_dualcore_x86` licenses. For example, a processor with 4 cores requires 3 `lsf_dualcore_x86` licenses.

Use `lshosts -l` to see the number of multicore licenses enabled and needed. For example:

```
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 4 2

LICENSES_ENABLED: (LSF_Base LSF_Manager LSF_MultiCluster LSF_DualCore_x86)
LICENSE_NEEDED: Class(B), Multi-cores
...
```

## Enforcement of grid license management plugin licenses

The new license optimization features enabled by the grid license management plugin require the `lsf_mv_grid_filter` license feature.

The number of `lsf_mv_grid_filter` licenses should be at least the number of LSF License Scheduler licenses.

## Banded licensing

You can use permanent licenses with restrictions in operating system and hardware configurations. These banded licenses have three classes, with the E-class licenses having no restrictions. Banded licenses support the following operating systems and hardware configurations:

| License type | Supported operating systems               | Processor                      | Physical memory                                     | Physical processors/sockets      |
|--------------|-------------------------------------------|--------------------------------|-----------------------------------------------------|----------------------------------|
| B-Class      | Linux, Windows, MacOS                     | Intel X86/AMD64/EM64T          | Up to and including 4 GB physical memory on a node  | Up to and including 2 processors |
| S-Class      | Linux, Windows, MacOS                     | Intel X86/AMD64/EM64T          | Up to and including 16 GB physical memory on a node | Up to and including 4 processors |
| E-Class      | Linux, Windows, MacOS                     | Intel X86/AMD64/EM64T          | More than 16 GB physical memory on a node           | More than 4 processors           |
|              | All other LSF-supported operating systems | Intel X86/AMD64/EM64T          | N/A                                                 | N/A                              |
|              | N/A                                       | All other supported processors | N/A                                                 | N/A                              |

In the LSF license file:

```
FEATURE lsf_manager lsf_ld 6.200 8-may-2009 2 ADE2C12C1A81E5E8F29C \
VENDOR_STRING=Platform NOTICE=Class(S)
FEATURE lsf_manager lsf_ld 6.200 8-may-2009 10 1DC2C1CCEF193E42B6DC \
VENDOR_STRING=Platform NOTICE=Class(E)
```

## Determining what licenses a host needs

Use `lim -t` and `lshosts -l` to see the license requirements for a host. For example:

```
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-core
Matched Type : NTX64
Matched Architecture : EM64T_3000
Matched Model : Intel_EM64T
CPU Factor : 60.0
```

## The LSF License File

```
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 2 4 2
...
LICENSE_NEEDED: Class(B), Multi-cores
...
```

## Format of the demo license file

This is intended to familiarize you with the demo license file. You do not need to read this section if you are only interested in installing the license.

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`.

For example:

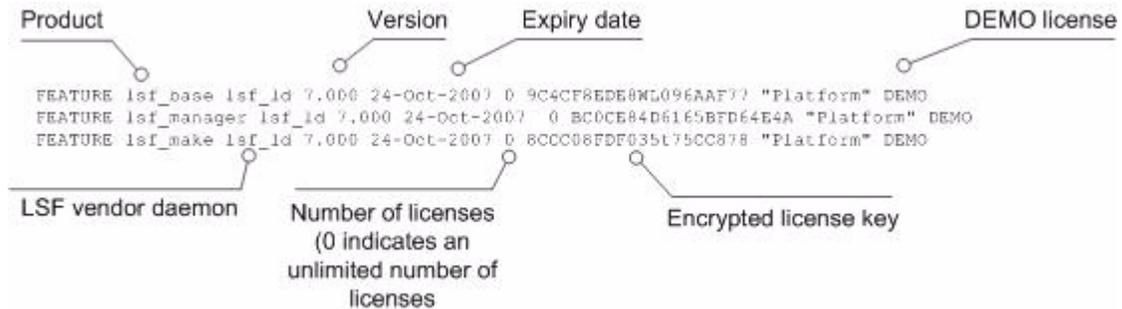
```
FEATURE lsf_base lsf_ld 7.000 24-Oct-2009 100 DCF7C3D92A5471A12345 "Platform" DEMO
```

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.

## Example demo license file

The following is an example of a demo license file. This file licenses LSF 7, advance reservation, and Platform LSF Make. The license is valid until October 24, 2009.



## Format of the permanent license file

A permanent license file has the same format as other products licensed with *FLEXlm*. If you are already familiar with *FLEXlm* license files, you can skip this section.

In addition to the information presented in the demo license file (see [Format of the demo license file](#) on page 158), the permanent license file includes the following:

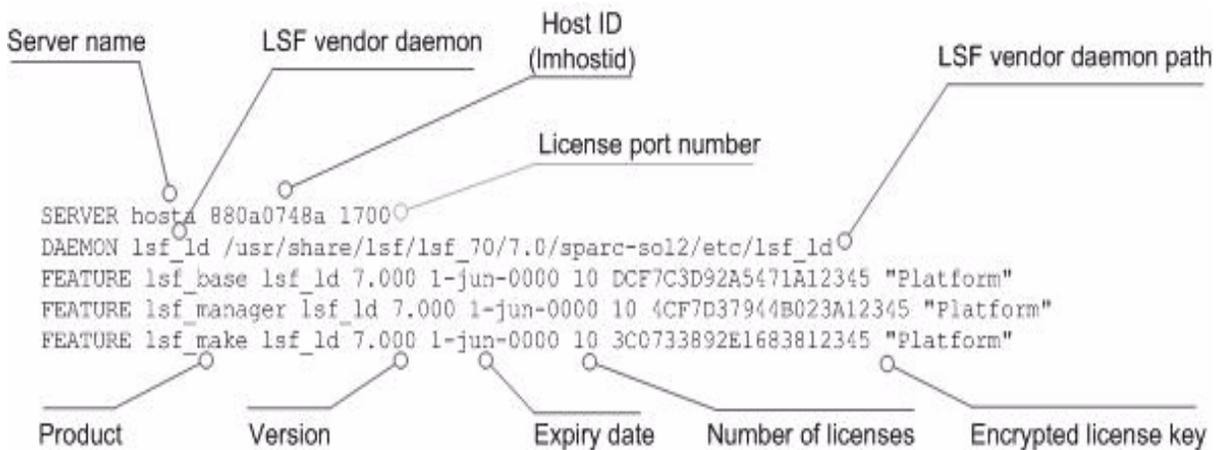
- ◆ A **SERVER** line for each *FLEXlm* server host. The **SERVER** line contains the following server information:
  - ❖ Host name
  - ❖ Hardware host ID
  - ❖ TCP port number used by the *FLEXlm* license server daemon (`lmgrd`)
- ◆ A **DAEMON** line for each software vendor, which gives the file path name of the LSF license vendor daemon (`lsf_ld`, normally installed in `LSF_SERVERDIR`).
- ◆ Each **FEATURE** line in the license ends with `Platform`, instead of `DEMO`.

```
FEATURE lsf_base lsf_ld 7.000 1-jan-0000 100 DCF7C3D92A5471A12345 "Platform"
```

For permanent licenses, the licenses granted by the **FEATURE** line can be accessed only through license server hosts listed on the **SERVER** lines.

## Example permanent license file

The following is an example of a permanent license file.



The license server daemon is configured to run on `hosta`, using TCP port 1700. It allows 10 single-processor hosts to run Platform LSF 7 and Platform LSF Make, with no expiry date.

## How LSF Permanent Licensing Works

This section is intended to give you a better understanding of how LSF licensing works in a production environment with a permanent license. It does not contain information required to install your license.

Platform LSF uses the FLEX $lm$  license management product from GLOBEtrrotter Software to control its licenses. LSF licenses are controlled centrally through the LSF master LIM.

### FLEX $lm$ license server

Permanent LSF licenses are managed by the FLEX $lm$  license server daemon (`lmgrd`). The FLEX $lm$  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_ld`, which periodically keeps track of how many LSF licenses are checked out and who has them. Only one `lsf_ld` can run on a host. If `lsf_ld` 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 FLEX $lm$  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. For information about how to configure the `TIMEOUT ALL` parameter, see the FLEX $lm$  documentation.

## LSF license checkout

Only the master LIM can check out licenses. No other part of LSF has any contact with the *FLEXlm* 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. Most LSF software is licensed per CPU, not per host or per cluster, so multi-processor hosts 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 Calculate the total number of licenses needed for the master LIM.
- 2 Before slave LIMs contact the master, calculate the total number of licenses needed for all LSF server hosts and check them out. When the slave LIMs start, they contact the master host to get the licenses they need.
- 3 Check 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.

## 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.

## Unlicensed cluster

While LSF cannot contact a license server daemon, LSF commands are automatically resubmitted, not aborted.

## Installing a Demo License

This section includes instructions for licensing LSF with a new demo license.

Most new users should follow the procedure under [Install and license LSF for the first time](#) on page 162.

If you already have LSF installed, see [Install a demo license manually](#) on page 162.

### Install and license LSF for the first time

If LSF has never been installed before, you should install and license LSF in one step, using a demo license and the LSF installation program for UNIX, `lsfinstall`.

- 
- 1 Acquire your demo license before you install LSF.  
See [Get a demo license](#) on page 163.
  - 2 When you receive your license file, save it as `license.dat`.  
See [Viewing and editing the license file](#) on page 165.
  - 3 Install LSF using `lsfinstall` as described in *Installing Platform LSF on UNIX and Linux*. `lsfinstall` automatically sets up the LSF demo license.
- 

### Install a demo license manually

If you just need to update or replace an existing LSF license, see [Updating a License](#) on page 170.

If LSF is installed without a license file, or the license file is not properly installed, you can install a demo license manually.

- 
- 1 Acquire your demo license.  
See [Get a demo license](#) on page 163.
  - 2 When you receive your license file, save it as `license.dat`.  
See [Viewing and editing the license file](#) on page 165.
  - 3 Move the license file to a location where it can be shared.  
See [Location of the LSF license file for a demo license](#) on page 163.
  - 4 Set the `LSF_LICENSE_FILE` parameter to point to your license file.  
See [LSF\\_LICENSE\\_FILE parameter](#) on page 166.
  - 5 Start or restart LSF. This causes the license file to be read and the changes accepted by LSF:
    - ❖ If LSF daemons are already running, reconfigure LSF using the following LSF commands:

```
lsadmin reconfig
badmin mbdrestart
```
    - ❖ If this is a new installation, start LSF using one of the following two methods:
      - a On the LSF master host, run the following LSF commands:

```
lsadmin limstartup all
```

```
lsadmin resstartup all
badmin hstartup all
```

- b** On any LSF host, run the script:  
LSF\_BINDIR/lsfstartup
- 

## Get a demo license

To get a demo license from Platform Computing or your Platform LSF vendor.

## Location of the 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 `lsfinstall` puts the LSF license file in a shared directory where it is available to all LSF hosts.

If you install the license manually, use either of the following methods to ensure that a license is available to all hosts:

- ◆ Share the same license file between all the hosts using NFS
- ◆ Install a separate copy of the license file on each host

## Installing a Permanent License

This section includes instructions for licensing LSF with a new permanent license. If you have not yet installed LSF, you can use a demo license to get started. See [Installing a Demo License](#) on page 162.

If you already have LSF, see [Install a permanent license for the first time](#) on page 164.

### 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.

If you just need to update an existing permanent license, see [Updating a License](#) on page 170.

- 1 Acquire your permanent license.  
See [Getting a permanent license](#) on page 165.
- 2 When you receive your license file, save it as `license.dat`.  
See [Viewing and editing the license file](#) on page 165.
- 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/lsf_62/7.0/sparc-sol2/etc/lsf_ld
```

The `lsf_ld` binary should be available to the `FLEXlm` 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 7.000 1-jun-0000 10 DCF7C3D92A5471A12345 "Platform"
FEATURE lsf_multicluster lsf_ld 7.000 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.

See [Licensing LSF products and features](#) on page 168.

- 5 Make sure the file is in a location where it can be accessed by the license server daemons.  
See [Location of the LSF license file for a permanent license](#) on page 166.
- 6 Set the `LSF_LICENSE_FILE` parameter to point to your license file.  
See [LSF\\_LICENSE\\_FILE parameter](#) on page 166.
- 7 Start the license server daemon.

See [Start the license daemons](#) on page 172.

- 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 (see [FLEXlm license server host](#) on page 168)
- ◆ Host identifier of the license server host (see [Getting the FLEXlm license server host identifier](#) on page 165)
- ◆ Products required (see [Licensing LSF products and features](#) on page 168)
- ◆ Number of licenses required for your cluster (see [LSF license checkout](#) on page 161)

## Getting the FLEXlm license server host identifier

When an LSF license is managed by FLEXlm, you must provide a hardware host name and host identifier for the FLEXlm license server host at your site.

If you do not already use FLEXlm to manage other applications, you must choose a host as the FLEXlm license server host before you request your license. See [Selecting a license server host](#) on page 168.

Use the `lmhostid` command (normally located in `LSF_SERVERDIR`) to get the hardware identifier of your FLEXlm license server host. For example, run this command on the FLEXlm server host:

```
lmhostid
lmhostid - Copyright (C) 1989-1997 Globetrotter Software, Inc.
The FLEXlm host ID of this machine is "68044d20"
```

In this example, send the code “68044d20” to Platform.

## Viewing and editing the license file

Your LSF license should be a text file (normally named `license.dat`). Use any text editor such as `vi` or `emacs` to open a copy of your license file for viewing or editing.

## Installing a Permanent License

For example,

- ◆ If you receive your license from Platform as text, you must create a new file and copy the text into the file.
- ◆ You might have to modify lines in the license, such as the path in the DAEMON line when you install a new permanent license.
- ◆ You might want to 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 FLEXlm.

If you can, make carriage returns visible, or view the text without word wrap. Although the lines might wrap when displayed on a screen, make sure each line in the text file has 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.

Do not try to modify lines or fields unless instructed to do so by Platform. You could corrupt the license. Do not combine demo license lines with permanent license lines. For more information about LSF license files, see [Format of the demo license file](#) on page 158 and [Format of the permanent license file](#) on page 159.

## Location of the LSF license file for a permanent license

For a permanent license, the FLEXlm license daemon `lmgrd` and the LSF vendor daemon `lsf_ld` 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 FLEXlm 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 colon (`:`) on UNIX and Linux, and a semicolon (`;`) on Windows to separate the `port@host_names`. The port number must be the same as that specified in the `SERVER` line of the license file. For example:

UNIX:

```
LSF_LICENSE_FILE="port@hostA:port@hostB:port@hostC"
```

Windows:

```
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, you specify the license installation directory. The default is the LSF configuration directory (`LSF_SERVERDIR` for the custom installation).
- ◆ If you installed FLEXlm separately from LSF to manage other software licenses, the default FLEXlm installation puts the license file a location you specify, usually:
  - ❖ UNIX: `/usr/share/flexlm/licenses/license.dat`
  - ❖ Windows: `C:\flexlm\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 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, or the FEATURE line in the license file is ignored. 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 strings are valid in the license file, but should not be used in the PRODUCTS line:

- ◆ LSF\_Client
- ◆ LSF\_Float\_Client

LSF client hosts are licensed per host, not per CPU, so there is no difference between licensing a single-processor host and a multi-processor host.

See [Floating Client Licenses](#) on page 181 for information about configuring LSF floating clients.

## FLEXlm 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 FLEXlm 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 FLEXlm license file.

If you are not already using FLEXlm, or prefer to administer LSF license management separately, you must choose a host to run the license daemons. See [Selecting a license server host](#) on page 168.

It is possible to run multiple license server hosts for failover purposes. See [Multiple FLEXlm License Server Hosts](#) on page 175.

## Selecting a license server host

By reading this, you will gain the information needed to make an educated decision when selecting a license server host.

The FLEXlm 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. LSF cannot run if it cannot contact the license server daemon. 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.

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.

## FLEXlm software for the license server host

Permanent (server-based) LSF licenses work with FLEXlm version 7.2 or later.

If your FLEXlm license server host is of the same host type as one or more LSF hosts, the FLEXlm 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 FLEXlm 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).

If your FLEXlm license server host is a different host type, you do not need the complete LSF distribution. You can download just the FLEXlm software from Platform's FTP site, and copy it to any convenient location.

## Updating a License

This section is intended for those who are updating an existing LSF license file.

To switch your demo license to a permanent license, see [Installing a Permanent License](#) on page 164.

To update a license:

- 1 Contact Platform to get the license. See [Requesting a new license](#) on page 170.
- 2 Update the license using one of the following procedures
  - ❖ [Updating a license with FEATURE lines](#) on page 170
  - ❖ [Update a license with INCREMENT lines](#) on page 171

---

**REMEMBER:** After updating an existing LSF license file or adding FLEXlm licenses, you must use `lsadmin limrestart` on the master LIM or `lsadmin reconfig` from any LIM to use the new licenses. You must also use `lmreread`, or restart the `lmgrd` daemon.

---

### Requesting a new license

To update your license, contact Platform Computing or your Platform LSF vendor. Since you already have a license, you will only receive new lines to put into your existing file.

#### FEATURE LINES

To update your license file, LSF licenses are sent to you in the form of FEATURE license lines when you:

- ◆ Already have some LSF products, but purchase a new product for the first time
- ◆ Upgrade LSF to a newer version
- ◆ Already have LSF, but time-limited licenses have expired

#### INCREMENT LINES

If you add hosts to your cluster and you already have an LSF product, licenses for the additional hosts are normally sent to you in the form of INCREMENT license lines.

### Updating a license with FEATURE lines

FLEXlm only accepts one license key for each feature listed in a license file. If there is more than one FEATURE line for the same feature, only the first FEATURE line is used.

If you received one or more FEATURE lines, update your license by adding the lines to your existing license file.

- 
- 1 Edit your `license.dat` file using a text editor like `vi` or `emacs`.  
See [Viewing and editing the license file](#) on page 165.
  - 2 You should always 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.

- 3 If you want LSF 4.x and LSF 5.x clusters to share a license file, make sure your license includes the `FEATURE` line for `lsf_batch` version 4.x.
- 4 Reconfigure LSF using either of the following LSF commands:
  - ❖ `lsadmin reconfig`
  - ❖ `lsadmin restart` on the master LIM

The license file is re-read and the changes accepted by LSF. At this point, the LSF license has been updated. However, some products may also require installation or upgrade of LSF software before you can use the new functionality.

---

## Update a license with INCREMENT lines

---

- 1 If you received one or more `INCREMENT` lines, update your license by adding the lines to your existing license file.
- 2 Edit your `license.dat` file using a text editor like `vi` or `emacs`.  
See [Viewing and editing the license file](#) on page 165.
- 3 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.
- 4 Reconfigure LSF using either of the following LSF commands:
  - ❖ `lsadmin reconfig`
  - ❖ `lsadmin restart` on the master LIM

The license file is re-read and the changes accepted by LSF.

---

## FLEXlm Basics

This section is for users installing a permanent license, as FLEXlm is not used with demo licenses. Users who already know how to use FLEXlm will not need to read this section.

FLEXlm is used by many UNIX software packages because it provides a simple and flexible method for controlling access to licensed software. A single FLEXlm 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, since you do not need to install a new license manager every time you get a new package.

### Start the license daemons

FLEXlm uses license daemons to manage permanent licenses. For a brief description of FLEXlm and its license daemons, see [FLEXlm license server](#) on page 160.

This is a procedure that describes how to start the FLEXlm license daemons.

- 1 Log on to the license server host as LSF administrator.

---

**IMPORTANT:** Do not run `lmgrd` as root.

---

- 2 If you have an old `lsf_d` running, run `lmdown` to kill it.  
You can only have one `lsf_d` daemon running on a host.
- 3 Run the `lmgrd` command in `LSF_SERVERDIR` to start the license server daemon:

```
lmgrd -c /usr/share/lsf/lsf_62/conf/license.dat -l
/usr/share/lsf/lsf_62/log/license.log
```

The `-c` option specifies the license file (or license file list, if you have multiple license server hosts). For more information, see [LSF\\_LICENSE\\_FILE parameter](#) on page 166.

The `-l` option specifies the debug log path. For more information, see [FLEXlm log file](#) on page 173.

---

**TIP:** 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 `lmgrd` starts automatically during system reboot.

---

See [Checking the license server status](#) on page 172 to check the status of `lmgrd`.

---

### Checking the license server status

If you are using a permanent LSF license, use the `lmstat` command to check the status of the license server daemon. This check can tell you whether or not your attempt to start your license server daemon succeeded. If your attempt failed, see [lmgrd fails with message "Port already in use"](#) on page 189.

The `lmstat` command is in `LSF_SERVERDIR`. For example:

```
/usr/share/lsf/lsf_62/7.0/sparc-sol2/etc/lmstat
```

Run `lmstat -a -c LSF_LICENSE_FILE` from the FLEXlm license server and also from the LSF master host. You must use the `-c` option of `lmstat` to specify the path to the LSF license file.

The output of `lmstat` gives the status of:

- ◆ The license server daemon (`lmgrd`)
- ◆ The LSF vendor daemon (`lsf_ld`)
- ◆ The number of available licenses for each product in the license file

For example, depending on the LSF features installed, the output of the command should look something like the following:

```
lmstat -a -c $LSF_ENVDIR/license.dat
lmstat - Copyright (C) 1989-1997 Globetrotter Software, Inc.
Flexible License Manager status on Fri 10/15/1999 13:23

License server status: 1711@hostA
 License file(s) on hostA: /usr/local/cluster1/mnt/conf/license.dat:

 hostA: license server UP (MASTER) v5.12

Vendor daemon status (on hostA):

 lsf_ld: UP v5.12

Feature usage info:
Users of lsf_base: (Total of 50 licenses available)

 "lsf_base" v4.100, vendor: lsf_ld
floating license

root hostB /dev/tty (v3.0) (hostA/1711 401), start Thu 10/14 12:32, 20 licenses
...
```

## FLEXlm log file

Read this to familiarize yourself with the FLEXlm log file.

The FLEXlm 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 FLEXlm log file at any time.

You must choose a location for the log file when you start the license daemon. If you already have FLEXlm server running for other products and Platform LSF licenses are added to the existing license file, then the log messages for FLEXlm should go to the same log file you set up for other products. If FLEXlm is dedicated to managing LSF licenses, you can put the FLEXlm log in the same directory as your other system logs, or in the `/tmp` directory.

## License management utilities

FLEXlm 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 FLEXlm 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 FLEXlm utilities in `LSF_SERVERDIR`:

|                 |                                                                           |
|-----------------|---------------------------------------------------------------------------|
| <b>lmcksum</b>  | Calculate check sums of the license key information                       |
| <b>lmdown</b>   | Shut down the FLEXlm server                                               |
| <b>lmhostid</b> | Display the hardware host ID                                              |
| <b>lmremove</b> | Remove a feature from the list of checked out features                    |
| <b>lmreread</b> | Tell the license daemons to re-read the license file                      |
| <b>lmstat</b>   | Display the status of the license server daemons and checked out licenses |
| <b>lmver</b>    | Display the FLEXlm version information for a program or library           |

For complete details on these commands, see the FLEXlm man pages.

## Multiple FLEXlm 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. Compare with [Selecting a license server host](#) on page 168 to make an educated decision.

Although it is not necessary, you may want to understand how the FLEXlm license server behaves prior to setting up your license server hosts. For a brief description on how FLEXlm works, see [FLEXlm license server](#) on page 160.

If you are concerned about the reliability of your license server host, you can distribute the LSF licenses across multiple FLEXlm 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. For more information, see [Distributed license server hosts](#) on page 175.
- ◆ Single license file with three redundant license server hosts. For more information, see [Redundant license server hosts](#) on page 176.

### Distributed 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. This section describes distributed license server hosts. See [Redundant license server hosts](#) on page 176 for information on the other configuration.

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.

### Enable multiple license server hosts

See the procedures for installing and configuring a permanent license. There are a few differences when you use distributed license server hosts:

- 1 See [Getting a permanent license](#) on page 165. You must obtain multiple license files, with your total number of licenses divided appropriately among the license server hosts. You must provide the following information for each license server host:
  - ❖ Host name and FLEXlm host ID
  - ❖ The products and number of licenses you want to be managed by this host
- 2 See [LSF\\_LICENSE\\_FILE parameter](#) on page 166. 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 See [Start the license daemons](#) on page 172. 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
```

## 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. This section describes redundant license server hosts. See [Distributed license server hosts](#) on page 175 for information on the second configuration.

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 (`lmgrd`) and LSF vendor license daemon (`lsf_ld`) 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.

## Enable multiple license server hosts

See the procedures for installing and configuring a permanent license. There are a few differences when you use redundant license server hosts:

- 1 See [Getting a permanent license](#) on page 165. You must obtain a license file that contains three `SERVER` lines. You must provide the following information for each license server host:
  - ❖ Host name and FLEXlm host ID
- 2 See [LSF\\_LICENSE\\_FILE parameter](#) on page 166. Specify the location of all the licenses in `LSF_LICENSE_FILE`, not just one. Use a colon (`:`) on UNIX and Linux or a semicolon (`;`) on Windows to separate each location. List the primary license server host first (the one you want LSF to contact first).
- 3 See [Start the license daemons](#) on page 172. 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

This section applies to permanent licenses. Read this if you have a cluster in which not all of the hosts will require licenses for the same LSF products. In this section, you will learn how to save money through distributing your licenses efficiently.

Not all hosts in the cluster need to be licensed for the same set of LSF products. For example, some hosts might be licensed only for Platform LSF Make, while others may also be licensed to run Platform MultiCluster. All hosts in the cluster remain licensed regardless of the license configuration of the rest of the cluster. If hosts become unavailable or new hosts are added, licenses are redistributed according to the new configuration.

This allows you to purchase only as many licenses as you need, rather than enabling the entire cluster for products that are only needed by a few hosts.

However, many LSF products do not support partial licensing. They must be enabled for the entire cluster, or not at all.

### Setting priority for license distribution

This describes how to define the order your licenses are given out to hosts.

To enable LSF server hosts to run partially licensed LSF products, edit the `Host` section of `LSF_CONFDIR/lsf.cluster.cluster_name` and include the product names in the `RESOURCES` column for specific hosts. 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`. You can see the order in which licenses are distributed with the command `lshosts`.

### Displaying licensed products

This describes how to view what products are licensed for any host in the cluster.

Use the `lshosts -l` command.

```
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
```

## Example of partial licensing

Here is an example that will allow you to better visualize the concept of partial licensing. Through this example, you can learn how to configure your hosts to use partial licensing.

### Scenario

In the following configuration, the license file contains licenses for LSF, Platform LSF Make and Platform LSF MultiCluster. The licenses have the following distribution:

- ◆ 3 LSF
- ◆ 1 Platform LSF Make
- ◆ 1 Platform MultiCluster

All three single-CPU hosts in the cluster are licensed for LSF, while `hostA` and `hostC` are licensed for Platform LSF Make. The `hostB` is explicitly licensed for Platform MultiCluster, and not for Platform LSF Make. The `RESOURCES` field in the Host section of `lsf.cluster.cluster_name` must contain the LSF products `LSF_Base` and `LSF_Manager`, in addition to `LSF_MultiCluster`. The rest of the licenses needed are enabled by

### Configuration

The `lsf.cluster.cluster_name` file contains the following configuration:

```
Begin Parameters
PRODUCTS=LSF_Base LSF_Manager LSF_Make
End Parameters

Begin Host
HOSTNAME model type server r1m mem swp RESOURCES
hostA DEFAULT LINUX86 1 - () () ()
hostB DEFAULT LINUX86 1 - () () (LSF_Base LSF_Manager LSF_Make)
hostC DEFAULT LINUX86 1 - () () ()
End Host
```

### Cluster startup

At cluster startup, all hosts are running, and the `lshosts -l` command displays the following license distribution:

```
lshosts -l

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

RESOURCES: Not defined
RUN_WINDOWS: (always open)

: (LSF_Base LSF_Manager LSF_Make)

LOAD_THRESHOLDS:
 r15s r1m r15m ut pg io ls it tmp swp mem
 - - - - - - - - - - -

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

RESOURCES: (LSF_Base LSF_Manager LSF_MultiCluster)
```

RUN\_WINDOWS: (always open)

LICENSES\_ENABLED: (LSF\_Base LSF\_Manager)

LOAD\_THRESHOLDS:

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

HOST\_NAME: hostC

| type    | model   | cpuf  | ncpus | ndisks | maxmem | maxswp | maxtmp | rexpri | server | nprocs | ncores | nthreads |
|---------|---------|-------|-------|--------|--------|--------|--------|--------|--------|--------|--------|----------|
| LINUX86 | DEFAULT | 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 |
|------|-----|------|----|----|----|----|----|-----|-----|-----|
| -    | -   | -    | -  | -  | -  | -  | -  | -   | -   | -   |

All hosts are licensed for the appropriate products except hostC, which does not have Platform LSF Make because its license is already being used by hostA. However, hostC is still available to run LSF jobs.

**Master host failover** If HostA becomes unavailable, HostB becomes master host. Now the `lshosts -l` command displays the following license distribution:

`lshosts -l`

HOST\_NAME: hostA

| type    | model   | cpuf  | ncpus | ndisks | maxmem | maxswp | maxtmp | rexpri | server | nprocs | ncores | nthreads |
|---------|---------|-------|-------|--------|--------|--------|--------|--------|--------|--------|--------|----------|
| LINUX86 | DEFAULT | 116.1 | 2     | 1      | 2016M  | 1983M  | 72917M | 0      | Yes    | 1      | 1      | 2        |

RESOURCES: Not defined

RUN\_WINDOWS: (always open)

LICENSES\_ENABLED: (LSF\_Client)

LOAD\_THRESHOLDS:

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

HOST\_NAME: hostB

| type    | model   | cpuf  | ncpus | ndisks | maxmem | maxswp | maxtmp | rexpri | server | nprocs | ncores | nthreads |
|---------|---------|-------|-------|--------|--------|--------|--------|--------|--------|--------|--------|----------|
| LINUX86 | DEFAULT | 116.1 | 2     | 1      | 2016M  | 1983M  | 72917M | 0      | Yes    | 1      | 1      | 2        |

RESOURCES: (LSF\_Base LSF\_Manager)

RUN\_WINDOWS: (always open)

LICENSES\_ENABLED: (LSF\_Base LSF\_Manager)

LOAD\_THRESHOLDS:

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

HOST\_NAME: hostC

| type    | model   | cpuf  | ncpus | ndisks | maxmem | maxswp | maxtmp | rexpri | server | nprocs | ncores | nthreads |
|---------|---------|-------|-------|--------|--------|--------|--------|--------|--------|--------|--------|----------|
| LINUX86 | DEFAULT | 116.1 | 2     | 1      | 2016M  | 1983M  | 72917M | 0      | Yes    | 1      | 1      | 2        |

## Partial Licensing

RESOURCES: Not defined

RUN\_WINDOWS: (always open)

LICENSES\_ENABLED: (LSF\_Base LSF\_Manager LSF\_Make)

LOAD\_THRESHOLDS:

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

**Note that hostC has now picked up the available Platform LSF Make license that was originally held by hostA.**

## Floating Client Licenses

LSF floating client is valuable if you have a cluster in which not all of the hosts will be active at the same time. In this section, you will learn how to save money through distributing your licenses efficiently.

An LSF floating client license is a type of LSF license to be shared among several client hosts at different times. Floating client licenses are not tied to specific hosts. They are assigned dynamically to any host that submits a request to LSF. The number of licenses acts as a license pool for the cluster from which LSF clients can draw required licenses. Although floating client licenses are supported, LSF does not support floating server licenses.

### Client hosts and floating client hosts

In LSF, you can have both client hosts and floating client hosts. The difference is in the type of license purchased.

If you purchased a regular (fixed) client license, LSF client hosts are static. The 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.

If you purchased a floating client license, LSF floating client hosts are dynamic. They are not listed in `lsf.cluster.cluster_name`. Since LSF does not take into account the host name but the number of floating licenses, clients can change dynamically and licenses will be distributed to clients that request to use LSF. When you submit a job from any unlicensed host, and if there are any floating licenses free, the host will check out a license and submit your job to LSF. However, once a host checks out a floating 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 floating client license becomes available to any other host that needs it.

### How floating licenses work in LSF

Read this to understand how floating licenses work. You will want to read this before configuring your cluster to use this distribution technique.

When the master LIM starts up, it verifies how many floating licenses there are for the cluster as specified in `lsf.cluster.cluster_name` with the parameter `FLOAT_CLIENTS`. The master LIM checks out the licenses and keeps track of license information—how many floating licenses have been assigned, and which client hosts are using the licenses.

Floating client licenses expire at midnight (local time) on the day the license was issued. The master LIM checks the host list and removes any floating client hosts whose license has expired.

### License reset

Whenever the master LIM is restarted, all LSF floating client licenses are released and checked out again.

## Administration commands

Since LSF floating client hosts are not listed in `lsf.cluster.cluster_name`, some administration commands will not work if issued from LSF floating client hosts. Always run administration commands from server hosts.

## 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 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

## Install LSF floating client licenses

If you believe that the use of floating client licenses is appropriate for your needs, follow this procedure to install LSF floating client licenses.

---

### 1 Obtain the floating client license.

This is similar to getting any other license. See [Getting a permanent license](#) on page 165.

### 2 Update your license file. Add the appropriate license line in the license file `license.dat`. The LSF license must contain FEATURE lines for `LSF_Float_Client`.

Although LSF Floating Client requires a license, `LSF_Float_Client` does not appear in the `PRODUCTS` line. `LSF_Float_Client` also cannot be added as a resource for specific hosts already defined in `lsf.cluster.cluster_name`. Should these lines be present, they are ignored by LSF.

### 3 Define server hosts in `lsf.conf`.

As with any client host, specify the parameter `LSF_SERVER_HOSTS` in `lsf.conf` to define LSF server hosts for the LSF client hosts to contact.

### 4 Edit `lsf.cluster.cluster_name` by adding or uncommenting the `FLOAT_CLIENTS` parameter in the `Parameters` section:

```
...
Begin Parameters
PRODUCTS=LSF_Base LSF_Manager LSF_Make
```

```

FLOAT_CLIENTS= 25
End Parameters
...

```

The `FLOAT_CLIENTS` parameter sets the size of your license pool in the cluster. When the master LIM starts up, the number of licenses specified in `FLOAT_CLIENTS` (or fewer) can be checked out for use as floating client licenses.

If the parameter `FLOAT_CLIENTS` is not specified in `lsf.cluster.cluster_name`, or there is an error in either `license.dat` or in `lsf.cluster.cluster_name`, the floating LSF client license feature is disabled.

5 Start the license server daemon.

See [Start the license daemons](#) on page 172.

6 To allow your changes to take effect, reconfigure the cluster with the commands:

```

lsadmin reconfig
badmin mbdrestart

```

---

**CAUTION:** When the LSF floating client license feature is enabled, any host can submit jobs to the cluster. You can limit which hosts can be LSF floating clients. See [Security issues with floating client licenses](#) on page 183.

---

## Security issues with floating client licenses

If you want to install or have installed floating client licenses, it is important that you read this section to inform yourself of the security issues. There are measures to compensate for these security issues (see [Configuring security for LSF floating client licenses](#) on page 183).

With LSF client licenses, 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 client licenses, 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.

## Configuring security for LSF floating client licenses

Read this section to learn how to configure security against the issues presented in [Security issues with floating client licenses](#) on page 183.

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_CLIENTS_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_CLIENTS_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_CLIENTS_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_CLIENTS_ADDR_RANGE=12.23.45.*
```

All client hosts belonging to domains starting with 12.23.45 are allowed. No IPv6 hosts are allowed.

```
FLOAT_CLIENTS_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_CLIENTS_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_CLIENTS_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:fffe::88bb:*
```

Expands to 3ffe:fffe:0:0:0:88bb:\*. All IPv6 client hosts belonging to domains starting with 3ffe:fffe::88bb:\* are allowed. No IPv4 hosts are allowed.

```
FLOAT_CLIENTS_ADDR_RANGE = 3ffe-4fff:fffe::88bb:aa-ff 12.23.45.*
```

All IPv6 client hosts belonging to domains starting with 3ffe up to 4fff, then fffe::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-*:fffe::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.

## Checking 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/lim.log.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.

## Verify LSF floating client license is working

Perform this procedure after setting up your floating client license to verify that your floating client license is enabled.

- 1 Start a cluster.
- 2 Run the `lshosts` command from a host listed in `lsf.cluster.cluster_name`:

```
lshosts
```

In the following example, only `hostA` and `hostB` are defined in `lsf.cluster.cluster_name`. `hostA` is a server and master host, and `hostB` is a static client. If you type the command from `hostA` or `hostB`, you will get the following output:

```
lshosts
HOST_NAME type model cpuf ncpus maxmem maxswp server RESOURCES
hostA SUNSOL DEFAULT 1.0 1 128M 602M Yes ()
hostB SUNSOL DEFAULT 1.0 - - - No ()
```

- 3 Submit a job from a host not listed in `lsf.cluster.cluster_name`.

For example, if you submitted the following job from `hostC`:

```
bsub sleep 1000
```

You would get the following response:

```
Job <104> is submitted to default queue <normal>.
```

- 4 From any LSF host, with `LSF_ENVDIR` set to this cluster, run the `lshosts` command:

```
lshosts
HOST_NAME type model cpuf ncpus maxmem maxswp server RESOURCES
hostA SUNSOL DEFAULT 1.0 1 128M 602M Yes ()
hostB SUNSOL DEFAULT 1.0 - - - No ()
hostC UNKNOWN UNKNOWN 1.0 - - - No ()
```

In the above example, although `hostC` shows the type `UNKNOWN` and `hostA` and `hostB` are of type `SUNSOL` (Sun Solaris), the job will be allowed to be executed on any host type because `hostC` is a floating client host without any model or type restrictions specified at job submission.

- 5 From any host, run the `lshosts -l` command:

```
lshosts -l hostC
```

where `hostC` is a floating client host.

```
HOST_NAME: hostC
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server
UNKNOWN UNKNOWN 1.0 - - - - - - No
```

```
RESOURCES: Not defined
```

```
RUN_WINDOWS: Not applicable for client-only host
```

```
LICENSES_ENABLED: (LSF_Float_Client)
```

## Troubleshooting License Issues

- ◆ "lsadmin reconfig" gives "User permission denied" message on page 187
- ◆ Primary cluster administrator receives email "Your cluster has experienced license overuse" message on page 187
- ◆ lsadmin command fails with "ls\_gethostinfo: Host does not have a software license" on page 187
- ◆ LSF commands give "Host does not have a software license" on page 188
- ◆ LSF commands fail with "ls\_initdebug: Unable to open file lsf.conf" on page 188
- ◆ lmgrd fails with message "Port already in use" on page 189

### "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 `badmin` might give the error message "User permission denied".

Use the following commands to change the ownership for `lsadmin` and `badmin` to root and the file permission mode to `-rwsr-xr-x`:

```
chown root lsadmin badmin
chmod 4755 lsadmin badmin
```

Now the user ID bit for the owner is `setuid`. If `lsadmin` and `badmin` 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 `badmin` 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.

## LSF commands give "Host does not have a software license"

You may see this message after running `lsid`, `lshosts`, or other `ls*` commands.

Typical problems and their solutions:

| If you experience this problem ...                                                                                                                                                                                                                                   | Do the following:                                                                                                                                                                                                                                                                                           |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Your demo license (not tied to FLEXlm server) has expired.                                                                                                                                                                                                           | Check the <code>license.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:<br>The license file may have more than one FEATURE on a line.<br>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>dos2unix</code> to remove the Windows line breaks (^M characters) from the license file.<br>If the license key is tied to a FLEXlm server, restart <code>lmgrd</code> .<br>Restart the master LIM. |
| The LSF master host is unable to communicate with the FLEXlm server.                                                                                                                                                                                                 | Check the network communication by entering the command:<br><code>ping FLEXlm_server</code>                                                                                                                                                                                                                 |
| License daemons ( <code>lmgrd</code> and <code>lsf_ld</code> ) are not running on the FLEXlm server.                                                                                                                                                                 | Check if <code>lmgrd</code> and <code>lsf_ld</code> are running by typing:<br><code>ps -ef   egrep 'lmgrd lsf_ld'</code><br>on the FLEXlm server. If not:<br>Check the <code>license.log</code> file for error messages.<br>Start <code>lmgrd</code> .<br>Restart the master LIM.                           |

## LSF commands fail with "ls\_initdebug: Unable to open file lsf.conf"

You might see this message after running `lsid`. This message indicates that the LSF commands cannot access the `lsf.conf` file or `lsf.conf` does not exist in `LSF_ENVDIR`.

Solution:

- ◆ Use `LSF_CONFDIR/csrhc.lsf` or `LSF_CONFDIR/profile.lsf` to set up your LSF environment, or
- ◆ If you know the location of `lsf.conf`, set the `LSF_ENVDIR` environment variable to point to the directory containing the `lsf.conf` file.

## lmgd fails with message "Port already in use"

The port number defined in `LSF_LICENSE_FILE` and `license.dat` is being used by another application (by default, LSF uses port number 1700).

Possible causes:

| If you experience this problem ...                                                   | Do the following:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>lmgd</code> is already running for this license                                | Use <code>ps -ef</code> and make sure that <code>lmgd</code> and <code>lsf_lid</code> are not running.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <code>lmgd</code> 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>If the port number is being used by another application, execute the following to change the port number used by LSF:</p> <ol style="list-style-type: none"> <li>1 Edit <code>license.dat</code> and change the port number in the line:<br/> <pre>SERVER flexlm_server 3f8b6a3 1700</pre>           The fourth field on the <code>SERVER</code> line of <code>license.dat</code> specifies the TCP port number that the FLEXlm server uses. Choose an unused port number. The default port set by FLEXlm is 1700. Platform LSF usually uses port numbers in the range 3879 to 3882, so the numbers from 3883 forward are good alternate choices.</li> <li>2 In <code>lsf.conf</code>:           <ul style="list-style-type: none"> <li>❖ If <code>LSF_LICENSE_FILE</code> is defined as follows:<br/> <code>LSF_LICENSE_FILE=port_number@flexlm_server</code> (for example: <code>1700@hostA</code>), the port number must be changed accordingly.</li> <li>❖ If <code>LSF_LICENSE_FILE</code> points to the license file path (for example:<br/> <code>LSF_LICENSE_FILE=/usr/local/lsf/conf/license.dat</code>), no changes are required.</li> </ul> </li> </ol> <p>a Restart <code>lmgd</code>.</p> |



# Managing LSF on Platform EGO

## Contents

- ◆ [About LSF on Platform EGO](#) on page 192
- ◆ [LSF and EGO directory structure](#) on page 195
- ◆ [Configuring LSF and EGO](#) on page 199
- ◆ [Managing LSF daemons through EGO](#) on page 202
- ◆ [Administrative Basics](#) on page 205
- ◆ [Logging and troubleshooting](#) on page 206
- ◆ [Frequently asked questions](#) on page 213

## 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 7, 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 FlexLM 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 does Platform EGO work?

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).

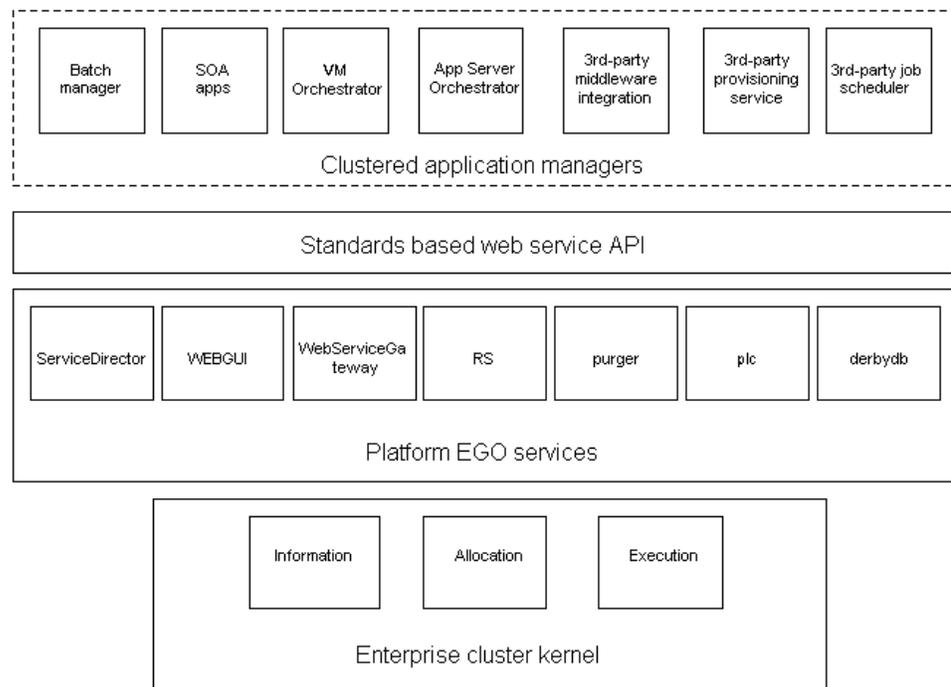
Once the consumer receives its allotted resources from Platform EGO, the consumer applies its own rules and policies. How the consumer decides to balance its workload across the fixed resources allotted to it is not the responsibility of EGO.

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** 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. Resources also have attributes. For example, a host has attributes of memory, processor utilization, operating systems type, etc.
- 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** 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.
- The distribution priority is to satisfy each consumer's reserved ownership, then distribute remaining resources to consumers that have demand.
- Services** 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.
- Run `egosh` to check service status.
- If EGO is disabled, the `egosh` command cannot find `ego.conf` or cannot contact `vemkd` (not started), and the following message is displayed:
- ```
You cannot run the egosh command because the administrator has chosen not to enable EGO in lsf.conf: LSF_ENABLE_EGO=N.
```
- EGO user accounts** 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.

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/7.0	LSF 7.0 binaries and other machine dependent files	Non-writable
LSF_TOP/conf	LSF 7.0 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 7.0 log files	Writable by all hosts in the cluster
LSF_TOP/work	LSF 7.0 working directory	Writable by the master host and master candidate hosts, and is accessible to slave hosts

EGO, GUI, and PERF directories

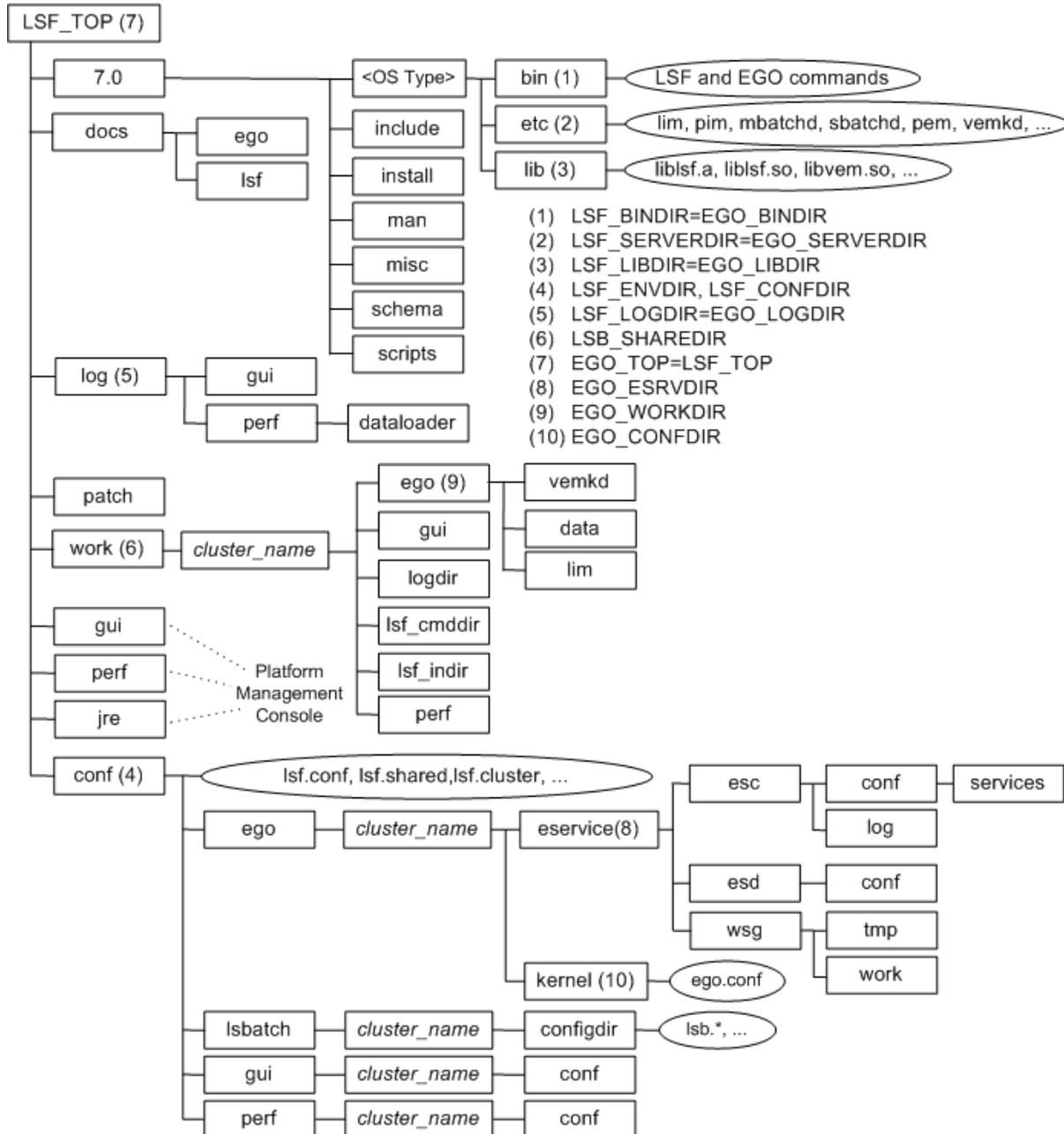
Directory Path	Description	Attribute
LSF_BINDIR	EGO binaries and other machine dependent files	Non-writable
LSF_LOGDIR/ego/cluster_name/eservice (EGO_ESRVDIR)	EGO services configuration and log files.	Writable
LSF_LOGDIR/ego/cluster_name/kernel (EGO_CONFDIR, LSF_EGO_ENVDIR)	EGO kernel configuration, log files and working directory, including conf/log/work	Writable
LSB_SHAREDIR/cluster_name/ego (EGO_WORKDIR)	EGO working directory	Writable
LSF_TOP/perf/1.2	PERF commands, library and schema	Non-writable
LSF_LOGDIR/perf/cluster_name/conf (PERF_CONFDIR)	PERF configuration	Writable
LSB_SHAREDIR/cluster_name/perf/data (PERF_DATADIR)	PERF embedded data files for derby	Writable
LSF_TOP/perf/1.2/etc	PERF script command for services	Non-writable
LSF_TOP/log/perf (PERF_LOGDIR)	PERF log files	Writable
LSB_SHAREDIR/cluster_name/perf (PERF_WORKDIR)	PERF working directory	Writable
LSF_TOP/jre	Java Runtime Environment	Non-writable
LSF_TOP/gui	GUI	Non-writable
LSF_LOGDIR/gui/cluster_name/conf (GUI_CONFDIR)	GUI configuration	Writable
LSB_SHAREDIR/cluster_name/gui (CATALINA_WORKDIR, CATALINA_TMPDIR)	GUI working directory	Writable
LSF_TOP/log/gui (GUI_LOGDIR)	GUI log files	Writable
LSF_TOP/gui/2.0/	GUI binaries and tomcat	Non-writable
LSF_TOP/gui/2.0/tomcat	Tomcat web server	Writable

NOTE: Several directories under LSF_TOP/gui/1.2/tomcat are writable by Tomcat servers. You should install the whole Tomcat directory on a writable file system.

Example directory structures

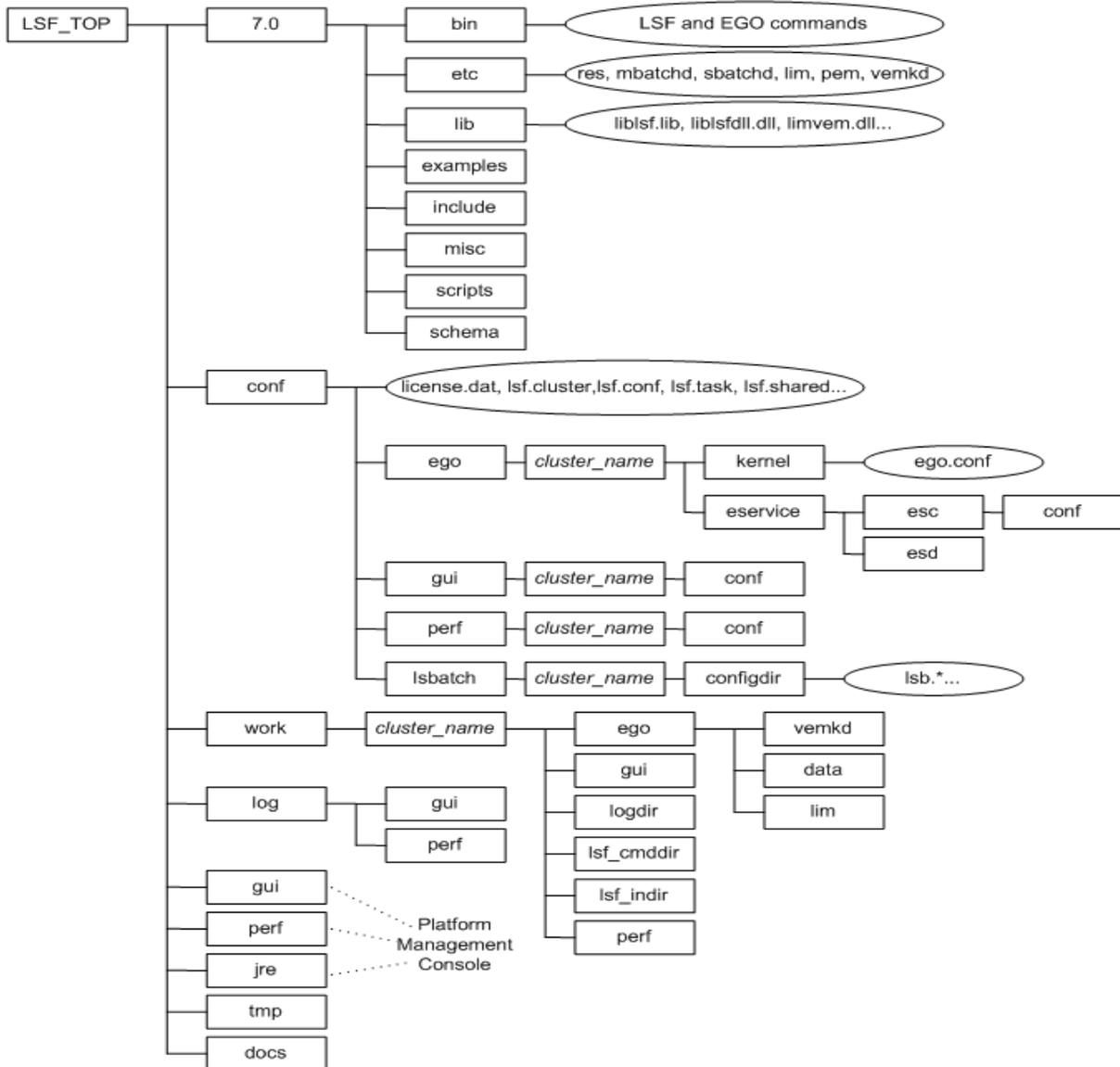
UNIX and Linux

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.



LSF and EGO directory structure

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



Configuring LSF and EGO

EGO configuration files for 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`.
- ◆ `sbatchd.xml`—EGO service configuration file for `sbatchd`.

When LSF daemon control through EGO Service Controller is configured, `lsadmin` uses the reserved EGO service name `res` to control the LSF `res` daemon, and `badmin` uses the reserved EGO service name `sbatchd` to control the LSF `sbatchd` 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`.

LSF and 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 LSF 7

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

On every host, a set of connection ports must be free for use by LSF and EGO components.

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 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.

Managing 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

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.
res	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. Under EGO Service Controller mode, res is started by pem as an EGO service on every host.

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 `inittab`) and use `lsadmin` and `badmin` 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 `badmin` to start LSF manually, but internally, `lsadmin` and `badmin` 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 `badmin hshutdown` and `lsadmin resshutdown` to manually shut down LSF, the LSF daemons are not restarted automatically by EGO. You must run `lsadmin resstartup` and `badmin 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 *Platform 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`)

Prerequisites: 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 `badmin` invoke the `egosh user logon` 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.

EGO control of PMC and PERF services

When EGO is enabled in the cluster, EGO may control services for components such as the Platform Management Console (PMC) or LSF Reports (PERF). This is recommended. It allows failover among multiple management hosts, and allows EGO cluster commands to start, stop, and restart the services.

PMC not controlled by EGO

For PMC, if it is not controlled by EGO, you must specify the host to run PMC. Use the `pmcadmin` command to start and stop PMC. Use the `pmcsetrc.sh` command to enable automatic startup on the host (the daemon will restart if the host is restarted).

PERF services not controlled by EGO

For PERF, if the services are not controlled by EGO, you must specify the host to run PERF services `plc`, `jobdt`, and `purger`. Use the `perfadmin` command to start and stop these services on the host. Use the `perfsetrc.sh` command to enable automatic startup of these services on the host (the daemons will restart if the host is restarted). If the PERF host is not the same as the Derby database host, run the same commands on the Derby database host to control `derbydb`.

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 `cshrc.lsf`.
`source LSF_TOP/conf/cshrc.lsf`
 - ◆ For `sh`, `ksh`, or `bash`, use `profile.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`), `cshrc.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 *Platform LSF Configuration Reference* for more information about `cshrc.lsf` and `profile.lsf`.

Logging and troubleshooting

LSF log files

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.

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`.

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

LSF log directory permissions and ownership

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`.

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)	<code>esc.log.hostname</code>
named	<code>named.log.hostname</code>
PEM (Process Execution Manager)	<code>pem.log.hostname</code>
VEMKD (Platform LSF Kernel Daemon)	<code>vemkd.log.hostname</code>
WSM (Platform Management Console/WEBGUI)	<code>wsm.log.hostname</code>
WSG (Web Service Gateway)	<code>wsg.log</code>

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).
- ◆ Platform Management Console log files (WSM and Catalina) are found in `LSF_LOGDIR/gui` (Linux) or `LSF_LOGDIR\gui` (Windows).
- ◆ Web service gateway log files are found in `LSF_LOGDIR/ego/cluster_name/eservice/wsg/log` (Linux) or `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) or `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
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.

Number	Level	Description
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.

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 `wsm` daemon reads `wsm.conf` to retrieve the following information:

- ◆ `LOG_LEVEL`: The configured log class controlling the level of event information that is logged (`INFO`, `ERROR`, `WARNING`, or `DEBUG`)

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.

TIP: 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
catalina.out	Linux: LSF_LOGDIR/gui/catalina.out Windows: LSF_LOGDIR\gui\catalina.out	Logs system errors and debug information from Tomcat web server startup.
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. Errors surrounding Platform Management Console startup are logged here.
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.

Log file	Default location	What it contains
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).
wsm.log	Linux: LSF_LOGDIR/gui/wsm.log. <i>hostname</i> Windows: LSF_LOGDIR\gui\wsm.log. <i>hostname</i>	Logs information collected by the web server monitor daemon. Failures of the WEBGUI service that runs the Platform Management Console are logged here.

Matching 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
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
JAVA_HOME is not defined, exit	WEBGUI service profile is wrong	wsm
failed to get hostname: %s	Host name configuration problem	wsm
event_init () failed	EGO event plugin configuration problem in ego.conf file	wsm
ego.conf_loadeventplug () failed	Event library problem	wsm
cannot write to child	Web server is down or there is no response	wsm
child no reply	Web server is down or there is no response	wsm
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

For more information

- ◆ About Platform LSF logging and troubleshooting, see [Error and Event Logging](#) on page 757 and [Troubleshooting and Error Messages](#) on page 769
- ◆ About Platform EGO loggings and troubleshooting, see *Administering and Using Platform EGO*

Frequently asked questions

- Question** *Does LSF 7 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 7 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 7?*
- Answer** LSF 7 hostsetup 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 7, 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 7, 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 7 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?*

Frequently asked questions

Answer LSF 7 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.

The Platform Management Console

Contents

- ◆ [Log on to the Platform Management Console](#) on page 215
- ◆ [Set the command-line environment](#) on page 216
- ◆ [Manage services](#) on page 216
- ◆ [Manage hosts](#) on page 218
- ◆ [Customize job submission interfaces](#) on page 219
- ◆ [Remove a job submission interface](#) on page 219

Log on to the Platform Management Console

You can use the `pmcadmin` command to administer the Platform Management Console. For more information, see the *Platform LSF Command Reference*.

The Platform Management Console (PMC) allows you to monitor, administer, and configure your cluster. To log on, give the name and password of the LSF administrator.

- 1 Browse to the web server URL and log in to the PMC using the LSF administrator name and password.

The web server URL is:

```
http://host_name:8080/platform
```

The host name is the PMC host.

If PMC is controlled by EGO and you have not specified a dedicated PMC host, PMC could start on any management host. To find out, log on to a command console as LSF administrator and run:

```
egosh client view GUIURL_1
```

The description part of the command output shows the full URL including the host name and port.

If PMC is not controlled by EGO, and you do not know the port, log on to the PMC host as LSF administrator and run:

Set the command-line environment

```
pmcadmin list
```

The command output shows the port you need to use in the URL.

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 `cshrc.lsf`.
`source LSF_TOP/conf/cshrc.lsf`
 - ◆ For `sh`, `ksh`, or `bash`, use `profile.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`), `cshrc.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 *Platform LSF Configuration Reference* for more information about `cshrc.lsf` and `profile.lsf`.

Manage services

Determine the host address where the service is running

Prerequisites: You need to know the address of where the service director (DNS server) is running (contact IT for assistance, if required).

EGO services may not all run on the same management host. You can use `nslookup` to find the address of the host where a specific service is running.

- 1 Using the CLI, type `nslookup`.
- 2 Type `server` and then enter the IP address of the service director (DNS server).

The Default Server and Address will return. For example,

```
> server 172.25.237.37
Default server: 172.25.237.37
Address: 172.25.237.37#53
```

- 3 Enter the name of the service for which you want to find the host address. For example,

```
> WEBGUI.ego
Server:          172.25.237.37
Address:         172.25.237.37#53
```

```
Name:   WEBGUI.ego
Address: 172.25.237.37
```

Troubleshoot service error states

If you receive a service error message, or a message indicating that a service will not transition out of the `Allocating` state, there are steps you can perform to troubleshoot the issue.

Responding to service message

Error

Normally, Platform EGO attempts to start a service multiple times, up to the maximum threshold set in the service profile XML file (containing the service definition). If the service cannot start, you will receive a service error message.

- 1 Try stopping and then restarting the service.
- 2 Review the appropriate service instance log file to discover the cause of the error.

Platform EGO service log files include those for the service director (`ServiceDirector`), web service gateway (`WebServiceGateway`), and the Platform Management Console (`WEBGUI`). If you have defined your own non-EGO services, you may have other log files you will need to review, depending on the service which is triggering the error.

Responding to service message

Allocating

`Allocating` is a transitional service state before the service starts running. If your service remains in this state for some time without transitioning to `Started`, or cycles between `Defining` and `Allocating`, you will want to discover the cause of the delay.

- 1 If you are the cluster administrator, review the allocation policy.
 - a Open the service profile XML file (containing the service definition).
 - b Find the consumer for which the service is expected to run.
 - c Ensure that a proper resource plan is set for that consumer.

During a service's "allocation" period, Platform EGO attempts to find an appropriate resource on which to run the service. If it cannot find the required resource, the service will not start.

Manage hosts

Important host roles

Hosts in the cluster may be described as the master host, master candidates, management hosts, compute hosts, or the web server host.

Master host

A cluster requires a master host. This is the first host installed. The master host controls the rest of the hosts in the grid.

Master candidates

There is only one master host at a time. However, if the master host ever fails, another host automatically takes over the master host role, allowing work to continue. This process is called failover. When the master host recovers, the role switches back again.

Hosts that can act as the master are called master candidates. This includes the original master host and all hosts that can take over the role in a failover scenario. All master candidates must be management hosts.

Master host failover

During master host failover, the system is unavailable for a few minutes while hosts are waiting to be contacted by the new master.

The master candidate list defines which hosts are master candidates. By default, the list includes just one host, the master host, and there is no failover. If you configure additional candidates to enable failover, the master host is first in the list. If the master host becomes unavailable, the next host becomes the master. If that host is also unavailable, the next host is considered to become the master, and so on down the list. A short list with two or three hosts is sufficient for practical purposes.

For failover to work properly, the master candidates must share a file system and the shared directory must always be available.

IMPORTANT: The shared directory should not reside on a master host or any of the master candidates. If the shared directory resides on the master host and the master host fails, the next candidate cannot access the necessary files.

Management host

Management hosts belong to the `ManagementHosts` resource group. These hosts are not expected to execute workload units for users. Management hosts are expected to run services such as the web server and web services gateway. The master host and all master candidates must be management hosts.

A slot is the basic unit of resource allocation, analogous to a "virtual CPU".

Management hosts share configuration files, so a shared file system is needed among all management hosts.

A management host is configured when you run `egoconfig mgghost` on the host. The tag `mg` is assigned to the management host, in order to differentiate it from a compute host.

Compute host

Compute hosts are distributed to cluster consumers to execute workload units. By default, compute hosts belong to the `ComputeHosts` resource group.

The `ComputeHosts` group excludes hosts with the `mg` tag, which is assigned to management hosts when you run `egoconfig mgghost`. If you create your own resource groups to replace `ComputeHosts`, make sure they also exclude hosts with the `mg` tag.

By default, the number of slots on a compute host is equal to the number of CPUs.

Web server host or PMC host

The web server is the host that runs the Platform Management Console, when you configure this you may call it the PMC host. There is only one host at a time acting as the web server host. If EGO controls the PMC, it does not need to be a dedicated host; by default, any management host in the cluster can be the web server (decided when the cluster starts up, failing over if the original host fails). However, if EGO does not control PMC, you must configure the PMC host manually. If you specify the PMC host, there can be no failover of PMC.

Customize job submission interfaces

From the Platform Console you can customize LSF application submission functionality, modify provided default applications (such as Fluent and NASTRAN), and add or remove application submission pages from the Platform Console.

Customizing, adding, and removing job submission interfaces requires you to manually change various XML configuration files and scripts.

- 1 Modify the submission script to point the field `APPLICATION_CMD` to an application. For example:

```
LSF_TOP/gui/lsf/7.0/batchgui/plugin/lsf/exec/fluent.cmd
#FLUENT_CMD="sleep 10;/bin/echo"
FLUENT_CMD="/pcc/app/fluent/Fluent.Inc/bin/fluent "
```

Remove a job submission interface

The Platform Console provides some default job submission interface templates. You can remove any of these, or other, interfaces from the organizational tree.

- 1 Using an XML editor, open this file:


```
LSF_ENVDIR/gui/cluster_name/conf/navigation/
pmc_navigation_jobsubmission.xml
```
- 2 Remove the XML element for the application submission node definition.
 - a Search for `<Name>Applications</Name>`.
 - b Under this section, delete or add comment markings around the `Category` section that contains the application you wish to remove.

For example:

```
<Name>Applications</Name>
<Display>Submission</Display>
```

Remove a job submission interface

```
<Description>Submission</Description>
<IconPath></IconPath>
<Role>1,3</Role>
<Categories>
<!--Category>
<Name>Fluent</Name>
<Display>Fluent</Display>
<Description></Description>
<IconPath>/batchgui/images/icon_treeNode_jobSubmission.gif</IconPath>
<Role>1,3</Role>
<Viewport-Ref>Fluent</Viewport-Ref>
</Category-->
<Category>...</Categories>
```

3 Click **Submit Job**.

Cluster Version Management and Patching on UNIX and Linux

IMPORTANT: *For LSF 7 Update 2 only, you cannot use the steps in this chapter to update your cluster from LSF 7 Update 1 to Update 3. You must follow the steps in "Migrating to LSF Version 7 Update 3 on UNIX and Linux" to manually migrate your LSF 7 cluster to Update 3.*

Contents

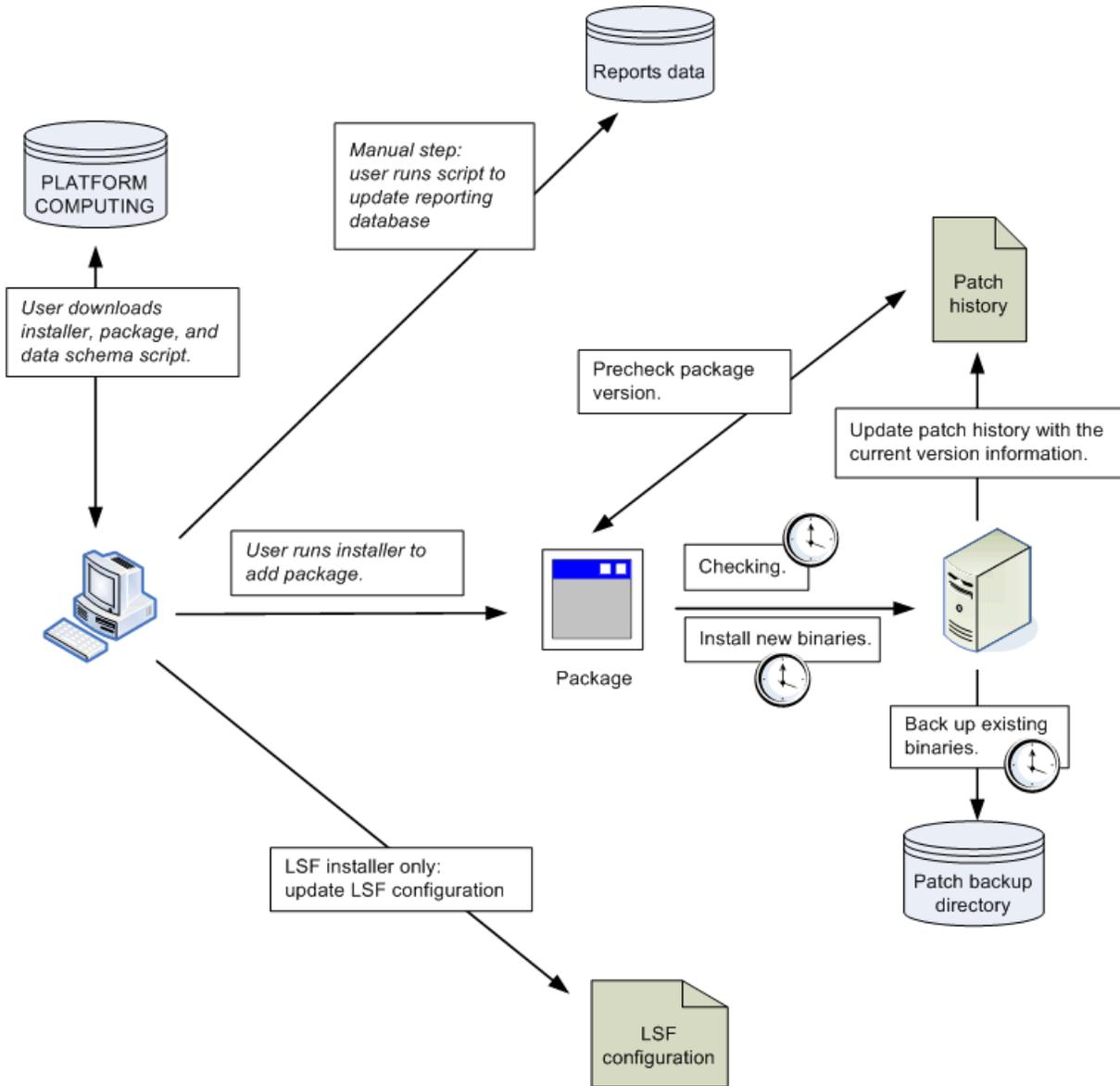
- ◆ [Scope](#) on page 222
- ◆ [Patch installation interaction diagram](#) on page 223
- ◆ [Patch rollback interaction diagram](#) on page 224
- ◆ [Version management components](#) on page 225
- ◆ [Version management concepts](#) on page 227
- ◆ [Cluster patching behavior table](#) on page 229
- ◆ [Cluster rollback behavior table](#) on page 230
- ◆ [Version management files](#) on page 230
- ◆ [Version management commands](#) on page 231
- ◆ [Installing update releases on UNIX and Linux](#) on page 232
- ◆ [Installing fixes on UNIX and Linux](#) on page 233
- ◆ [Rolling back patches on UNIX and Linux](#) on page 233
- ◆ [Patching the Oracle database](#) on page 234
- ◆ [Patching the Derby database](#) on page 235

Scope

Operating system	◆ Supports UNIX hosts within a single cluster
Limitations	pvversions supports LSF Update 1 and later patchinstall supports LSF Update 1 and later For installation of a new cluster, see <i>Installing Platform LSF on UNIX and Linux</i> .

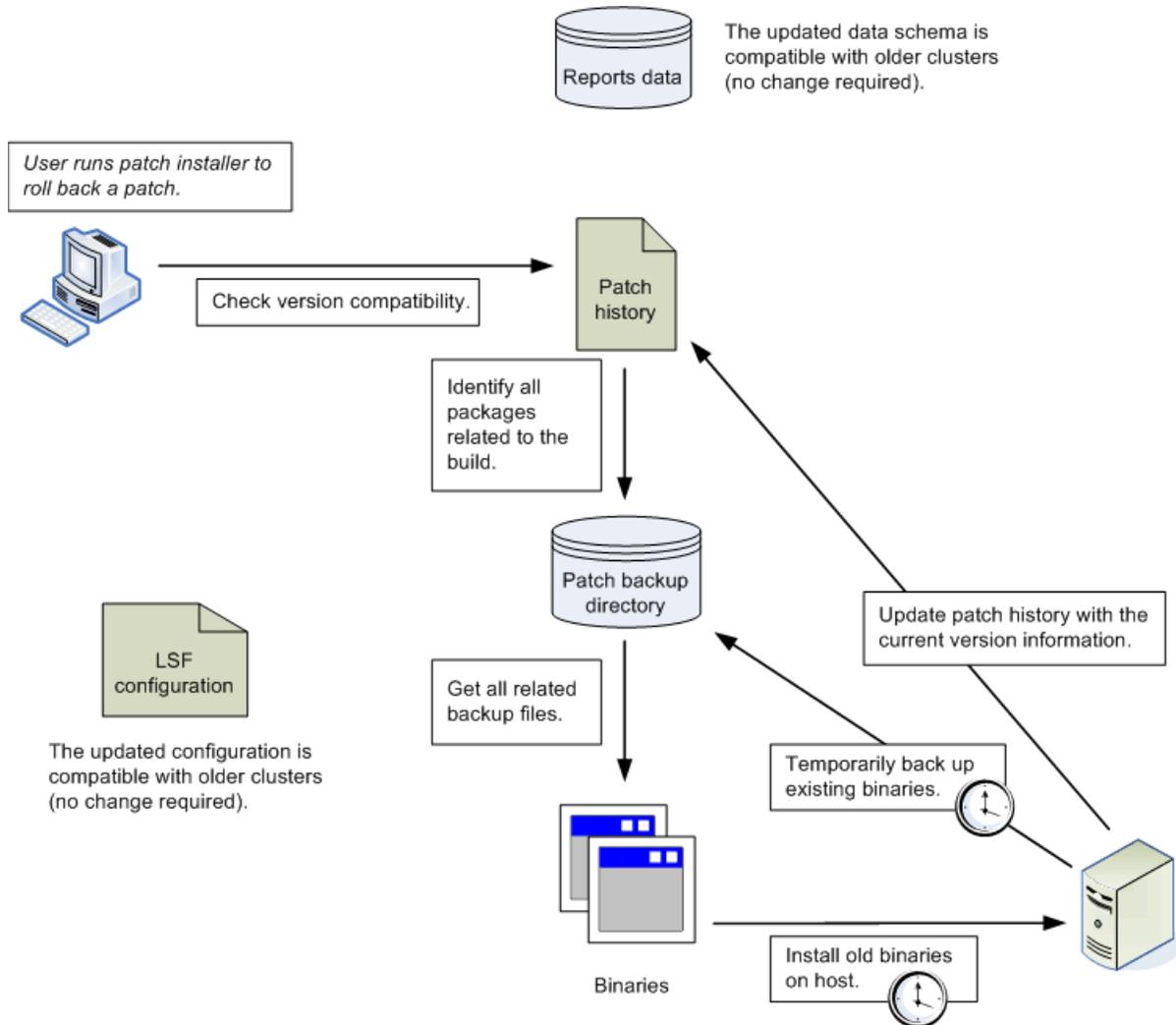
Patch installation interaction diagram

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 Management Console are licensed together, but delivered as separate distributions and patched separately.

Product version is a number identifying the release, such as LSF version 7.0.5. 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 may help 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.

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, LSF 7 Update 5 is version 7.0.5).
- ◆ *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

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.

The version command is not located with other LSF commands so it may not be in your path. The command location is `LSF_TOP/7.0/install/pversions`

Data schema update script

The update script for your data schema is a tool provided by Platform to update an existing database before you patch the cluster. The update script modifies the data schema so the database is prepared to handle data from an updated cluster.

Use the script that matches your database and cluster version. A patch may not involve any change to the database, or it may require multiple scripts to update different parts of the database.

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.

Patch history

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`.

Patch backups

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 will be 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 will 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.

Version management concepts

Multiple distributions

Like installation, patching the cluster sometimes requires you to download packages for each binary type or product component.

For example, to install an update, you may need to download multiple patches, such as distributions for LSF and distributions for the Platform Management Console.

Depending on the problem, a fix or fix pack may involve changes affecting just one binary type, or multiple distributions to patch multiple binary types.

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.

Installers

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 patch installer installs all patches and never modifies configuration. A partial distribution (FP or fix) can only be installed by the patch installer.

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/7.0/install/patchinstall`).

Version command accessibility

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/7.0/install/pversions`). It is not located with other LSF commands so it may not be in your path by default.

lsfinstall and install.config versions

The LSF installer may change with each update. You should not install a new update using the old `lsfinstall` program or `install.config` template. To properly update your cluster with new parameters and new configuration, make sure your installers match the version of the distribution you are installing.

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, you may use the `-f` option in the command line and specify a file that defines your environment. For more information, see the command reference.

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.

TIP: You can also install silently.

Cluster patching behavior table

When ...	Actions...	The result ...
Normal behavior.	The installer replaces current files with new.	◆ Success, cluster is updated.
Installing an update and the patch history is missing (files are not found in the directory defined by the parameter PATCH_HISTORY_DIR in patch.conf)	The installer creates new history files in the directory. The installer cannot determine compatibility but installs anyway because an update is a full distribution.	◆ 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 PATCH_HISTORY_DIR in patch.conf)	For a fix, the installer cannot determine compatibility.	◆ No update, cluster remains in same state ◆ Error presented on screen and logged in patch.log and patch.err
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.	◆ 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.	◆ 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.	◆ Overwriting a corrupt file will result in correct behavior. ◆ Overwriting a customized file will break 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 table

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 files

Logs

File	Description
<code>patch.log</code>	<p>This file:</p> <ul style="list-style-type: none"> ◆ Created by the patch installer (not created if you use <code>lsfinstall</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
<code>precheck.log</code>	<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
<code>install.log</code>	<p>This file:</p> <ul style="list-style-type: none"> ◆ Created by the LSF installer (not created if you use <code>patchinstall</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>lsfinstall</code>	This command: <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>patchinstall</code>	This command: <ul style="list-style-type: none"> ◆ Patches a cluster by installing binaries from a full or partial distribution (does not update configuration, so <code>lsfinstall</code> is recommended for an update release)
<code>patchinstall -r</code>	This command <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>pversions</code>	This command: <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>	This command: <ul style="list-style-type: none"> ◆ Displays detailed information for a specific file in the cluster (specify the installed file, for example <code>lim -V</code>)

Commands to check uninstalled packages

Command	Description
<code>pversions -c</code>	This command: <ul style="list-style-type: none"> ◆ Displays detailed information about the contents of an uninstalled package
<code>patchinstall -c</code>	This command: <ul style="list-style-type: none"> ◆ Tests if an uninstalled package is compatible with the cluster

Installing update releases on UNIX and Linux

To install an update release to the cluster.

IMPORTANT: For LSF 7 Update 3, you cannot use the steps in this section to update your cluster from LSF 7 Update 1 to Update 3. You *must* follow the steps in “*Migrating to LSF Version 7 Update 3 on UNIX and Linux*” to manually migrate your LSF 7 cluster to Update 3.

- 1 If you need to patch the reporting database, download the corresponding database update scripts and update the database schema first.
 - 2 Download and extract the new version of `lsfinstall`.
For example,

```
zcat lsf7Update5_lsfinstall.tar.Z | tar xvf -
```
 - 3 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.
 - 4 Download the patches. If hosts in your cluster have multiple binary types, you may require multiple distribution files to patch the entire cluster. Put the distribution files in the same directory as `lsfinstall`.
 - 5 Run the new LSF installer.
For example,

```
lsfinstall -f install.config
```


Specify the patches to install and let the installer finish.
 - 6 Restart the cluster.
This will make changes to daemons take effect.
 - 7 Optional. Run `pversions` to determine the state of the cluster.
 - 8 Optional. Free some space by deleting the contents of `backup` directories under EGO and LSF installation directories.
-

Installing fixes on UNIX and Linux

To install fixes or fix packs to update the cluster.

- 1 To patch the reporting database, download the corresponding database update scripts and update the database schema first.
- 2 Download the patches from Platform. If hosts in your cluster have multiple binary types, you may require multiple distribution files to patch the entire cluster.

Put the distribution files on any host.

For example,

```
//HostB/downloads/pkg1
```

```
//HostB/downloads/pkg2
```

- 3 Log on to a host in the cluster.
- 4 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)
```

- 5 Run the patch installer tool and specify the patches to install.

For example,

```
LSF_TOP/7.0/install/patchinstall //HostB/downloads/pkg1
```

```
//HostB/downloads/pkg2
```

Let the patch installer finish.

- 6 If you were prompted to do so, restart the cluster.
Patches that affect running daemons require you to restart manually.
- 7 Optional. Run `LSF_TOP/7.0/install/pversions` to determine the state of the cluster.
- 8 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.

Rolling back patches on UNIX and Linux

To remove patches that you installed using `patchinstall`, and return 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/7.0/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/7.0/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. The database schema is backwards compatible, so you do not need to change the reporting database.

Patching the Oracle database

Prerequisites: The Oracle database is properly configured and running:

- ◆ You have a user name, password, and URL to access the database server.
- ◆ You installed the latest JDBC driver (`ojdbc14.jar` or newer) for the Oracle database. This driver is available from the following URL:
http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html
- ◆ You are able to run `sqlplus`.

To patch the reporting database as part of patching the cluster, get the corresponding database update scripts and update the database schema first.

- 1 When you download the patches for your cluster, download the corresponding database update scripts from Platform.
- 2 In the command console, open the database schema directory.

```
cd LSF_TOP/perf/lsf/version/DBschema/Oracle
```
- 3 Run the scripts to create a database schema.

```
sqlplus user_name/password@connect_string @update_script
```

where
 - ◆ *user_name* is the user name on the database server
 - ◆ *password* is the password for this user name on the database server
 - ◆ *connect_string* is the named SQLNet connection for this database
 - ◆ *update_script* is the name of the patch script

Patching the Derby database

Prerequisites: The Derby database is properly configured and running:

To patch the reporting database as part of patching the cluster, get the corresponding database update scripts and update the database schema first.

1 When you download the patches for your cluster, download the corresponding database update scripts from Platform.

2 In the command console, open the database schema directory.

```
cd LSF_TOP/perf/lsf/version/DBschema/Derby
```

3 Start the `ij` tool.

```
ij.sh
```

4 Connect to the database.

```
connect
```

```
'jdbc:derby://host_name:port/db_name;user=user_name;password=password'
```

where

- ◆ *host_name* is the Derby database host
- ◆ *port* is the Derby database port, 1527 by default
- ◆ *db_name* is the database name, `app` by default
- ◆ *user_name* is the database user name, `app` by default
- ◆ *password* is the database login password, `app` by default

5 Run the scripts to create a database schema.

```
run 'update_script'
```

where

- ◆ *update_script* is the full path to the patch script

Patching the Derby database

Upgrading Platform LSF HPC

Contents

- ◆ [Upgrade Platform LSF HPC](#) on page 237
- ◆ [What lsinstall does](#) on page 237

Upgrade Platform LSF HPC

You can install Platform LSF HPC on UNIX or Linux hosts. Refer to the installation guide for more information.

When you run `lsinstall` for Platform LSF HPC, a number of changes are made for you automatically.

A number of shared resources are added to `lsf.shared` that are required by LSF HPC. When you upgrade Platform LSF HPC, you should add the appropriate resource names under the RESOURCES column of the Host section of `lsf.cluster.cluster_name`.

What lsinstall does

- ◆ Installs Platform LSF HPC binary and configuration files
- ◆ Installs the LSF HPC license file
- ◆ Automatically configures the following files:
 - ❖ `lsb.hosts`
 - ❖ `lsb.modules`
 - ❖ `lsb.resources`
 - ❖ `lsb.queues`
 - ❖ `lsf.cluster.cluster_name`
 - ❖ `lsf.conf`
 - ❖ `lsf.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    r1m    pg    ls    tmp    DISPATCH_WINDOW # Keywords
#hostA      () 3.5/4.5 15/   12/15 0      ()              # Example
default    !   ()    ()    ()    ()    ()              #
HPPA11     !   ()    ()    ()    ()    ()              #pset host
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_rms          ()                  ()
schmod_crayx1       ()                  ()
schmod_crayxt3      ()                  ()
End PluginModule
```

NOTE: The LSF 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 ReservationUsag
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 = Platform HPC 7 for IBM. 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 = Platform HPC 7 for IBM TotalView debug queue. 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 = Platform HPC 7 for linux.
End Queue
```

```
Begin Queue
QUEUE_NAME   = hpc_linux_tv
```

What lsinstall does

```
PRIORITY      = 30
NICE          = 20
# ...
TERMINATE_WHEN = LOAD PREEMPT WINDOW
RERUNNABLE   = NO
INTERACTIVE   = NO
DESCRIPTION   = Platform HPC 7 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.

- ◆ Configures `rms` queue for RMS jobs running in LSF HPC for LinuxQsNet.

Begin Queue

```
QUEUE_NAME    = rms
PJOB_LIMIT    = 1
PRIORITY      = 30
NICE          = 20
STACKLIMIT    = 5256
DEFAULT_EXTSCHED = RMS[RMS_SNODE] # LSF will using this scheduling policy if
                                   # -extsched is not defined.
# MANDATORY_EXTSCHED = RMS[RMS_SNODE] # LSF enforces this scheduling policy
RES_REQ       = select[rms==1]
DESCRIPTION   = Run RMS jobs only on hosts that have resource 'rms' defined
End Queue
```

TIP: 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 LSF HPC, 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.

TIP: LSF_IRIX_BESTCPUS is obsolete.

- ◆ On SGI IRIX and SGI Altix 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 LSF HPC in `lsf.shared`:

Begin Resource

RESOURCENAME	TYPE	INTERVAL	INCREASING	DESCRIPTION	# Keywords
rms	Boolean	()	()	(RMS)	
pset	Boolean	()	()	(PSET)	
slurm	Boolean	()	()	(SLURM)	
cpuset	Boolean	()	()	(CPUSET)	

What lsinstall does

mpich_gm	Boolean	()	()	(MPICH GM MPI)
lammpi	Boolean	()	()	(LAM MPI)
mpichp4	Boolean	()	()	(MPICH P4 MPI)
mvapich	Boolean	()	()	(Infiniband MPI)
sca_mpimon	Boolean	()	()	(SCALI MPI)
ibmmpi	Boolean	()	()	(IBM POE MPI)
hpmpi	Boolean	()	()	(HP MPI)
sgimpi	Boolean	()	()	(SGI MPI)
intelmpi	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)
hmmmer	Boolean	()	()	(hmmmer 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

TIP: You should add the appropriate resource names under the RESOURCES column of the Host section of `lsf.cluster.cluster_name`.

II

Working with Resources

- ◆ [Understanding Resources](#) on page 245
- ◆ [Adding Resources](#) on page 267
- ◆ [Managing Software Licenses with LSF](#) on page 277

Understanding Resources

Contents

- ◆ [About LSF Resources](#) on page 246
- ◆ [How Resources are Classified](#) on page 248
- ◆ [How LSF Uses Resources](#) on page 251
- ◆ [Load Indices](#) on page 253
- ◆ [Static Resources](#) on page 257
- ◆ [Automatic Detection of Hardware Reconfiguration](#) on page 264

About 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 available resources

View cluster resources (lsinfo)

- 1 Use `lsinfo` to list the resources available in your cluster.

The `lsinfo` command lists all the resource names and their 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 IO 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` to get 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 `lsload -s` command displays dynamic shared resources.

How Resources are Classified

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
solaris	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 will be 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 the LSF system will have the resources they use 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.

Viewing job resource usage

The `-l` option of the `bjobs` command displays the current resource usage of the job. The usage information is sampled by PIM every 30 seconds and collected by `sbatchd` at a maximum frequency of every `SBD_SLEEP_TIME` (configured in the `lsb.params` file) and sent to `mbatchd`. The update is done 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.

TIP: The parameter `LSF_PIM_LINUX_ENHANCE` in `lsf.conf` enables the enhanced PIM, returning the exact memory usage of processes using shared memory on Linux operating systems. By default memory shared between jobs may be counted more than once.

View load on a host

- 1 Run `bhosts -l` to check the load levels on the host, and adjust the suspending conditions of the host or queue if necessary.

The `bhosts -l` command gives the most recent load values used for the scheduling of jobs. 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
```

How LSF Uses Resources

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 (`elim`) executable. The `elim` 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>status</code>	host status	string			15 seconds
<code>r15s</code>	run queue length	processes	increasing	15 seconds	15 seconds
<code>r1m</code>	run queue length	processes	increasing	1 minute	15 seconds
<code>r15m</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>ls</code>	logins	users	increasing	N/A	30 seconds
<code>it</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>tmp</code>	available space in temporary file system	MB	decreasing	N/A	120 seconds
<code>io</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 `status` index is a string indicating the current status of the host. This status applies to the LIM and RES.

The possible values for `status` are:

Status	Description
ok	The host is available to accept remote jobs. The LIM can select the host for remote execution.
-ok	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.
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 <code>lshosts</code> 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 `available` is frequently used in command output titles and headings. `Available` means a host is in any state except `unavail`. This means an `available` 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 `lsload -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 will run a CPU-intensive job the fastest.

Use `lsload -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 will be high. Paging rate is a good measure of how a machine will respond 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.

Viewing 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`, `nthreads`, `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	<code>nice(2)</code> 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>nthreads</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.shared`. 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.shared`. 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 physical processors 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 only the number of physical processors (`EGO_DEFINE_NCPUS=procs`).

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 processors, however `lshosts` output will continue 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

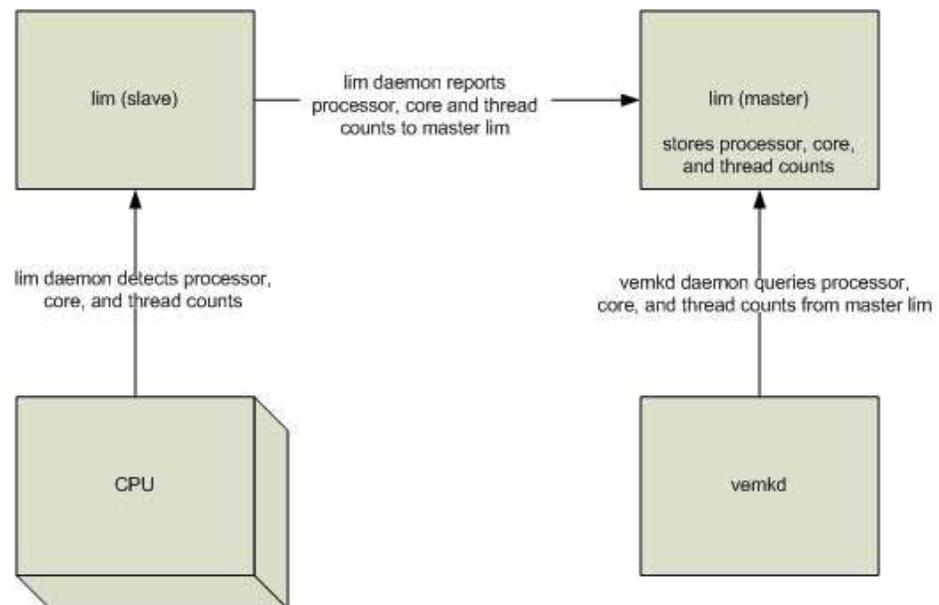
Traditionally, the value of `ncpus` has been equal to the number of physical CPUs. However, many CPUs consist of multiple cores and threads, so the traditional 1:1 mapping is no longer useful. A more useful approach is to set `ncpus` to equal one of the following:

- ◆ The number of processors (this is the `ncpus` default setting)
- ◆ Cores—the number of cores (per processor) * the number of processors
- ◆ 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`).

See [Define ncpus—processors, cores, or threads](#) on page 260 for details.

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. See [Override the global configuration of ncpus computation](#) on page 261 for details.

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 will 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 $\text{ncpus} = \text{procs} * \text{cores} * \text{threads}$)

By default, `ncpus` is set to `procs` (number of processors).

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.

Override the global configuration of `ncpus` computation

The cluster administrator globally defines how the `ncpus` resource is computed. The `ncpus` global definition can be overridden on specified dynamic and static hosts in the cluster.

Defining computation of `ncpus` on dynamic hosts

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.

Defining computation of ncpus on static hosts

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
#lemon    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.

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.

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 will be automatically recognized by LSF. When it detects a change in the number of processors, the local LIM also checks `maxmem`, `maxswp`, `maxtmp`.

Viewing 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 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`, remove the comment from 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.

Set the external static LIM

Adding Resources

Contents

- ◆ [About Configured Resources](#) on page 268
- ◆ [Add New Resources to Your Cluster](#) on page 269
- ◆ [Static Shared Resource Reservation](#) on page 275
- ◆ [External Load Indices](#) on page 276
- ◆ [Modifying a Built-In Load Index](#) on page 276

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 will be displayed to a user by `lsinfo`.
See [Configuring lsf.shared Resource Section](#) on page 270.
- 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`.
See [Configuring lsf.cluster.cluster_name Host Section](#) on page 272.
- 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).
See [Configuring lsf.cluster.cluster_name ResourceMap Section](#) on page 273.
- 5 Reconfigure your cluster.

Configuring lsf.shared Resource Section

Configured resources are defined in the `Resource` section of `lsf.shared`. There is no distinction between shared and non-shared resources.

You must specify at least a name and description for the resource, using the keywords `RESOURCENAME` and `DESCRIPTION`.

- ◆ A resource name cannot begin with a number.
- ◆ A resource name cannot contain any of the following characters:
: . () [+ - * / ! & | < > @ =
- ◆ A resource name 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
- ◆ To avoid conflict with `inf` and `nan` keywords in 3rd-party libraries, resource names should not begin with `inf` or `nan` (upper case or lower case). Resource requirement strings, such as `-R "infra"` or `-R "nano"` will cause an error. Use `-R "defined(infxx)"` or `-R "defined(nanxx)"`, to specify these resource names.
- ◆ Resource names are case sensitive
- ◆ Resource names can be up to 39 characters in length

You can also specify:

- ◆ The resource type (`TYPE = Boolean | String | Numeric`). The default is `Boolean`.
- ◆ For dynamic resources, the update interval (`INTERVAL`, in seconds)
- ◆ For numeric resources, where a higher value indicates greater load (`INCREASING = Y`)
- ◆ For numeric shared resources, where LSF releases the resource when a job using the resource is suspended (`RELEASE = Y`)

When the optional attributes are not specified, the resource is treated as static and `Boolean`.

Defining consumable resources

Specify resources as consumable in the `CONSUMABLE` column of the `RESOURCE` section of `lsf.shared` to explicitly control if a resource is consumable. Static and dynamic numeric resources can be specified as consumable. `CONSUMABLE` is optional. The defaults for the consumable attribute are:

- ◆ Built-in indicies:
 - ❖ The following are consumable: `r15s, r1m, r15m, ut, pg, io, ls, it, tmp, swp, mem`.
 - ❖ All other built-in static resources are not consumable. (e.g., `ncpus, ndisks, maxmem, maxswp, maxtmp, cpuf, type, model, status, rexpri, server, hname`).
- ◆ External shared resources:

- ❖ All numeric resources are consumable.
- ❖ String and boolean resources are not consumable.

You should only specify consumable resources in the `rusage` section of a resource requirement string. Non-consumable resources are ignored in `rusage` sections.

A non-consumable resource should not be releasable. Non-consumable numeric resource should be able to be used in `order`, `select` and `same` sections of a resource requirement string.

When `LSF_STRICT_RESREQ=Y` in `lsf.conf`, LSF rejects resource requirement strings where an `rusage` section contains a non-consumable resource.

Viewing consumable resources

Use `lsfinfo -l` to view consumable resources. For example:

```
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
```

Example

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

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	(IndividualCPUSpeed)

Configuring `lsf.cluster.cluster_name` Host Section

```
bandwidth    Numeric 60    Y          (IndividualNetworkBandwidth)
End Resource
```

Configuring `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.

Define the resource names as strings in the Resource section of `lsf.shared`. You may list any number of resources, enclosed in parentheses and separated by blanks or tabs.

If you need to define shared resources across hosts, you must use the ResourceMap section.

String resources cannot contain spaces. Static numeric and string resources use following syntax:

```
resource_name=resource_value
```

For dynamic numeric and string resources, use *resource_name* directly.

If resources are defined in both the resource column of the Host section and the ResourceMap section, the definition in the resource column takes affect.

Example

```
Begin Host
HOSTNAME  model type server r1m 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
```

Configuring `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`.

For each resource, you must specify the name and the hosts that have it.

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.

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

If you plan to submit jobs using JSDL files, you must uncomment the following lines:

```
RESOURCENAME  LOCATION
osname        [default]
osver         [default]
cpuarch       [default]
cpuspeed      [default]
bandwidth     [default]
```

RESOURCENAME

The name of the resource, as defined in `lsf.shared`.

LOCATION

Defines 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.

Possible states of a resource:

- ◆ Each host in the cluster has the resource
- ◆ The resource is shared by all hosts in the cluster
- ◆ There are multiple instances of a resource within the cluster, and each instance is shared by a unique subset of hosts.

Syntax

```
([resource_value@][host_name... | all [~host_name]... | others | default] ...)
```

- ◆ For static resources, you must include the resource value, which indicates the quantity of the resource. Do not specify the resource value for dynamic resources because information about dynamic resources is updated by ELIM.
- ◆ Type square brackets around the list of hosts, as shown. You can omit the parenthesis if you only specify one set of hosts.
- ◆ Each set of hosts within square brackets specifies an instance of the resource. The same host cannot be in more than one instance of a resource. 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.

Non-batch configuration

The following items should be taken into consideration when configuring resources.

- ◆ In `lsf.cluster.cluster_name`, the `Host` section must precede the `ResourceMap` section, since the `ResourceMap` section uses the host names defined in the `Host` section.
- ◆ Use the `RESOURCES` column in the `Host` section of the `lsf.cluster.cluster_name` file to associate static Boolean resources with particular hosts.
- ◆ 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`.

Static Shared Resource Reservation

You must use resource reservation to prevent over-committing static shared resources when scheduling.

The usual situation is that you configure single-user application licenses as static shared resources, and make that resource one of the job requirements. You should also reserve the resource for the duration of the job. Otherwise, LSF updates resource information, assumes that all the static shared resources can be used, and places another job that requires that license. The additional job cannot actually run if the license is already taken by a running job.

If every job that requests a license and also reserves it, LSF updates the number of licenses at the start of each new dispatch turn, subtracts the number of licenses that are reserved, and only dispatches additional jobs if there are licenses available that are not already in use.

Reserving a static shared resource

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 that will 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 will be 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 the *Platform LSF Configuration Reference*.

Modifying 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 the *Platform LSF Configuration Reference*.

Managing Software Licenses with LSF

Software licenses are valuable resources that must be fully utilized. This section discusses how LSF can help manage licensed applications to maximize utilization and minimize job failure due to license problems.

Contents

- ◆ [Using Licensed Software with LSF](#) on page 277
- ◆ [Host-locked Licenses](#) on page 277
- ◆ [Counted Host-Locked Licenses](#) on page 277
- ◆ [Network Floating Licenses](#) on page 278

Using 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.

Configuring host-locked licenses

You can 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 [Boolean resources](#) on page 248 for information about configuring Boolean resources.

See the *Platform 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.

Configuring counted host-locked licenses

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_licenses` shell script

Using an External LIM (ELIM)

To use an external LIM (ELIM) to get the number of licenses currently available, configure an external load index `licenses` giving the number of free licenses on each host. To restrict the application to run only on hosts with available licenses, specify `licenses>=1` in the resource requirements for the application.

See [External Load Indices](#) on page 276 for instructions on writing and using an ELIM and configuring resource requirements for an application.

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

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. See [Configuring Pre- and Post-Execution Commands](#) on page 618 for information about configuring queue-level pre-execution commands.

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 will be requeued if they exit with values indicating that the license was not successfully obtained.

See [Automatic Job Requeue](#) on page 505 for more information.

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, they belong to all hosts in the cluster.

LSF can be used to manage floating licenses using the following LSF features:

- ◆ Shared resources
- ◆ Resource reservation
- ◆ Job queuing

Using LSF to run licensed software can improve the utilization of the licenses. 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. Floating licenses also increase 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 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 Verilog 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
```

Submitting jobs

The users would 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 Verilog 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
verilog         ([all])
End ResourceMap
```

The `INTERVAL` in the `lsf.shared` file indicates how often the ELIM is expected to update the value of the `Verilog` 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 `Verilog` 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`.

Submitting jobs

The users would submit jobs requiring `verilog` licenses as follows:

```
bsub -R "rusage[verilog=1:duration=1]" myprog
```

Configuring a dedicated queue for floating licenses

Whether you run all license jobs through LSF or run jobs that use licenses that are outside of LSF control, you can configure a dedicated queue to run jobs requiring a floating software license.

For each job in the queue, LSF reserves a software license before dispatching a job, and releases the license when the job finishes.

Use the `bhosts -s` command to display the number of licenses being reserved by the dedicated queue.

Example

The following example defines a queue named `q_verilog` in `lsb.queues` dedicated to jobs that require `Verilog` licenses:

```
Begin Queue
QUEUE_NAME = q_verilog
RES_REQ=rusage[verilog=1:duration=1]
End Queue
```

The queue named `q_verilog` contains jobs that will reserve one `Verilog` license when it is started.

If the `Verilog` licenses are not cluster-wide, but can only be used by some hosts in the cluster, the resource requirement string should include the `defined()` tag in the `select` section:

```
select[defined(verilog)] rusage[verilog=1]
```

Preventing underutilization of licenses

One limitation to using a dedicated queue for licensed jobs is that if a job does not actually use the license, then the licenses will be under-utilized. This could happen if the user mistakenly specifies that their application needs a license, or submits a non-licensed job to a dedicated queue.

LSF assumes that each job indicating that it requires a `Verilog` license will actually use it, and simply subtracts the total number of jobs requesting `Verilog` licenses from the total number available to decide whether an additional job can be dispatched.

Use the `duration` keyword in the queue resource requirement specification to release the shared resource after the specified number of minutes expires. This prevents multiple jobs started in a short interval from over-using the available licenses. 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, configure job requeue by using the `REQUEUE_EXIT_VALUES` parameter in a queue definition in `lsb.queues`. If a job exits with one of the values in the `REQUEUE_EXIT_VALUES`, LSF will requeue the job.

Example

Jobs submitted to the following queue will use Verilog 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 causes the job to be requeued and LSF will attempt to reschedule it at a later time.

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
```

For more information

- ◆ See [Automatic Job Requeue](#) on page 505 for more information about configuring job requeue
- ◆ See Chapter 39, “[Job Starters](#)” for more information about LSF job starters

III

Job Scheduling Policies

- ◆ [Time Syntax and Configuration](#) on page 285
- ◆ [Deadline Constraint and Exclusive Scheduling](#) on page 291
- ◆ [Preemptive Scheduling](#) on page 295
- ◆ [Specifying Resource Requirements](#) on page 297
- ◆ [Fairshare Scheduling](#) on page 335
- ◆ [Goal-Oriented SLA-Driven Scheduling](#) on page 383
- ◆ [Working with Application Profiles](#) on page 401

Time Syntax and Configuration

Contents

- ◆ [Specifying Time Values](#) on page 285
- ◆ [Specifying Time Windows](#) on page 285
- ◆ [Specifying Time Expressions](#) on page 286
- ◆ [Using Automatic Time-based Configuration](#) on page 287

Specifying 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 7, representing the day of the week, where 0 represents every day, 1 represents Monday, and 7 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.

Specifying 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]
```

Specifying Time Expressions

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
```

Specifying 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)
```

Using 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 `badadmin reconfig` command.

The expressions are evaluated by LSF every 10 minutes based on `mbatchd` 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 `mbatchd`, providing continuous system availability.

In the following examples, the `#if`, `#else`, `#endif` 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
```

Using Automatic Time-based Configuration

```
# 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

```

Creating 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.

For syntax of a time expression, see [Specifying Time Expressions](#) on page 286.

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 Use the following LSF commands to verify configuration:

❖ `bhosts`

Using Automatic Time-based Configuration

- ❖ `bladmin ckconfig`
 - ❖ `blimits -c`
 - ❖ `blinfo`
 - ❖ `blstat`
 - ❖ `bparams`
 - ❖ `bqueues`
 - ❖ `bresources`
 - ❖ `busers`
-

Deadline Constraint and Exclusive Scheduling

Contents

- ◆ [Using Deadline Constraint Scheduling](#) on page 291
- ◆ [Using Exclusive Scheduling](#) on page 292

Using 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.

Disabling deadline constraint scheduling

Deadline constraint scheduling is enabled by default. To disable it 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
```

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.

Using 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

- 1 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

- 1 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

- 1 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

- 1 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[exc1]" my_job`

Preemptive Scheduling

Contents

- ◆ [About Preemptive Scheduling](#) on page 295

About Preemptive Scheduling

Preemptive scheduling lets a pending high-priority job take job slots away from a running job of lower priority. When two jobs compete for the same job slots, LSF automatically suspends the low-priority job to make slots available to the high-priority job. The low-priority job 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.

For detailed information about the preemptive scheduling feature and how to configure it, see the *Platform LSF Configuration Guide*.

Limitation

By default, the following types of jobs cannot be preempted:

- ◆ Jobs that have been forced to run with the command `brun`
- ◆ Backfill and exclusive jobs, including compute unit exclusive jobs

By default exclusive jobs, including compute unit exclusive jobs, cannot preempt other jobs.

Preemptive and preemptable queues

Preemptive queues Jobs in a preemptive queue can preempt jobs in any queue of lower priority, even if the low-priority queues are not specified as preemptable.

Preemptable queues Jobs in a preemptable queue can be preempted by jobs from any queue of a higher priority, even if the high-priority queues are not specified as preemptive.

Preemptive and preemptable jobs

Preemptive jobs Preemptive jobs are pending in a high-priority queue and require the specified job slots. Their queue must be able to preempt the low-priority queue.

Preemptable jobs Preemptable jobs are running in a low-priority queue and are holding the specified job slot. Their queue must be able to be preempted by the high-priority queue.

Specifying Resource Requirements

Contents

- ◆ [About Resource Requirements](#) on page 298
- ◆ [Queue-level Resource Requirements](#) on page 300
- ◆ [Job-level Resource Requirements](#) on page 302
- ◆ [About Resource Requirement Strings](#) on page 304
- ◆ [Selection String](#) on page 310
- ◆ [Order String](#) on page 318
- ◆ [Usage String](#) on page 320
- ◆ [Span String](#) on page 327
- ◆ [Same String](#) on page 329
- ◆ [Compute Unit String](#) on page 331

About Resource Requirements

Resource requirements define which hosts a job can run on. Each job has its resource requirements. 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.

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 Chapter 23, “[Working with Application Profiles](#)” 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 `badadmin reconfig`. 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`.

Examples

```
RES_REQ=select[((type==LINUX2.4 && r1m < 2.0)|| (type==AIX && r1m < 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.

See Chapter 37, “[Load Thresholds](#)” for information about suspending conditions and configuring load thresholds.

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_rsrc=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.

TIP: With `esub`, you must use the `&&` operator to specify multiple resource requirements. The `LSB_SUB_RES_REQ` variable in `esub` does not support the use of multiple `-R` sections.

Examples

```
bsub -R "swp > 15 && hpux order[ut]" myjob
```

or

```
bsub -R "select[swp > 15]" -R "select[hpux] 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[hpux] 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: Submitted from host <hostA>, CWD <${HOME}>,
Requested Resources <type==any>;
```

```
Fri May 10 17:21:31: Started on <hostB>, Execution Home
</home/user1>, Execution CWD </home/user1>;
```

```
Fri May 10 17:21:47: 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: Submitted from host <hostA>, to Queue
<normal>, CWD
    <${HOME}>, Requested Resources <type==any>;
Fri May 10 17:21:31: Dispatched to <hostB>;
Fri May 10 17:21:32: Starting (Pid 1850232);
Fri May 10 17:21:33: Running with execution home </home/user1>,
Execution
    CWD </home/user1>, Execution Pid <1850232>;
Fri May 10 17:21:45: 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, `bjobs -l` and `bhist -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 `lslload` command displays the load levels for all hosts that have the requested resources.

Specifying resource requirements for the `lslrun` 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
- ◆ `lslload` selects and orders hosts
- ◆ `lslplace` uses the information in `select`, `order`, and `rusage` sections to select an appropriate host for a task
- ◆ `lslloadadj` 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

$num1*\{simple_string1\} + num2*\{simple_string2\} + \dots$

where num_x is the number of slots affected and $simple_string_x$ 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 num_x 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) = $\text{MAX}(0, (\text{default number of job slots from PROCLIMIT}) - (num1 + num2 + \dots))$

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 num_x can be omitted:

- ◆ $num_slots = (num1 + num2 + num3 + \dots)$

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:

- ◆ $num1*\{simple_string1\} + num2*\{simple_string2\} + num3*\{simple_string3\}$

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
<code>select</code>	all levels satisfied
<code>same</code>	all levels combined
<code>order</code> <code>span</code> <code>cu</code>	job-level section overwrites application-level section, which overwrites queue-level section (if a given level is present)
<code>rusage</code>	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 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.

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 $num1*\{rusage[R1]\} + num2*\{rusage[R2]\}$ and the queue-level requirement is $rusage[RQ]$ where RQ is a job-based resource, the merged requirement is $num1*\{rusage[merge(R1,RQ)]\} + num2*\{rusage[R2]\}$</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: $2\{\text{select}[\text{type}==\text{X86}_64] \text{rusage}[\text{licA}=1] \text{span}[\text{hosts}=1]\} + 8\{\text{select}[\text{type}==\text{any}]\}$

application level: not defined

queue level: $\text{rusage}[\text{perslot}=1]$

The final job scheduling resource requirement merges the simple queue-level rusage section into each term of the compound job-level requirement, resulting in: $2\{\text{select}[\text{type}==\text{X86}_64] \text{rusage}[\text{licA}=1:\text{perslot}=1] \text{span}[\text{hosts}=1]\} + 8\{\text{select}[\text{type}==\text{any}] \text{rusage}[\text{perslot}=1]\}$

Example 2

A compound job requirement and compound queue requirement.

job level: $2\{\text{select}[\text{type}==\text{X86}_64 \ \&\& \ \text{tmp}>10000] \text{rusage}[\text{mem}=1000] \text{span}[\text{hosts}=1]\} + 8\{\text{select}[\text{type}==\text{X86}_64]\}$

application level: not defined

queue level: $2\{\text{select}[\text{type}==\text{X86}_64] \text{rusage}[\text{mem}=1000] \text{span}[\text{hosts}=1]\} + 8\{\text{select}[\text{type}==\text{X86}_64]\}$

The final job scheduling resource requirement ignores the compound queue-level requirement, resulting in: $2\{\text{select}[\text{type}==\text{X86}_64 \ \&\& \ \text{tmp}>10000] \text{rusage}[\text{mem}=1000] \text{span}[\text{hosts}=1]\} + 8\{\text{select}[\text{type}==\text{X86}_64]\}$

Example 3

A compound job requirement and simple queue requirement where the queue requirement is a job-based resource.

job level: $2\{\text{select}[\text{type}==\text{X86}_64]\} + 2\{\text{select}[\text{mem}>1000]\}$

application level: not defined

queue level: $\text{rusage}[\text{licA}=1]$ 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]}`

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 `r1m` respectively.

For `ut`, specify the percentage CPU utilization as an integer between 0-100.

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`, `lsp1ace`, `lstrun`, and `lsg1run` the default is `type==local` unless a string or Boolean resource is specified, in which case it is `type==any`. For `lshosts`, `lslload`, `lsm1on` and `lsllogin` the default is `type==any`.

TIP: When `PARALLEL_SCHED_BY_SLOT=Y` in `l1sb.params`, the resource requirement string keyword `ncpus` refers to the number of slots instead of the number of processors, however `lshosts` output will continue to show `ncpus` as defined by `EGO_DEFINE_NCPUS` in `l1sf.conf`.

Specifying 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[hpux]"
```

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.

Selecting shared string resources

You must use single quote characters (`'`) around string-type shared resources. For example, use `lsload -s` to see the shared resources defined for the cluster:

```
lsload -s
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 `lsload -R` to specify the shared resources you want to view, enclosing the shared resource values in single quotes. For example:

```
lsload -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 `lsload`.

Operators

These operators can be used in selection strings. The operators are listed in order of decreasing precedence.

Syntax	Meaning
(a)	When <code>LSF_STRICT_RESREQ=Y</code> is configured in <code>lsf.conf</code> , an expression between parentheses has higher priority than other operators.
-a	Negative of a
!a	Logical not: 1 if <code>a==0</code> , 0 otherwise
a * b	Multiply a and b
a / b	Divide a by b

Syntax	Meaning
<code>a + b</code>	Add a and b
<code>a - b</code>	Subtract b from a
<code>a > b</code>	1 if a is greater than b, 0 otherwise
<code>a < b</code>	1 if a is less than b, 0 otherwise
<code>a >= b</code>	1 if a is greater than or equal to b, 0 otherwise
<code>a <= b</code>	1 if a is less than or equal to b, 0 otherwise
<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)]
```

Specifying 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 builtin 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 builtin 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-builtin resource names not configured in `lsf.shared`.

Specifying exclusive resources

An exclusive resource may be used in the resource requirement string of any placement or scheduling command, such as `bsub`, `lspplace`, `lsrun`, or `lsgrun`. 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.shared`. 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`, `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.

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>
```

Selection String

```
<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`, `hpux`, `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:

```
defined( 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]
select[type==any] rusage[mem=1024]
select[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:

```
select[mem>0] select[maxmem>0]
```

Escape characters (like '\n') are not supported in string literals.

A colon (:) is not allowed inside the select string. For example,

```
select[mg:bigmem]
```

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 `lsb.queues`, `RES_REQ='swp>100'` and `RES_REQ="swp>100"` are correct. Neither `RES_REQ=select['swp>100']` nor `RES_REQ=select["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 "select['swp>100']"`
- ◆ `bsub -R 'select["swp>100"]'`

Some incorrect resource requirements are no longer silently ignored. For example, when `LSF_STRICT_RESREQ=Y` is configured in `lsf.conf`, the following are rejected by the resource requirement parser:

- ◆ `microcs73` is rejected:
`linux rusage[mem=16000] microcs73`
- ◆ `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[selection_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.

Selection String

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
```

Checking 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.

Specifying 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 `lslogin(1):ls:r1m`).

Example

```
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

Batch jobs

The resource usage (`rusage`) section can be specified at the job level, with the queue configuration parameter `RES_REQ`, or the application profile parameter `RES_REQ`.

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]]
```

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:
 - ❖ `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. 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`, `ls`, `it`, `tmp`, `swp`, and `mem`), the job-level value cannot be larger than 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.

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:

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.

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 Chapter 23, “[Working with Application Profiles](#)” 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.

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*{[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.

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.

Specifying multiple -R options

`bsub` accepts multiple `-R` options for the `rusage` 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 "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.

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
```

Specifying 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
```

Usage String

- ◆ 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
```

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
<code>[mem=200:duration=180]</code>	<code>[w1=1 w2=1 w3=1]"</code>	<code>[w1=1, mem=200:duration=180 w2=1, mem=200:duration=180 w3=1, mem=200:duration=180]</code>
<code>[w1=1 w2=1 w3=1]</code>	<code>[mem=200:duration=180]"</code>	<code>[mem=200:duration=180, w1=1 mem=200:duration=180, w2=1 mem=200:duration=180, w3=1]</code>

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 `lshrun`, 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

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
```

Usage String

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.

See [Controlling Processor Allocation Across Hosts](#) on page 554 for more information about specifying `span` strings.

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`.

Specifying 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 Chapter 23, “[Working with Application Profiles](#)” 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.

For a complete description of compute units see [Controlling Job Locality using Compute Units](#) on page 547 in Chapter 34, “Running Parallel Jobs”.

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 `lsf.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 `lsf.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 `lsf.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 `lsf.hosts`.

Free slots include all available slots not occupied by running jobs.

When `pref` is used with the keyword `balance`, `balance` takes precedence.

- 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.

`usablecuslots` and `balance` 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.

`balance` and `usablecuslots` cannot be used together.

Compute Unit String

When `balance` and `pref` are both used, `balance` 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 `balance` 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 Chapter 23, “[Working with Application Profiles](#)” for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

Compute Unit String

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.

To get ideas about how to use fairshare scheduling to do different things, see [Ways to Configure Fairshare](#) on page 364.

Contents

- ◆ Basic Concepts
 - ❖ [Understanding Fairshare Scheduling](#) on page 336
 - ❖ [User Share Assignments](#) on page 337
 - ❖ [Dynamic User Priority](#) on page 338
 - ❖ [How Fairshare Affects Job Dispatch Order](#) on page 340
- ◆ User-based Fairshare
 - ❖ [Host Partition User-based Fairshare](#) on page 341
 - ❖ [Queue-level User-based Fairshare](#) on page 343
 - ❖ [Cross-queue User-based Fairshare](#) on page 343
 - ❖ [Hierarchical User-based Fairshare](#) on page 347
- ◆ Queue-based Fairshare
 - ❖ [Queue-based Fairshare](#) on page 350
 - ❖ [Configuring Slot Allocation per Queue](#) on page 352
 - ❖ [View Queue-based Fairshare Allocations](#) on page 354
- ◆ Advanced Topics

- ❖ [Using Historical and Committed Run Time](#) on page 360
- ❖ [Users Affected by Multiple Fairshare Policies](#) on page 363
- ❖ [Ways to Configure Fairshare](#) on page 364
- ❖ [Resizable jobs and fairshare](#) on page 367

Understanding 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 User1 and User2.
- ◆ If there are 12 users in total, then 10 users compete for those 8 shares, and each of them is significantly less important than User1 and User2.

```
[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 User1, 6 to User2, and 4 to the remaining user. User1 has half of the available resources (10 shares out of 20).
- ◆ If there are 12 users in total, assigns 56 shares: 10 to User1, 6 to User2, and 4 to each of the remaining 10 users. User1 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 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:

- ◆ 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 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:

$$\text{dynamic priority} = \frac{\text{number_shares}}{(\text{cpu_time} * \text{CPU_TIME_FACTOR} + \text{run_time} * \text{RUN_TIME_FACTOR} + (1 + \text{job_slots}) * \text{RUN_JOB_FACTOR} + \text{fairshare_adjustment} * \text{FAIRSHARE_ADJUSTMENT_FACTOR})}$$

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.

<i>number_shares</i>	The number of shares assigned to the user.
<i>cpu_time</i>	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 <code>lsb.params</code> (5 hours by default).
<i>run_time</i>	The total run time of running jobs (measured in hours).
<i>job_slots</i>	The number of job slots reserved and in use.
<i>fairshare_adjustment</i>	The adjustment calculated by the fairshare adjustment plugin (<code>libfairshareadjust.*</code>).

Configuring the default dynamic priority

You can give additional weight to the various factors in the priority calculation by setting the following parameters in `lsb.params`.

- ◆ 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.
------------------------	--------------------------------

How Fairshare Affects Job Dispatch Order

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

Customizing 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.

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:

$$\text{fairshare adjustment} = (1 + \text{slots}) * ((\text{total_memory} / \text{slots}) / (\text{slot_memory} * \text{THRESHOLD}))$$

slots The number of job slots in use by started jobs.

total_memory The total memory in use by started jobs.

slot_memory The average memory allocated per slot.

THRESHOLD The memory threshold set in the fairshare adjustment plugin.

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. For more information, see [Cross-queue User-based Fairshare](#) on page 343.

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 `lsb.queues`) and must have a share assignment (`FAIRSHARE` in `lsb.queues`). Even cluster and queue administrators cannot submit jobs to a fairshare queue if they do not have a share assignment.

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 `lsb.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.

Applying 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 `lsb.queues`) and must have a share assignment (`FAIRSHARE` in `lsb.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  -   -   -   -   -   -   -   -   -   -   -

loadSched      cpuspeed  bandwidth
                -           -
    
```

```

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
...

```

Configuring cross-queue fairshare

Considerations

- ◆ 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.`

Configure cross-queue fairshare

- ◆ Cross-queue fairshare cannot be used with host partition fairshare. It is part of queue-level fairshare.

- 1 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
```

Controlling 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
```

Configuring 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.

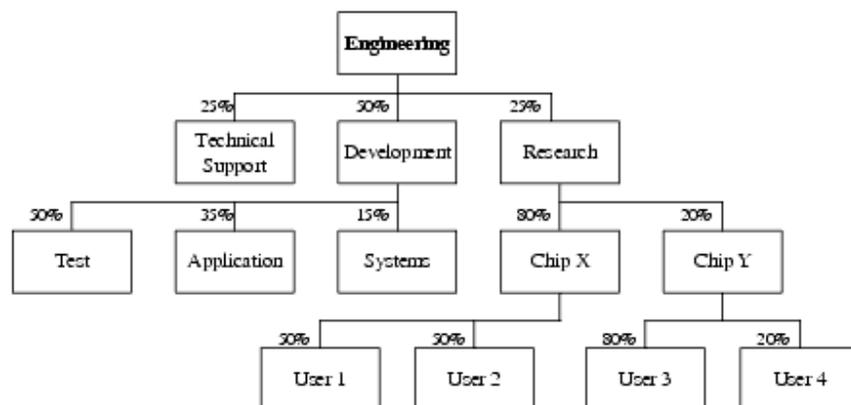
Configure a share tree

- 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.
- 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, `User1` is entitled to 10% of the available resources ($.50 \times .80 \times .25 = .10$) and `User3` 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 `ChipY` has a higher dynamic priority than `ChipX`, the next comparison is between `User3` and `User4`, so the importance of `User1` is not relevant. The dynamic priority of `User1` 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.

Managing 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.

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.

Configuring Slot Allocation per Queue

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
```

Configuring Slot Allocation per Queue

Configure as many pools as you need in `lsb.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.

Host job slot limit

The hosts used by the pool must have a maximum job slot limit, configured either in `lsb.hosts (MXJ)` or `lsb.resources (HOSTS and SLOTS)`.

Configure slot allocation per queue

-
- 1 For each queue that uses queue-based fairshare, define the following in `lsb.queues`:
 - a `SLOT_SHARE`
 - b `SLOT_POOL`
 - 2 Optional: Define the following in `lsb.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 For each host used by the pool, define a maximum job slot limit, either in `lsb.hosts (MXJ)` or `lsb.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  = poolA
SLOT_SHARE = 50
HOSTS      = groupA
...
End Queue

Begin Queue
QUEUE_NAME = queue2
PRIORITY   = 48
SLOT_POOL  = poolA
SLOT_SHARE = 30
HOSTS      = groupA
...
End Queue

Begin Queue
QUEUE_NAME = queue3
PRIORITY   = 46
SLOT_POOL  = poolA
SLOT_SHARE = 20
HOSTS      = groupA
...
End Queue

```

The following configures a pool named poolB, with three queues with equal shares, using the hosts in host group groupB:

```

Begin Queue
QUEUE_NAME = queue4
PRIORITY   = 44
SLOT_POOL  = poolB
SLOT_SHARE = 30
HOSTS      = groupB
...
End Queue

Begin Queue
QUEUE_NAME = queue5
PRIORITY   = 43
SLOT_POOL  = poolB
SLOT_SHARE = 30
HOSTS      = groupB
...
End Queue

Begin Queue
QUEUE_NAME = queue6
PRIORITY   = 42
SLOT_POOL  = poolB
SLOT_SHARE = 30
HOSTS      = groupB
...
End Queue

```

View Queue-based Fairshare Allocations

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

      cpuspeed  bandwidth
loadSched      -   -
loadStop       -   -

USERS:  all users
HOSTS:  groupA/
SLOT_SHARE: 50%
SLOT_POOL: poolA

```

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 poolA contains queue1, queue2, and queue3. poolB contains queue4, queue5, and queue6. The `bqueues` command shows the number of running jobs in each queue:

```
bqueues
QUEUE_NAME  PRIO 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  PRIO 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 3. 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:

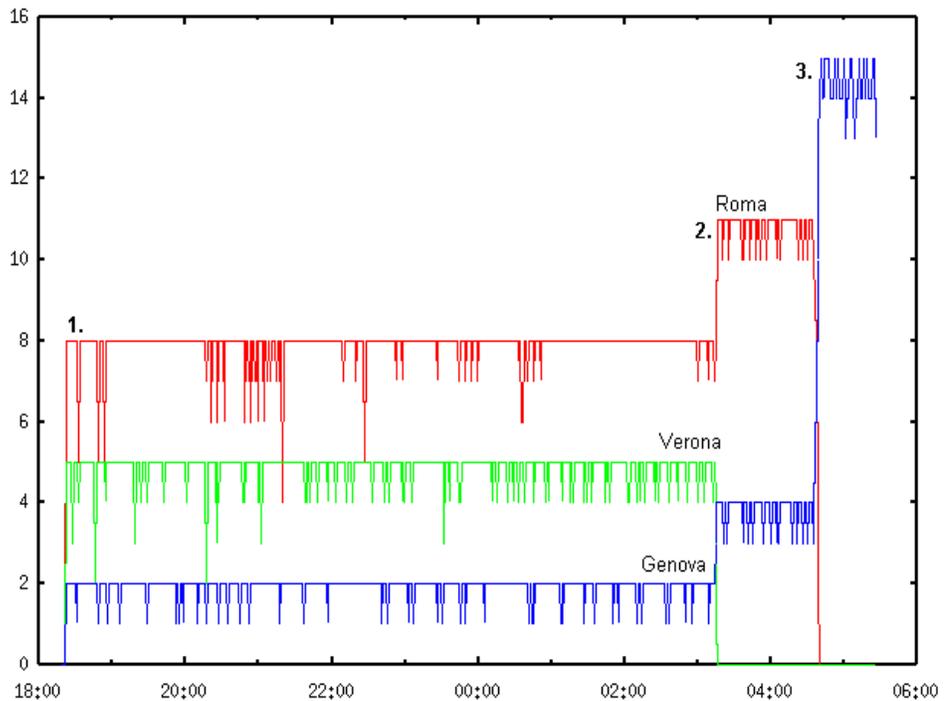
```
bqueues
QUEUE_NAME  PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND  RUN  SUSP
Roma        50  Open:Active      -  -  -  -  231  221  11   0
Verona      48  Open:Active      -  -  -  -   0    0    0   0
Genova      48  Open:Active      -  -  -  -  496  491   4   0
```

- 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  PRIO 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:

Typical Slot Allocation Scenarios

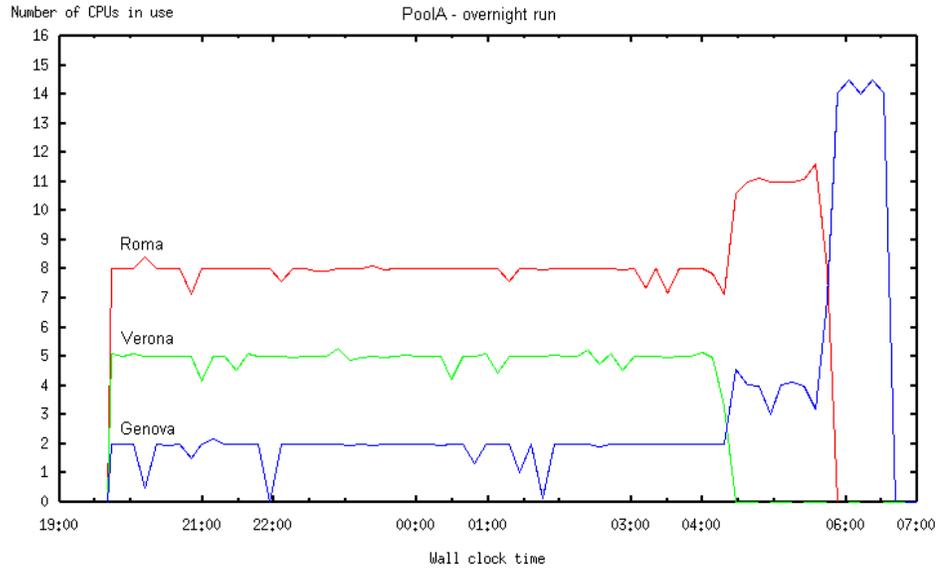


2 pools, 30 job slots, and 2 queues out of any pool

- ◆ poolA 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).
- ◆ poolB with 15 slots containing queues Pisa (30% share, 5 slots), Venezia (30% share, 5 slots), and Bologna (30% share, 5 slots).
- ◆ Two other queues Milano and Parma do not belong to any pool, but they can use the hosts of poolB. The queues from Milano to Bologna all have the same priority.

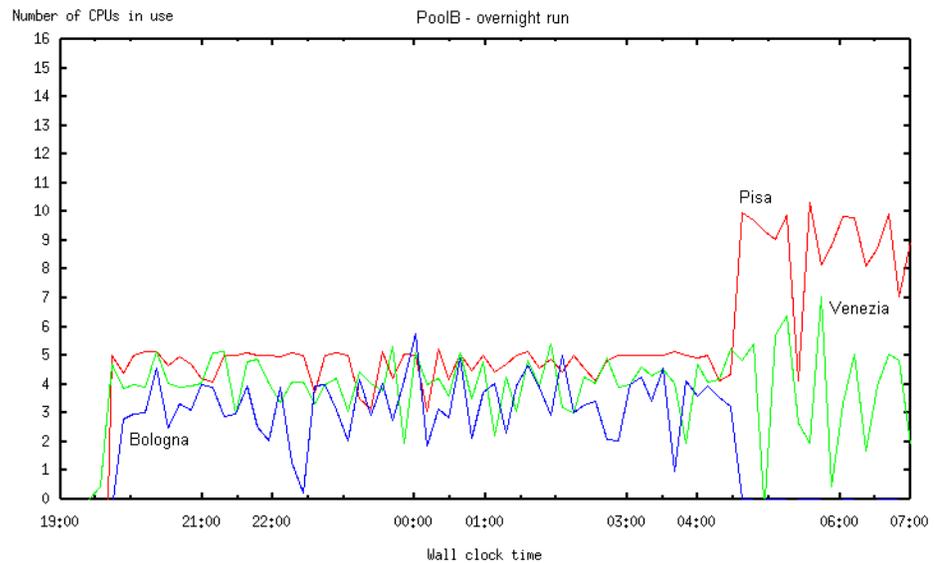
The queues Milano and Parma run very short jobs that get submitted periodically in bursts. When no jobs are running in them, the distribution of jobs looks like this:

QUEUE_NAME	PRIO	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
Pisa	44	Open:Active	-	-	-	-	1000	995	5	0
Milano	43	Open:Active	-	-	-	-	2	2	0	0
Parma	43	Open:Active	-	-	-	-	2	2	0	0
Venezia	43	Open:Active	-	-	-	-	1000	995	5	0
Bologna	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	PRIO	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 poolA 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 poolA 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 poolB, 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:

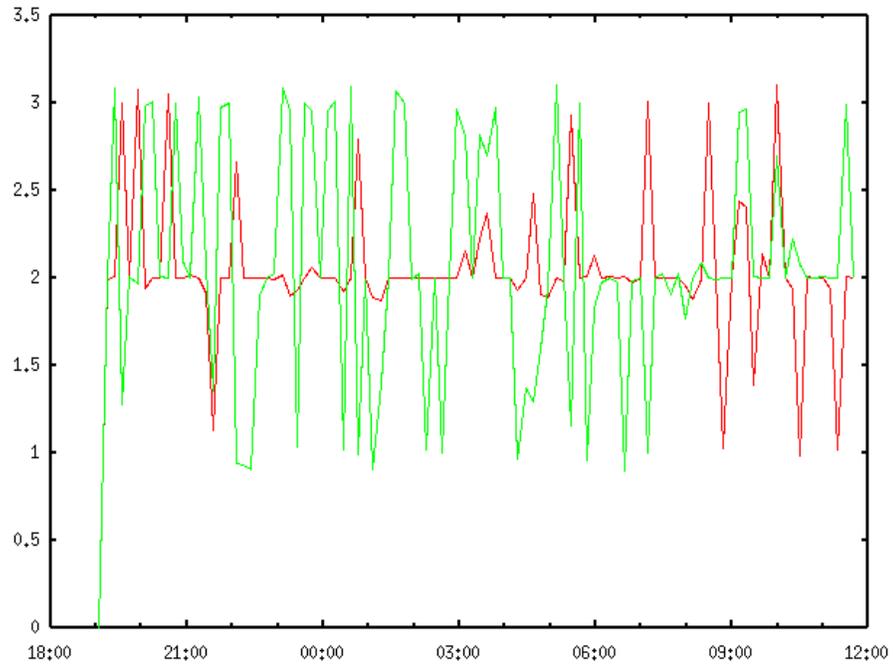
```
bqueues
QUEUE_NAME  PRIO STATUS      MAX JL/U JL/P JL/H NJOBS  PEND  RUN  SUSP
Roma        50  Open:Active   -  -  -  -    15    6   11   0
Verona      48  Open:Active   -  -  -  -    25   18    7   0
Genova      47  Open:Active   -  -  -  -   460  455    3   0
Pisa       44  Open:Active   -  -  -  -   264  261    3   0
Milano     43  Open:Active   -  -  -  -   262  259    3   0
Parma      42  Open:Active   -  -  -  -   260  257    3   0
Bologna   40  Open:Active   -  -  -  -   260  257    3   0
Sora       40  Open:Active   -  -  -  -   261  258    3   0
Ferrara   40  Open:Active   -  -  -  -   258  255    3   0
Napoli    40  Open:Active   -  -  -  -   259  256    3   0
Livorno   40  Open:Active   -  -  -  -   258  258    0   0
Palermo   40  Open:Active   -  -  -  -   256  256    0   0
Venezia    4  Open:Active   -  -  -  -   255  255    0   0
```

Initially, queues Livorno, Palermo, and Venezia in poolB 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 poolB, slots are redistributed to the other queues that originally had no jobs (queues Livorno, Palermo, and Venezia). The total slot count remains 21 in all queues in poolB.

```
bqueues
QUEUE_NAME  PRIO STATUS      MAX JL/U JL/P JL/H NJOBS  PEND  RUN  SUSP
Roma        50  Open:Active   -  -  -  -    15    6    9   0
V           48  Open:Active   -  -  -  -    25   18    7   0
Genova      47  Open:Active   -  -  -  -   460  455    5   0
Pisa       44  Open:Active   -  -  -  -   263  261    2   0
Milano     43  Open:Active   -  -  -  -   261  259    2   0
Parma      42  Open:Active   -  -  -  -   259  257    2   0
Bologna   40  Open:Active   -  -  -  -   259  257    2   0
Sora       40  Open:Active   -  -  -  -   260  258    2   0
Ferrara   40  Open:Active   -  -  -  -   257  255    2   0
Napoli    40  Open:Active   -  -  -  -   258  256    2   0
Livorno   40  Open:Active   -  -  -  -   258  256    2   0
Palermo   40  Open:Active   -  -  -  -   256  253    3   0
Venezia    4  Open:Active   -  -  -  -   255  253    2   0
```

The following figure illustrates the round-robin distribution of slot allocations between queues Livorno and Palermo:



How LSF rebalances slot usage

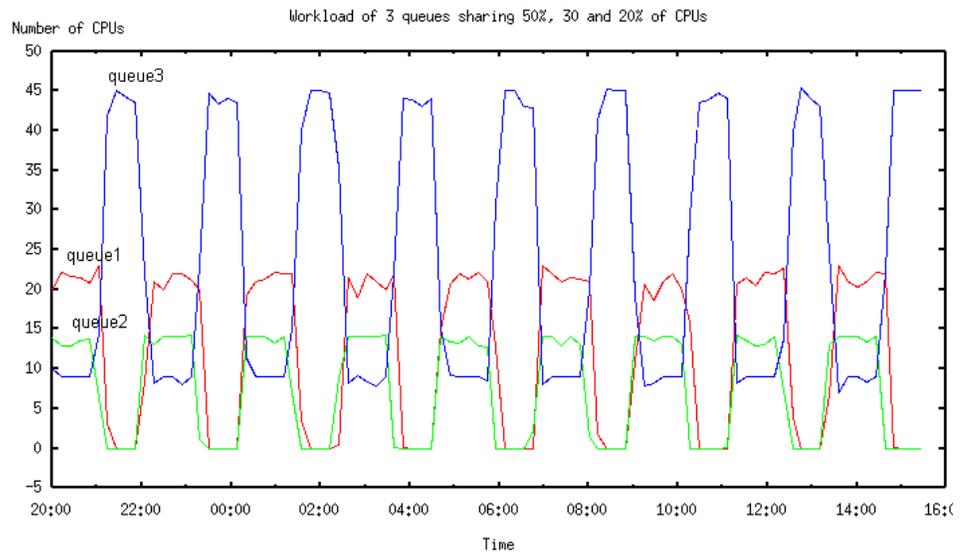
In the following examples, job runtime is not equal, but varies randomly over time.

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

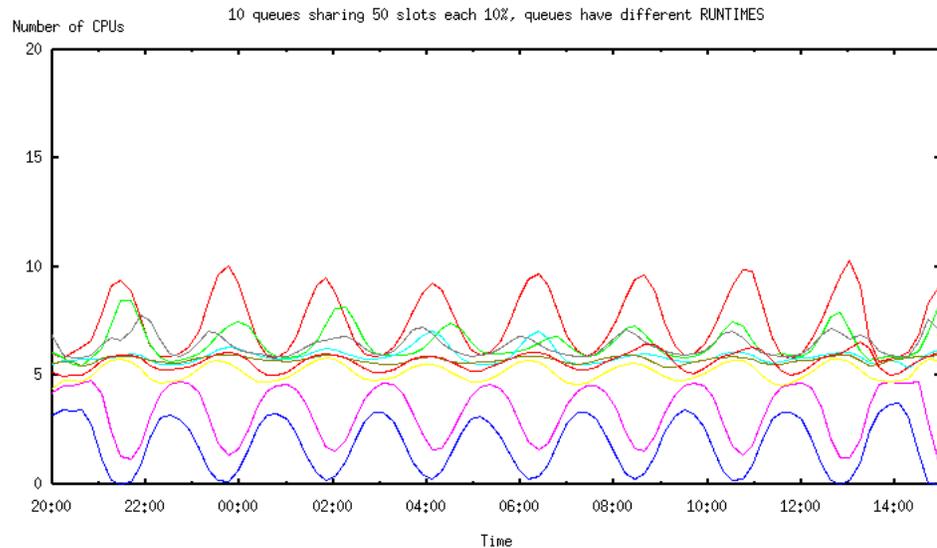
As queue1 and queue2 finish their jobs, the number of jobs in queue3 expands, and as queue1 and queue2 get more work, LSF rebalances the usage:



Using Historical and Committed Run Time

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.



Using Historical 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 either or both of the following weighting factors:

- ◆ 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` in `lsb.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 + 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 `lsb.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 `lsb.events`, according to the value of `CLEAN_PERIOD` in `lsb.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 `lsb.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.

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.

Configure committed run time

- 1 Set a value for the `COMMITTED_RUN_TIME_FACTOR` parameter in `lsb.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 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.

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`

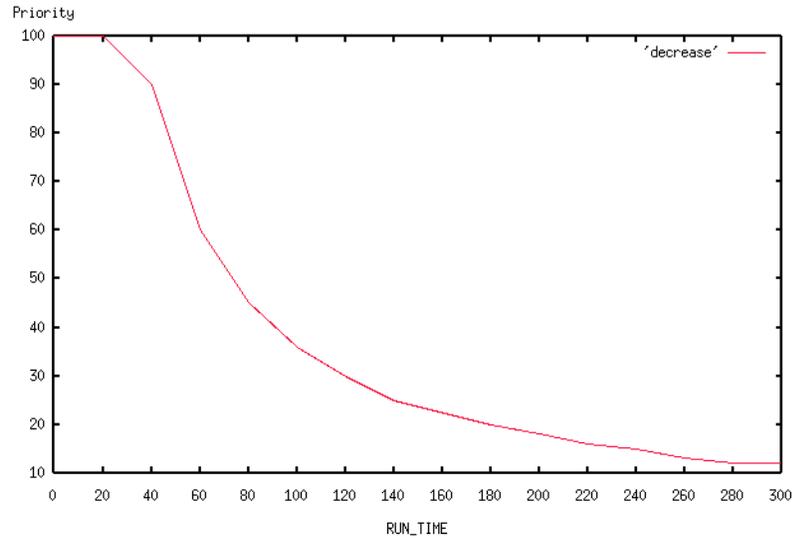
The run time displayed by `bqueues` and `bhpart` is the sum of the actual, accumulated run time and the historical run time, but does not include the committed run time.

Example

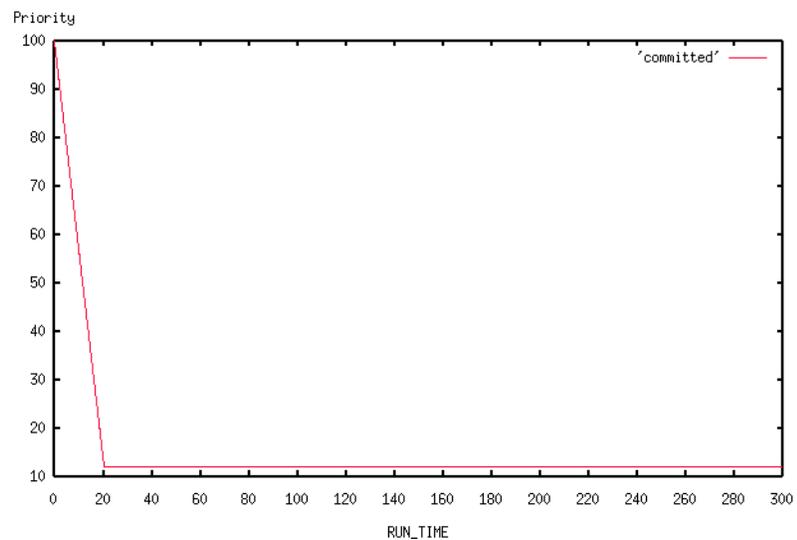
The following fairshare parameters are configured in `lsb.params`:

```
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:



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

- 1 To associate a job with a user group for the purposes of fairshare scheduling, use `bsub -G` and specify a group that you belong to. If you use hierarchical fairshare, you must specify a group that does not contain any subgroups.

Example

User1 shares resources with groupA and groupB. User1 is also a member of groupA, but not any other groups.

User1 submits a job:

```
bsub sleep 100
```

By default, the job could be considered for dispatch if either User1 or GroupA has highest dynamic share priority.

User1 submits a job and associates the job with GroupA:

```
bsub -G groupA sleep 100
```

If User1 is the highest priority user, this job will not be considered.

- ◆ User1 can only associate the job with a group that he is a member of.
- ◆ User1 cannot associate the job with his individual user account, because `bsub -G` only accepts group names.

Example with hierarchical fairshare

In the share tree, User1 shares resources with GroupA at the top level. GroupA has 2 subgroups, B and C. GroupC has 1 subgroup, GroupD. User1 also belongs to GroupB and GroupC.

User1 submits a job:

```
bsub sleep 100
```

By default, the job could be considered for dispatch if either User1, GroupB, or GroupC has highest dynamic share priority.

User1 submits a job and associates the job with GroupB:

```
bsub -G groupB sleep 100
```

If User1 or GroupC is the highest priority user, this job will not be considered.

- ◆ User1 cannot associate the job with GroupC, because GroupC includes a subgroup.
- ◆ User1 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.

Resizable jobs and fairshare

Resource Preemption

Contents

- ◆ [About Resource Preemption](#) on page 370
- ◆ [Requirements for Resource Preemption](#) on page 371
- ◆ [Custom Job Controls for Resource Preemption](#) on page 371
- ◆ [Resource Preemption Steps](#) on page 373
- ◆ [Configure Resource Preemption](#) on page 375
- ◆ [License Preemption Example](#) on page 377
- ◆ [Memory Preemption Example](#) on page 379

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 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.

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).

Other Resources 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.

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

Using 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** 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.
- 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 `lsb.params`).
- ◆ Jobs must reserve the correct amount of the preemption resource, using resource reservation (the `usage` 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 `lsf.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 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.

Customizing 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.

Killing 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 `lsb.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 `lsf.shared`

Define the custom resource in the Resource section.

c `lsb.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.

Resource Preemption Steps

- ◆ 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 `lsf.cluster.cluster_name`
Define how the custom resource is shared in the ResourceMap section.
 - e `lsf.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 events in your cluster.
-

Configure Resource Preemption

- 1 Configure preemptive scheduling (PREEMPTION in `lsb.queues`).
- 2 Configure the preemption resources (PREEMPTABLE_RESOURCES in `lsb.params`).

Job slots are the default preemption 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 preemption action.

Preemptive scheduling uses the SUSPEND and RESUME job control actions to suspend and resume preempted jobs. For resource preemption, it is critical that the preempted job releases the resource. You must modify LSF default job controls to make resource preemption work.

- ❖ Suspend using a custom job control.

To modify the default suspend action, set JOB_CONTROLS in `lsb.queues` and use `replace` to 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 preemption resources.

For example, if your application vendor tells you to use the SIGTSTP signal, set JOB_CONTROLS in `lsb.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 `lsb.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 preemption 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 preemption resources.

If you do this, the preempted job does not get to run unless you resubmit it.

- 4 Optional. Configure the preemption wait time.

To specify how long LSF waits for the ELIM to report that the resources are available, set PREEMPTION_WAIT_TIME in `lsb.params` and specify the number of seconds to wait. You cannot specify any less than the default time (300 seconds).

Configure Resource Preemption

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]
```

License Preemption Example

```
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 ...
```

Using 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_HOS 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 ...
```

Preempting 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]
```

Memory Preemption Example

```
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
lsload > /dev/null 2>&1
if [ $? != 0 ] ; then exit 1
fi
memStr=`lsload -I mem -w $host|grep $host|awk '{print $3}'|sed 's/M//`
reportStr="1 "pre_mem "$memStr"
echo "$reportStr \c"
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
...
```

Using 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
...
```

Preempting 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

```
...
```

Memory Preemption Example

Goal-Oriented SLA-Driven Scheduling

Contents

- ◆ [Using Goal-Oriented SLA Scheduling](#) on page 383
- ◆ [Configuring Service Classes for SLA Scheduling](#) on page 386
- ◆ [View Information about SLAs and Service Classes](#) on page 388
- ◆ [Understanding Service Class Behavior](#) on page 392

Using Goal-Oriented SLA Scheduling

Goal-oriented SLA scheduling policies help you configure your workload so that your jobs are completed on time and reduce the risk of missed deadlines. They enable you to focus on the “what and when” of your projects, not the low-level details of “how” resources need to be allocated to satisfy various workloads.

Service-level agreements in LSF

A *service-level agreement* (SLA) defines how a service is delivered and the parameters for the delivery of a 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

An SLA in LSF is a “just-in-time” scheduling policy that defines an agreement between LSF administrators and LSF users. The SLA scheduling policy defines how many jobs should be run from each SLA to meet the configured goals.

RESTRICTION: LSF MultiCluster does not support SLAs.

Service classes

SLA definitions consist of service-level goals that are expressed in individual *service classes*. A service class is the actual configured policy that sets the service-level goals for the LSF system. The SLA defines the workload (jobs or other services) and users that need the work done, while the service class that addresses the SLA defines individual goals, and a time window when the service class is active.

Service-level goals

You configure the following kinds of goals:

Deadline goals

A specified number of jobs should be completed within a specified time window. For example, run all jobs submitted over a weekend.

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.

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.

Combining different types of goals

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.

How service classes perform goal-oriented scheduling

Goal-oriented 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 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 service class to meet its service-level 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 SLA 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 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.

Submit jobs to a service class

You submit jobs to a service class as you would to a queue, except that a service class is a higher level scheduling policy that 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 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.

Outside of the configured time windows, the SLA is not active, and LSF schedules jobs without enforcing any service-level goals. Jobs will flow through queues following queue priorities even if they are submitted with `-sla`.

-
- 1 Run `bsub -sla service_class_name` to submit a job to a service class for SLA-driven scheduling.

```
bsub -W 15 -sla Kyuquot sleep 100
```

submits the UNIX command `sleep` together with its argument `100` as a job to the service class named `Kyuquot`.
-

Submitting with a run limit

You should submit your jobs with a run time limit at the job level (`-W` option), the application level (`RUNLIMIT` parameter in the application definition in `lsb.applications`), 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.applications`) instead of or in conjunction with the run time limit.

The following table describes how LSF uses the values that you provide for SLA-driven 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.

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 Kyuquot 2307
```

Attaches job 2307 to the service class `Kyuquot`.

```
bmod -slan 2307
```

Detaches job 2307 from the service class `Kyuquot`.

You cannot:

- ◆ Use `-sla` with other `bmod` options
- ◆ Move job array elements from one service class to another, only entire job arrays
- ◆ Modify the service class of jobs already attached to a job group

If a default SLA is configured in `lsb.params`, `bmod -slan` moves the job to the default SLA. If the job is already attached to the default SLA, `bmod -slan` has no effect on that job.

Configuring Service Classes for SLA Scheduling

Configure service classes in

`LSB_CONFDIR/cluster_name/configdir/lsb.serviceclasses`. Each service class is defined in a `ServiceClass` section.

Each service class section begins with the line `Begin ServiceClass` and ends with the line `End ServiceClass`. You must specify:

- ◆ A service class name
- ◆ At least one goal (deadline, throughput, or velocity) and a time window when the goal is active
- ◆ A service class priority

All other parameters are optional. You can configure as many service class sections as you need.

IMPORTANT: The name you use for your service classes cannot be the same as an existing host partition or user group name.

User groups for service classes

You can control access to the SLA 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.

Service class priority

A higher value indicates a higher priority, relative to other service classes. Similar to queue priority, service classes access the cluster resources in priority order.

LSF schedules jobs from one service class at a time, starting with the highest-priority service class. If multiple service classes have the same priority, LSF runs the jobs from these service classes in the order the service classes are configured in `lsb.serviceclasses`.

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.

Service class configuration examples

- ◆ The service class `Uclulet` 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 = Uclulet
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 `Inuvik` defines a throughput goal of 6 jobs per hour that is always active:

```
Begin ServiceClass
NAME = Inuvik
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
```

View Information about SLAs and Service Classes

```
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 `Kyuquot` 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 = Kyuquot  
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 `Kyuquot`, 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
```

View Information about SLAs and Service Classes

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

- ◆ 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 `Uclulet` is not being met, and `bsla` displays status `Active:Delayed`:

```
bsla
SERVICE CLASS NAME: Uclulet
-- 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 `Kyuquot` is active and on time. The configured deadline goal of the service class is inactive.

```
bsla Kyuquot
SERVICE CLASS NAME: Kyuquot
-- 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 `Inuvik` is always active. `bsla` 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 Inuvik
SERVICE CLASS NAME: Inuvik
-- 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
```

View Information about SLAs and Service Classes

```
      NJOBS  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 Inuvik
```

```
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
```

Use `-sla` with `-g` to display job groups attached to a service class. Once a job group is attached to a 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 `Inuvik` and `Tuktoyaktuk` configure throughput goals.

```
bsla
SERVICE CLASS NAME: Inuvik
-- 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
```

```
      NJOBS  PEND  RUN  SSUSP  USUSP  FINISH
      111    94   5     0     0     12
```

```
-----
SERVICE CLASS NAME: Tuktoyaktuk
-- 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
```

```
      NJOBS  PEND  RUN  SSUSP  USUSP  FINISH
      104    96   4     0     0     4
```

These two service classes have the following historical performance. For SLA `Inuvik`, `bacct` shows a total throughput of 8.94 jobs per hour over a period of 20.58 hours:

```
bacct -sla Inuvik
```

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 Inuvik,

```

-----
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 Tuktoyaktuk, bacct shows a total throughput of 4.36 jobs per hour over a period of 19.95 hours:

bacct -sla Tuktoyaktuk

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 Tuktoyaktuk,

```

-----
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.

Understanding Service Class Behavior

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. `bsla` shows that the deadline can be met:

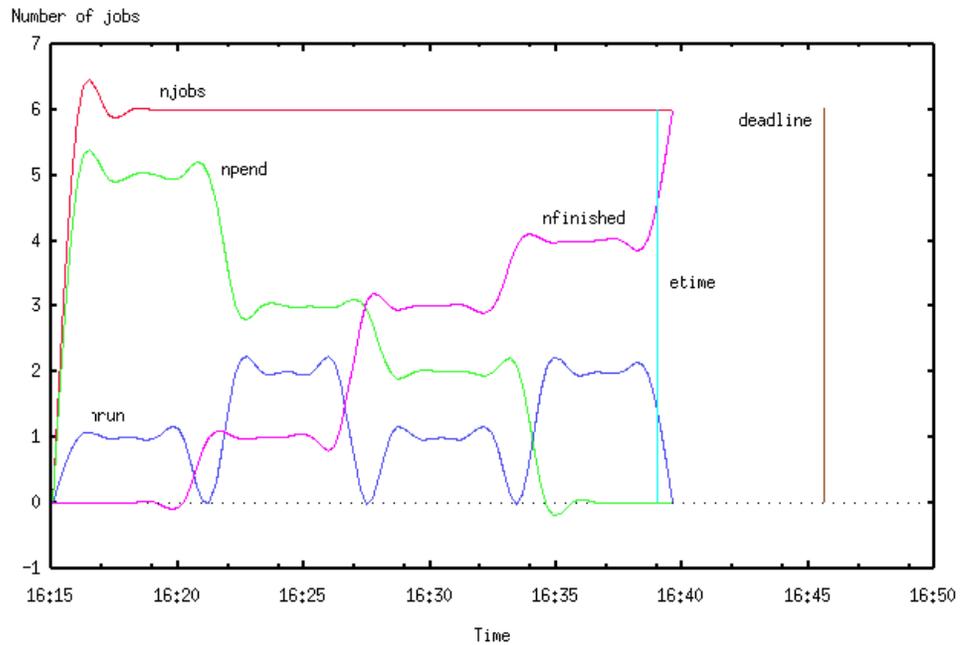
```
bsla Quadra
SERVICE CLASS NAME: Quadra
-- short window
PRIORITY: 20

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

bsla shows the configuration and status of two service classes Qualicum and Comox:

- ◆ Qualicum has a deadline goal with a time window which is active overnight:

```
bsla Qualicum
SERVICE CLASS NAME: Qualicum
PRIORITY: 23

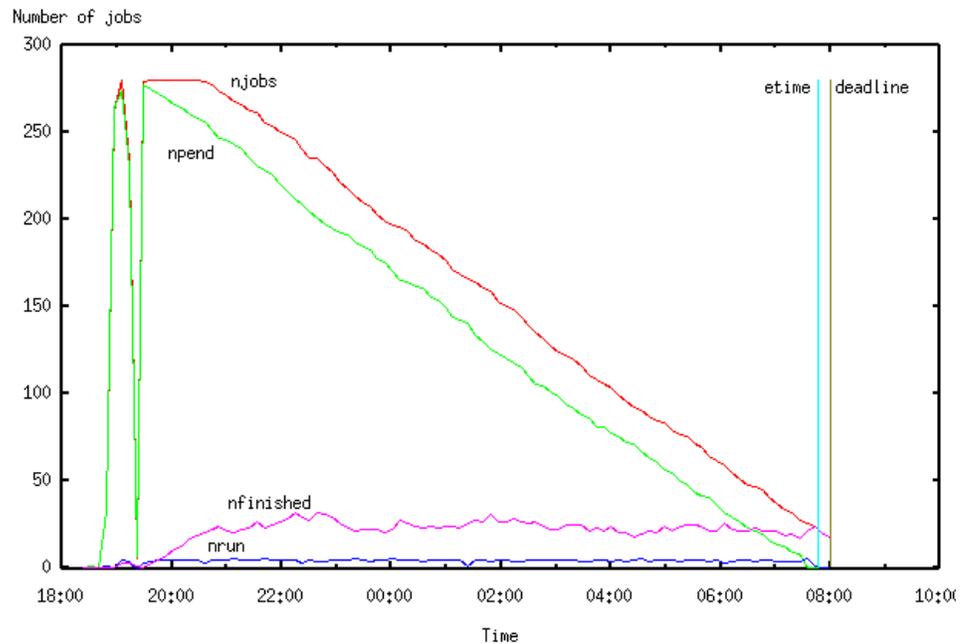
GOAL: VELOCITY 8
ACTIVE WINDOW: (8:00-18:00)
STATUS: Inactive
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

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 Qualicum 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.

Understanding Service Class Behavior



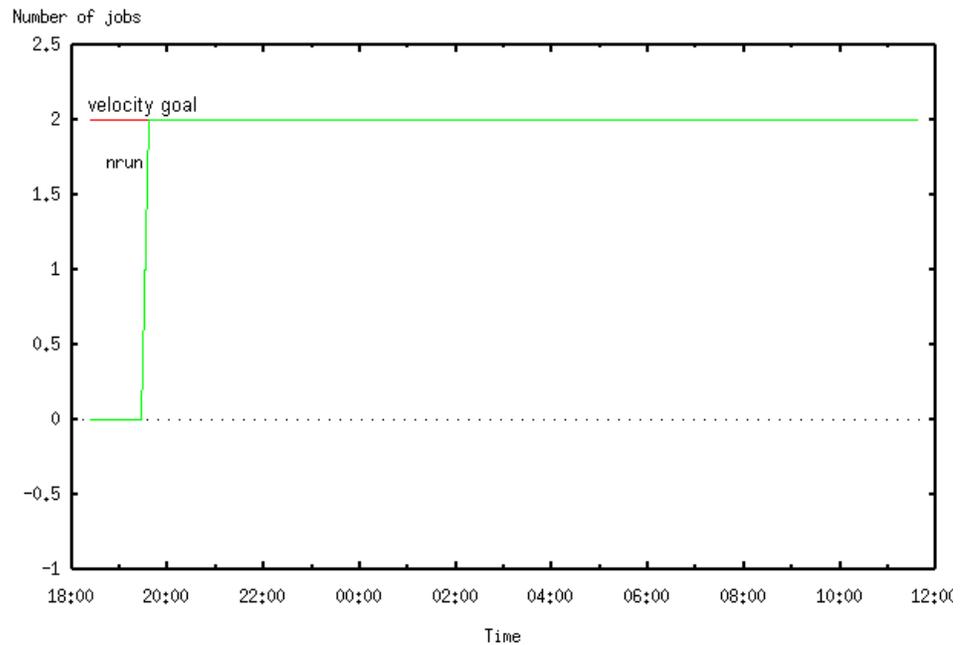
- ◆ Comox has a velocity goal of 2 concurrently running jobs that is always active:

```
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.



When an SLA is missing its goal

- 1 Use the `CONTROL_ACTION` parameter in your service class to configure an action to be run if the SLA goal is delayed for a specified number of minutes.

CONTROL_ACTION (lsb.serviceclasses)

```
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 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]
```

Preemption and SLA policies

SLA jobs cannot be preempted. You should avoid running jobs belonging to an SLA in low priority queues.

Chunk jobs and SLA policies

SLA jobs will not get chunked. You should avoid submitting SLA jobs to a chunk job queue.

SLA statistics files

Each active SLA goal generates a statistics file for monitoring and analyzing the system. When the goal becomes inactive the file is no longer updated. The 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
```

Resizable jobs and SLA scheduling

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.

Job groups and SLA scheduling

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 and 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 `-sla` to specify a service class when you create a job group with `bgadd`.

View job groups attached to an SLA (bjgroup)

1 Run `bjgroup` to display job groups attached to a service class:

```
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
```

`bjgroup` displays the name of the service class that the job group is attached to with `bgadd -sla service_class_name`. If the job group is not attached to any service class, empty parentheses () are displayed in the SLA name column.

Job Scheduling and Dispatch

- ◆ [Resource Allocation Limits](#) on page 419
- ◆ [Reserving Resources](#) on page 437
- ◆ [Advance Reservation](#) on page 453
- ◆ [Dispatch and Run Windows](#) on page 477
- ◆ [Job Dependencies](#) on page 481
- ◆ [Job Priorities](#) on page 489
- ◆ [Job Requeue and Job Rerun](#) on page 503
- ◆ [Job Checkpoint, Restart, and Migration](#) on page 513
- ◆ [Chunk Job Dispatch](#) on page 519
- ◆ [Job Arrays](#) on page 525
- ◆ [Running Parallel Jobs](#) on page 537
- ◆ [Submitting Jobs Using JSDL](#) on page 581

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.

Contents

- ◆ [Manage application profiles](#) on page 402
- ◆ [View application profile information](#) on page 407
- ◆ [Use application profiles](#) on page 405
- ◆ [How application profiles interact with queue and job parameters](#) on page 411

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
DATALIMIT     = 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 or remove application profiles

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 `badmin reconfig` to reconfigure `mbatchd`.

Adding an application profile does not affect pending or running jobs.

Remove an application profile

Prerequisites: 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 `badadmin reconfig` 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 `badadmin reconfig` to reconfigure `mbatchd`.
Adding an application profile does not affect pending or running jobs.

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 `badmin reconfig` to reconfigure `mbatchd`.

Understanding 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 counted in the `EXIT_RATE` calculation.

0 always indicates application success regardless of `SUCCESS_EXIT_VALUES`.

If both `SUCCESS_EXIT_VALUES` and `QUEUE_EXIT_VALUES` are defined, job will be set to `PEND` state and queued.

`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.

Use application profiles

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

Prerequisites: 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`

Use application profiles

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	bjobs -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 bhist -l bjobs -l

View available application profiles

- 1 Run `bapp`. You can view a particular application profile or all profiles.

```
bapp
APPLICATION_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
```

View application profile information

```
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: Submitted from host <hostA> with hold, CWD
</clusters/lsf7.0/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	tlu
loadSched	-	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-	-
	cpuspeed		bandwidth									
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: Submitted from host <hostA>, CWD <${HOME}/src/mainline/lSBATCH
/cmd>;
Wed May 31 16:52:48: Dispatched to 10 Hosts/Processors <10*hostA>
Wed May 31 16:52:48: 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>, Co
mmand <myjob 1234>
Tue Dec 11 23:32:00: 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

View application profile information

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: Submitted from host <hostA> with hold, to Queue <norma

l>, CWD </scratch/dev/lsfjobs/user1/work>, Requeue Exit Values <1>;

Tue Dec 11 23:33:14: 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, Command <myjob 1234>

...

Sun Oct 7 22:30:19: Done successfully. Success Exit Code: 230 222 12.

...

`bacct -l 405`

...

Job <405>, User <user1>, Project <default>, Status <DONE>, Queue <normal>, Command <myjob 1234>

Wed Sep 26 18:37:47: Submitted from host <hostA>, CWD </scratch/dev/lsfjobs/user1/work>;

Wed Sep 26 18:37:50: Dispatched to <hostA>;

Wed Sep 26 18:37:51: 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
- ◆ MIG
- ◆ 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	job-level section overwrites application-level section, which overwrites queue-level section (if a given level is present)
cu	
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
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 $num1*\{rusage[R1]\} + num2*\{rusage[R2]\}$ and the queue-level requirement is $rusage[RQ]$ where RQ is a job resource, the merged requirement is $num1*\{rusage[merge(R1,RQ)]\} + num2*\{rusage[R2]\}$</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 bjobs.

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.

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.applications.

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.applications.

The job-level runtime estimate setting overrides the RUNTIME setting in an application profile in lsb.applications.

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`

Configuring 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 `lsb.params` or in the file `lsb.applications` for the application associated with your job.

LSF calculates the normalized run time using the following formula:

$$\text{NORMALIZED_RUN_TIME} = \text{RUNTIME} * \text{CPU_Factor_Normalization_Host} / \text{CPU_Factor_Execute_Host}$$

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...
<code>DEFAULT_HOST_SPEC</code>	<code>lsb.queues</code>	LSF selects the default normalization host for the queue.
<code>DEFAULT_HOST_SPEC</code>	<code>lsb.params</code>	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 `lsf.cluster.clustername`) or host model (defined in `lsf.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 `hostA`.

```
RUNTIME=210/Ultra5S
bsub -We 210/Ultra5S
```

LSF calculates the normalized run time using the CPU factor defined for host model `Ultra5S`.

TIP: Use `lsinfo` 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 `lsb.applications`, or queue level with `RUNLIMIT` in `lsb.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.

- 3 For chunk jobs, ensure that the estimated runtime value is
 - ❖ Less than the `CHUNK_JOB_DURATION` defined in the file `lsb.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 `lsb.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 $T1 * ratio$ T2 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

Job-runtime estimate	Job-run limit	Application runtime estimate	Application run limit	Queue default run limit	Queue hard run limit	Result
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
T1	—	—	—	T5	T6	Job is accepted Jobs running longer than $T1 * ratio$ are killed If $T1 * ratio > T6$, the job is rejected

How application profiles interact with queue and job parameters

Resource Allocation Limits

Contents

- ◆ [About Resource Allocation Limits](#) on page 420
- ◆ [Configuring Resource Allocation Limits](#) on page 425
- ◆ [Viewing Information about Resource Allocation Limits](#) on page 434

About Resource Allocation Limits

Contents

- ◆ [What resource allocation limits do](#) on page 420
- ◆ [How LSF enforces limits](#) on page 421
- ◆ [How LSF counts resources](#) on page 421
- ◆ [Limits for resource consumers](#) on page 423

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 rusage` 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_limit1
MEM = 4
End Limit
```

A job submitted with an `rusage` resource requirement that exceeds this limit:

```
bsub -R "rusage[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"rusage[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 allocation limits and resource usage 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. See Chapter 36, “[Runtime Resource Usage Limits](#)” for more information.

How 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 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` (1sb.queues), `HJOB_LIMIT` (1sb.queues), `UJOB_LIMIT` (1sb.queues), `MXJ` (1sb.hosts), `JL/U` (1sb.hosts), `MAX_JOBS` (1sb.users), and `MAX_PEND_JOBS` (1sb.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.

About Resource Allocation Limits

Job limits Job limits, specified by JOBS in a Limit section in `lsb.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.resources` 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                sleep 1000  Aug 30 16:26
18     user1  PEND  normal hosta                sleep 1000  Aug 30 16:26
19     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 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 1 job running in the short queue, and two jobs 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		sleep 1000	Aug 30 16:26
19	user1	RUN	short	hosta	hosta	sleep 1000	Aug 30 16:26
16	user1	RUN	normal	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` 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.

Configuring Resource Allocation Limits

Contents

- ◆ [lsb.resources file](#) on page 425
- ◆ [Enable resource allocation limits](#) on page 426
- ◆ [Configure cluster-wide limits](#) on page 426
- ◆ [Compatibility with pre-version 7 job slot limits](#) on page 426
- ◆ [How resource allocation limits map to pre-version 7 job slot limits](#) on page 427
- ◆ [How conflicting limits are resolved](#) on page 428
- ◆ [Example limit configurations](#) on page 430

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.

Resource parameters

To limit ...	Set in a <code>Limit</code> section of <code>lsb.resources</code> ...
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 <code>lsb.resources</code> ...
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 each specified project	PER_PROJECT

Enable resource allocation limits

- 1 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

- 1 To configure limits that take effect for your entire cluster, configure limits in `lsb.resources`, but do not specify any consumers.

Compatibility with pre-version 7 job slot limits

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 will continue to apply.

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 (lsb.resources)	Resource consumers (lsb.resources)					Equivalent existing limit (file)
	USERS	PER_USER	QUEUES	HOSTS	PER_HOST	
SLOTS	—	all	—	<i>host_name</i>	—	JL/U (lsb.hosts)
SLOTS_PER_PROCESSOR	<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_PROCESSOR	—	—	<i>queue_name</i>	—	all	PJOB_LIMIT (lsb.queues)
SLOTS	—	—	<i>queue_name</i>	—	—	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

How conflicting limits are resolved

LSF handles two kinds of limit conflicts:

- ◆ [Similar conflicting limits](#)
- ◆ [Equivalent conflicting limits](#)

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 user1 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 `lsb.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 `lsb.resources` that are equivalent to existing limits in `lsb.users`, `lsb.hosts`, or `lsb.queues`, but with a different value override the existing limits. The equivalent limits in `lsb.users`, `lsb.hosts`, or `lsb.queues` are ignored, and the value of the new limit in `lsb.resources` is used.

For example, a *per-user* job slot limit in `lsb.resources` is equivalent to a `MAX_JOBS` limit in `lsb.users`, so only the `lsb.resources` limit is enforced, the limit in `lsb.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.

Stopping and resuming jobs

Jobs stopped with `bstop`, 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 (`brresume`) 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 is 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 PENDING or PSUSP. Usage of JOBS limit is decreased by the number of requeued jobs.
- ◆ Checkpointed jobs restarted with `brstart` 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 `Limit` section enclosed by `Begin Limit` and `End Limit`.

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 Limit
NAME    HOSTS    SLOTS  MEM   SWP  TMP   USERS    QUEUES
Limit1  hostA    2      -     -    -     user1    -
-       -        -      20    -    -     user2    normal
End Limit
```

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 Limit
HOSTS  SLOTS  PROJECTS  USERS    QUEUES
hostA  2      -         user1    normal
-      1      test      user1    -
End Limit
```

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 Limit
HOSTS      SLOTS  MEM    PER_QUEUE
license1   10     -      normal
license1   -       200    short
license1   30     300    (all ~normal ~short)
End Limit

```

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

```

Configuring Resource Allocation Limits

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 `develop` group to 1 job on each host, and 50% of the memory on the host.

```
Begin Limit
NAME = develop_group_limit
# Resources:
SLOTS = 1
MEM = 50%
#Consumers:
USERS = develop
PER_HOST = all
End Limit
```

Example 9

Limit software license `lic1`, with quantity 100, where `user1` 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
```

Viewing 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.

Viewing Information about Resource Allocation Limits

- ◆ 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

Contents

- ◆ [About Resource Reservation](#) on page 437
- ◆ [Using Resource Reservation](#) on page 438
- ◆ [Memory Reservation for Pending Jobs](#) on page 440
- ◆ [Time-based Slot Reservation](#) on page 442
- ◆ [Viewing Resource Reservation Information](#) on page 450

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.

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, 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
```

Using 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 `bjobs`. 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.

Configuring resource reservation at the queue level

Queue-level resource reservation can be configured as part of the RES_REQ parameter. The resource reservation requirement can be configured at the queue level as part of the queue level resource requirements. Use the resource usage (`rusage`) section of the resource requirement string to specify the amount of resources a job should reserve after it is started.

Examples

```
Begin Queue
.
RES_REQ = select[type==any] rusage[swp=100:mem=40:duration=60]
.
End Queue
```

This will allow a job to be scheduled on any host that the queue is configured to use and will reserve 100 MB of swap and 40 MB of memory for a duration of 60 minutes.

```
Begin Queue
```

```
RES_REQ = select[type==any] rusage[mem=20||mem=10:swp=20]
```

```
End Queue
```

This will allow a job to be scheduled on any host that the queue is configured to use. The job will attempt to reserve 20 MB of memory, or 10 MB of memory and 20 MB of swap if the 20 MB of memory is unavailable.

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 ReservationUsage
RESOURCE          METHOD
licenseX          PER_JOB
licenseY          PER_HOST
licenseZ          PER_SLOT
End ReservationUsage
```

Only user-defined numeric resources can be reserved. Builtin resources like `mem`, `cpu`, `swp`, etc. cannot be configured in the `ReservationUsage` section.

The cluster-wide `RESOURCE_RESERVE_PER_SLOT` parameter in `lsb.params` is obsolete. Configuration in `lsb.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 `lsb.resources`. Resources not reserved in `lsb.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=config`).

Memory Reservation for Pending Jobs

About memory reservation for pending jobs

By default, the `usage` 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 `usage` 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.

Configure memory reservation for pending jobs

RESOURCE_RESERVE parameter

- 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.

Configure lsb.modules

- 1 To enable memory reservation for sequential jobs, add the LSF scheduler plugin module name for resource reservation (`schmod_reserve`) to the `lsb.modules` file:

```
Begin PluginModule
SCH_PLUGIN          RB_PLUGIN          SCH_DISABLE_PHASES
schmod_default      ()                  ()
schmod_reserve      ()                  ()
schmod_preemption   ()                  ()
End PluginModule
```

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.

See [Examples](#) on page 449 for examples of jobs that use memory reservation.

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 host-based MXJ limit is only constraint to job slot request. In complex environments, where more than one constraints exist, 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[ptile=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 vs. greedy slot reservation

With time-based reservation, a set of pending jobs get future allocation and an estimated start time, so that job can be placed in the future. Reservations are made based on future allocation to guarantee estimated start time.

Time-based resource reservation provides a more accurate pending job predicted start time. When calculating job predicted start time, 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 will 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:

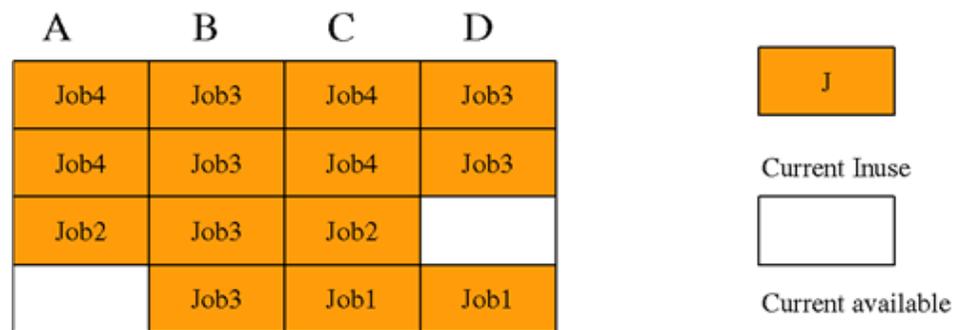
```
Job will start no sooner than indicated time stamp
```

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
bjobs displays best estimate	Yes	No
bjobs 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: $FT(\text{Job2}) < FT(\text{Job4}) < FT(\text{Job1}) < FT(\text{Job3})$.



The highest priority pending job requests `-n 6 -R "span[ptile=2]"`. Based on this, the job needs three hosts and two CPUs on each host. The default greedy slot reservation calculates job start time as the job finish time of Job4; after Job4 finishes, requested slots of pending job can be satisfied.

Configuring time-based slot reservation

Greedy slot reservation is the default slot reservation mechanism and time-based slot reservation is disabled.

LSB_TIME_RESERVE_NUMJOBS (lsf.conf)

- 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.

Some 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:

A	B	C	D
Job4	Job3	Job4	Job3
Job4	Job3	Job4	Job3
Job2	Job3	Job2	
	Job3	Job1	Job1



J

Current Inuse

Current available

- 2 Job2 finishes, freeing 2 more CPUs for future allocation, 1 on host A, and 1 on host C:

Time-based Slot Reservation

A	B	C	D
Job4	Job3	Job4	Job3
Job4	Job3	Job4	Job3
	Job3		
	Job3	Job1	Job1

Future available

J

Current Inuse

Current available

3 Job4 finishes, freeing 4 more CPUs for future allocation, 2 on host A, and 2 on host C:

A	B	C	D
	Job3		Job3
	Job3		Job3
	Job3		
	Job3	Job1	Job1

Future available

J

Current Inuse

Current available

4 Job1 finishes, freeing 2 more CPUs for future allocation, 1 on host C, and 1 host D:

A	B	C	D
	Job3		Job3
	Job3		Job3
	Job3		
	Job3		

Future available

J

Current Inuse

Current available

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
	Job3		Job3
Job5	Job3	Job5	Job5
Job5	Job3	Job5	Job5

J

Future allocation

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.queue`s, 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 `lsb.resources` (`SLOTS`, `PER_SLOT`)
- ◆ `MXJ`, `JL/U` in `lsb.hosts`
- ◆ `PJOB_LIMIT`, `HJOB_LIMIT`, `QJOB_LIMIT`, `UJOB_LIMIT` in `lsb.queues`

Memory request

To request memory resources, configure `RESOURCE_RESERVE` in `lsb.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: `alloc=2*hostA 3*hostB`
- ◆ LSF integrations define their own future allocation string to override the default LSF allocation. For example, in RMS, future allocation is displayed as:
`rms_alloc=2*sierra0 3*sierra1`

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 `lsf.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: Reserved <2> job slots on host(s) <2*qat26>;
Sat Apr 23 03:28:46: Estimated Job Start Time;
      alloc=2*qat25 2*qat24 2*qat26.lsf.platform.com
```

Example 2

Two RMS hosts, `sierraA` and `sierraB`, 8 CPUs per host. Job 3873 uses 4*`sierra0` and will last for 10 hours. Job 3874 uses 4*`sierra1` and will run for 12 hours. Job 3875 uses 2*`sierra2` and 2*`sierra3`, and will run for 13 hours.

Job 3876 is submitted and requests `-n 6 -ext "RMS[nodes=3]"`.

```
bjobs -l 3876
Job <3876>, User <user2>, Project <default>, Status <PEND>, Queue <rms>, Extsch
      ed <RMS[nodes=3]>, Command <sleep 1000000>
Fri Apr 22 15:35:28: Submitted from host <sierraA>, CWD <${HOME}>, 6 Processors R
      equested;
RUNLIMIT
  840.0 min of sierraA
Fri Apr 22 15:35:46: Reserved <4> job slots on host(s) <4*sierraB>;
Sat Apr 23 01:34:12: Estimated job start time;
      rms_alloc=2*sierra[0,2-3]
...
```

Viewing Resource Reservation Information

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: 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: Reserved <2> job slots on host(s) <2*qat26.lsf.platform.co
      m>;
Sat Apr 23 01:39:13: Job will start no sooner than indicated time stamp;
```

Viewing 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 r15s r1m r15m ut pg io ls it tmp swp mem
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.

- 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: Submitted from host <user1>, CWD </home/user1/>, 2 Processors
Requested, Requested Resources <rusage[mem=60]>, Specified Hosts <hostA>;
Tue Jan 22 17:01:15: Reserved <1> job slot on host <hostA>;
Tue Jan 22 17:01:15: 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)

- 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

```

Viewing Resource Reservation Information

The following example displays only `licenseZ` configuration:

```
bresources -s licenseZ
RESOURCE          METHOD
licenseZ          PER_SLOT
```

Advance Reservation

Contents

- ◆ [Understanding Advance Reservations](#) on page 454
- ◆ [Configure Advance Reservation](#) on page 456
- ◆ [Using Advance Reservation](#) on page 458

Understanding 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.

Configure Advance Reservation

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.

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          = dayPolicy
USERS         = user1 user2    # optional
HOSTS        = hostA hostB    # optional
TIME_WINDOW  = 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          = nightPolicy
USERS         = ugroup1 ~user1
HOSTS         = hgroup1 ~hostB
TIME_WINDOW  = 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: `dayPolicy`
 Users: `user1`
 Hosts: `hostA`
 Time Window: `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: `dayPolicy`
 Users: `user3`
 Hosts: `hostA`
 Time Window: `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.**

USER_ADVANCE_RESERVATION is obsolete (`lsb.params`)

`USER_ADVANCE_RESERVATION` in `lsb.params` is obsolete in LSF Version 7. Use the `ResourceReservation` section configuration in `lsb.resources` to configure advance reservation policies for your cluster.

Using Advance Reservation

Advance reservation commands

Use the following commands to work with advance reservations:

<code>brsvadd</code>	Add a reservation
<code>brsvdel</code>	Delete a reservation
<code>brsvmod</code>	Modify a reservation
<code>brsvs</code>	View reservations

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

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 described in Chapter 19, “[Specifying Resource Requirements](#)”. 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`).

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 8:0
Reservation "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 8:0
Reservation "user1#1" is created
```

The following command creates a one-time advance reservation that reserves 12 slots on `hostA` 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  
2004:01:31:06:00  
Reservation 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 "user2#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
```

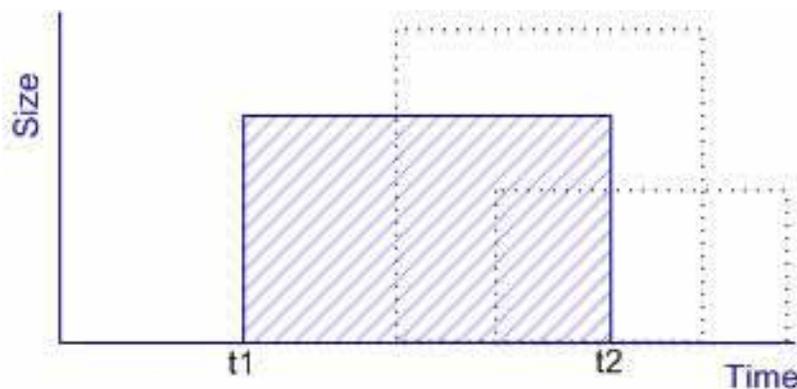
Administrators and root can modify any reservations. Users listed in the `ResourceReservation` section of `lsb.resources`, can only modify reservations they created themselves.

Using 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 t_1 and t_2 , 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.

Extending 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

Adding 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
```

Using Advance Reservation

```
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
```

Removing 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     1/24/12/2-1/24/13/0
                               hostB:1/2
```

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
```

Modifying 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

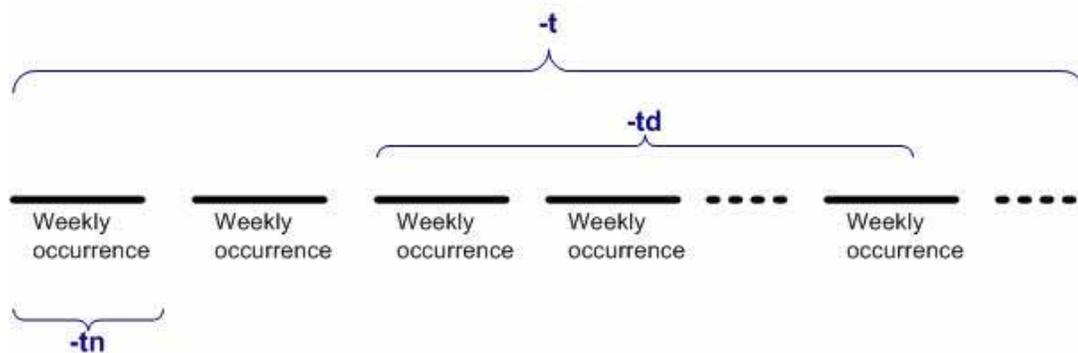
Disable specified occurrences for recurring reservations

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"
```

Using Advance Reservation

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.

		Disable	Modification							
			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
		Specified	Yes	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 in the scenario; otherwise, "No" is marked. For example, all modifications are acceptable in the case that the advance reservation is inactive.

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

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#0
Reservation 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 0/2048  hostA:0/1024  3:0:0-3:3:0 *
                               hostB:0/1024
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 *

Show a weekly planner

- 1 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.hosts`).

```
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    3:0:0-3:3:0 *
           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:Min	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

Advance Reservation

...							
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:Min	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

Using Advance Reservation

- 2 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      12/28/14/30-12/28/15/30
                               hostB:0/1
```

HOST: hostA (MAX = 2)

Week: 12/23/2007 - 12/29/2007

Hour:Min	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:Min	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

- 1 Use the `-l` option of `brsvs` to show each advance reservation in long format. The rows that follow the reservation information show the type of reservation (open or closed) and the status of any job associated with the specified reservation (EXIT, PEND, RUN, or SUSP) as well as:

- ◆ The status of the reservation
- ◆ Resource requirements
- ◆ Disabled time duration
- ◆ Time when the next instance of recurring reservation is inactive

```
brsvs -l
RSVID      TYPE      USER      NCPUS      RSV_HOSTS      TIME_WINDOW
user1#5    user      user1      0/1        hostA:0/1      10:00-12:00 *
```

```

Reservation Status: Inactive
Reservation disabled for these dates:
    Fri Feb 15 2008
    Wed Feb 20 2008 - Mon Feb 25 2008
Next Active Period:
    Sat Feb 16 10:00:00 2008 - Sat Feb 16 12:00:00 2008
Creator: user1
Reservation Type: CLOSED

```

RSVID	TYPE	USER	NCPUS	RSV_HOSTS	TIME_WINDOW
user1#6	user	user1	0/1	hostA:0/1	10:00-13:00 *

```

Reservation Status: Active
Creator: user1
Reservation Type: CLOSED

```

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: 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
```

Using Advance Reservation

- ◆ 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 advanced reservations that are:

- accounted on advanced reservation IDs user1#1,
- accounted on advanced 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 advanced reservations that are:

- accounted on all advanced reservation IDs:
- accounted on advanced 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 `hostA`. When all 1024 slots on `hostA` are used by jobs referencing `user2#0`, `hostA` 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

Prerequisites: 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.

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 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.

Forcing 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 7 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.

Dispatch and Run Windows

Contents

- ◆ [Dispatch and Run Windows](#) on page 477
- ◆ [Run Windows](#) on page 477
- ◆ [Dispatch Windows](#) on page 478

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 `lsb.hosts`. Dispatch and run windows can be defined in `lsb.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 `lsb.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.

For more information about the syntax of time windows, see [Specifying Time Windows](#) on page 285.

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 dispatch windows

Dispatch windows can be defined for both queues and hosts. The default is no restriction, or always open.

Configure host dispatch windows

- 1 To configure dispatch windows for a host, set `DISPATCH_WINDOW` in `lsb.hosts` 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 `lsb.queues` and specify one or more time windows. If no queue dispatch window is configured, the window is always open.

Display queue dispatch windows

- 1 Use `bqueues -l` to display queue dispatch windows.

Display host dispatch windows

- 1 Use `bhosts -l` to display host dispatch windows.

Dispatch Windows

Job Dependencies

Contents

- ◆ [Job Dependency Terminology](#) on page 481
- ◆ [Job Dependency Scheduling](#) on page 482
- ◆ [Dependency Conditions](#) on page 484
- ◆ [View Job Dependencies](#) on page 486

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.

Specify a job dependency

-
- 1 To specify job dependencies, use `bsub -w` to specify a dependency expression for the job.
-

Syntax

```
bsub -w 'dependency_expression'
```

The dependency expression is a logical expression composed of one or more dependency conditions. For syntax of individual dependency conditions, see [Dependency Conditions](#) on page 484.

- ◆ 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 `jobA*` as the job name, it specifies jobs named `jobA`, `jobA1`, `jobA_test`, `jobA.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.

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

Job arrays

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 `bjdepinfo` 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
```

View Job Dependencies

29

Job Priorities

Contents

- ◆ [User-Assigned Job Priority](#) on page 490
- ◆ [Automatic Job Priority Escalation](#) on page 492
- ◆ [Absolute Job Priority Scheduling](#) on page 493

User-Assigned Job Priority

User-assigned job priority provides controls that allow users to order their jobs 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 `btop` and `bbot` commands move jobs relative to other jobs of the same priority. These commands do not change job priority.

In this section

- ◆ [Configure job priority](#) on page 490
- ◆ [Specify job priority](#) on page 490
- ◆ [View job priority information](#) on page 491

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.
- 2 Use `bparams -l` to display the value of `MAX_USER_PRIORITY`.

Syntax

```
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.

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`.

Syntax

```
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 (see [User-Assigned Job Priority](#) on page 490) 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`.
User-assigned job priority must also be configured,
- 2 Use `bparams -l` to display the values of `JOB_PRIORITY_OVER_TIME`.

Syntax

```
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.

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	The fairshare factor automatically adjusts the APS value based on dynamic user priority. FAIRSHARE must be defined in the queue. The FS factor is ignored for non-fairshare queues. The FS factor is influenced by the following fairshare parameters in lsb.params: <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,max</code> , the min of <code>bsub -n min</code> , or the value of <code>PROCLIMIT</code> in <code>lsb.queues</code> .
	MEM	Total real memory requested (in MB). Memory requests appearing to the right of a <code> </code> symbol in a usage string are ignored in the APS calculation.
	SWAP	Total swap space requested (in MB). As with MEM, swap space requests appearing to the right of a <code> </code> symbol in a usage string are ignored.

Factors	Subfactors	Metric
WORK (job attributes)	JRIORITY	<p>The job priority specified by:</p> <ul style="list-style-type: none"> ◆ Default specified by <code>MAX_USER_PRIORITY</code> in <code>lsb.params</code> ◆ Users with <code>bsub -sp</code> or <code>bmod -sp</code> ◆ Automatic priority escalation with <code>JOB_PRIORITY_OVER_TIME</code> in <code>lsb.params</code>
	QRIORITY	The priority of the submission queue.
ADMIN		<p>Administrators use <code>bmod -aps</code> 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.</p> <p>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 LSF gets the job information for each factor

Factor or subfactor	Gets job information from...
MEM	<p>The value for jobs submitted with <code>-R "rusage[mem]"</code></p> <p>For compound resource requirements submitted with <code>-R "n1*{rusage[mem1]} + n2*{rusage[mem2]}"</code> the value of MEM depends on whether resources are reserved per slot.</p> <ul style="list-style-type: none"> ◆ If <code>RESOURCE_RESERVED_PER_SLOT=N</code>, then $MEM = mem1 + mem2$ ◆ If <code>RESOURCE_RESERVED_PER_SLOT=Y</code>, then $MEM = n1 * mem1 + n2 * mem2$
SWAP	<p>The value for jobs submitted with <code>-R "rusage[swp]"</code></p> <p>For compound resource requirements, SWAP is determined in the same manner as MEM.</p>
PROC	The value of n for jobs submitted with <code>bsub -n (min, max)</code> , or the value of <code>PROCLIMIT</code> in <code>lsb.queues</code>
JRIORITY	The dynamic priority of the job, updated every scheduling cycle and escalated by interval defined in <code>JOB_PRIORITY_OVER_TIME</code> defined in <code>lsb.params</code>
QRIORITY	The priority of the job submission queue
FS	The fairshare priority value of the submission user

Enable absolute priority scheduling

Configure `APS_PRIORITY` in an absolute priority queue in `lsb.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]]
```

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. Administrators can use `bmod` to manually override the calculated APS value.

Run `bmod -apn job_ID` to undo the previous `bmod -aps` setting.

Assign a static system priority and ADMIN factor value

Administrators can use using `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

Administrators can 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 Use `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

Use `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>

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 13 15:15:26: Submitted from host <HostB>, to Queue <normal>, CWD </scratch/user1>;
 Tue Feb 13 15:15:40: Parameters of Job are changed:
 Absolute Priority Scheduling factor string changed to : system=10;
 Tue Feb 13 15:15:48: Parameters of Job are changed:
 Absolute Priority Scheduling factor string changed to : admin=20;
 Tue Feb 13 15:15:58: Parameters of Job are changed:
 Absolute Priority Scheduling factor string deleted;
 Summary of time in seconds spent in various states by Tue Feb 13 15:16:02

PEND	PSUSP	RUN	USUSP	SSUSP	UNKWN	TOTAL
36	0	0	0	0	0	36

Configure APS across multiple queues

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.

Absolute Job Priority Scheduling

Example 1

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:

$$\text{APS_PRIORITY} = 1 * (1 * \text{job_priority} + 10 * \text{queue_priority})$$

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`.

Example 2

Extending example 1, you 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

$$\text{APS_PRIORITY} = 1 * (1 * \text{job_priority} + 10 * \text{queue_priority}) + 100 * \text{user_priority}$$

Example 3

Extending example 2, you now to add swap space to the APS value calculation. The APS configuration changes to:

$$\text{APS_PRIORITY} = \text{WEIGHT} [[\text{JPRIORITY}, 1] [\text{QPRIORITY}, 10] [\text{FS}, 100] [\text{SWAP}, -10]]$$

And the APS value is now calculated as

$$\text{APS_PRIORITY} = 1 * (1 * \text{job_priority} + 10 * \text{queue_priority}) + 100 * \text{user_priority} + 1 * (-10 * \text{SWAP})$$

View pending job order by the APS value

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, $\text{APS} = 10 * 20 + 1 * (50 + 220 * 5 / 10) = 360$

For job 12, $\text{APS} = 10 * 30 + 1 * (50 + 10 * 5 / 10) = 355$

For job 4, $\text{APS} = 10 * 20 + 1 * (50 + 40 * 5 / 10) = 270$

View APS configuration for a queue

Run `bqueues -l` to see the current APS information for a queue:

`bqueues -l normal`

Absolute Job Priority Scheduling

QUEUE: normal

-- No description provided. This is the default queue.

PARAMETERS/STATISTICS

PRIO	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

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

Feature interactions

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 SLA scheduling.
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 (<code>bstop</code>) jobs and migrated jobs (<code>bmig</code>) are always scheduled before pending jobs. For migrated jobs, LSF keeps the existing job priority information. If <code>LSB_REQUEUE_TO_BOTTOM</code> and <code>LSB_MIG2PEND</code> are configured in <code>lsf.conf</code> , the migrated jobs keep their APS information. When <code>LSB_REQUEUE_TO_BOTTOM</code> and <code>LSB_MIG2PEND</code> 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 <code>brequeue</code> , not <code>bmig</code> .
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: <ul style="list-style-type: none"> ◆ 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
JPRIORITY	<p>Use the value inherited from the original job. If the automatic job priority escalation is configured, the dynamic value is calculated as described in Automatic Job Priority Escalation on page 492.</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

Contents

- ◆ [About Job Requeue](#) on page 504
- ◆ [Automatic Job Requeue](#) on page 505
- ◆ [Job-level automatic requeue](#) on page 507
- ◆ [Reverse Requeue](#) on page 508
- ◆ [Exclusive Job Requeue](#) on page 509
- ◆ [User-Specified Job Requeue](#) on page 510
- ◆ [Automatic Job Rerun](#) on page 511

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 `lsb.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 (`lsb.params`), in the queue definition (`lsb.queues`), or in an application profile (`lsb.applications`). Specify an integer greater than zero (0). `MAX_JOB_REQUEUE` in `lsb.applications` overrides `lsb.queues`, and `lsb.queues` overrides `lsb.params` configuration. Specifying a job-level exit value using `bsub -Q` overrides all `MAX_JOB_REQUEUE` settings.

Automatic Job Requeue

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.

Viewing the requeue retry limit

- 1 Run `bjobs -l` to display the job exit code and reason if the job requeue limit is exceeded.
- 2 Run `bhist -l` to display the exit code and reason for finished jobs if the job requeue limit is exceeded.

How job requeue retry limit is recovered

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 `lsb.events`.

Job-level automatic requeue

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).

Define an exit code as `EXCLUDE(exit_code)` to enable exclusive job requeue. Exclusive job requeue does not work for parallel jobs.

If `mbatchd` 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.

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 jobs

Job forwarding model For jobs sent to a remote cluster, arguments of `bsub -Q` take effect on remote clusters.

Lease model The arguments of `bsub -Q` apply to jobs running on remote leased hosts as if they are running on local hosts.

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.

Configure reverse requeue

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 `mbatchd` 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 `mbatchd` was restarted, the job could be dispatched to the same host again after `mbatchd` 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 `hostA`, `hostB` or `hostC`.

If a job running on `hostA` exits with value 30 and is requeued, it can be dispatched to `hostA`, `hostB`, or `hostC`. However, if a job running on `hostA` exits with value 20 and is requeued, it can only be dispatched to `hostB` or `hostC`.

If the job runs on `hostB` and exits with a value of 20 again, it can only be dispatched on `hostC`. Finally, if the job runs on `hostC` and exits with a value of 20, it cannot be dispatched to any of the hosts, so it is pending forever.

User-Specified Job Requeue

You can use `brequeue` to kill a job and requeue it. When the job is requeued, it is assigned the PENDING status and the job's new position in the queue is after other jobs of the same priority.

Requeue a job

-
- 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 PENDING 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.
-

Job Checkpoint, Restart, and Migration

Job checkpoint and restart optimizes resource usage by enabling a non-interactive job to restart on a new host from the point at which the job stopped—checkpointed jobs do not have to restart from the beginning. Job migration facilitates load balancing by enabling users to move a job from one host to another while taking advantage of job checkpoint and restart functionality.

Contents

- ◆ [Checkpoint and restart options](#) on page 514
- ◆ [Checkpoint directory and files](#) on page 514
- ◆ [Checkpoint and restart executables](#) on page 516
- ◆ [Job restart](#) on page 516
- ◆ [Job migration](#) on page 517

Checkpoint and restart options

You can implement job checkpoint and restart at one of the following levels.

- ◆ Kernel level—provided by your operating system, enabled by default
- ◆ User level—provided by special LSF libraries that you link to your application object files
- ◆ Application level—provided by your site-specific applications and supported by LSF through the use of application-specific `echkpnt` and `erestart` executables

NOTE: For a detailed description of the job checkpoint and restart feature and how to configure it, see the *Platform LSF Configuration Reference*.

Checkpoint directory and files

The job checkpoint and restart feature requires that a job be made checkpointable at the job, application profile, or queue level. LSF users can make a job checkpointable by submitting the job using `bsub -k` and specifying a checkpoint directory, and optional checkpoint period, initial checkpoint period, and checkpoint method. Administrators can make all jobs in a queue or an application profile checkpointable by specifying a checkpoint directory for the queue or application.

Requirements

The following requirements apply to a checkpoint directory specified at the queue or application profile level:

- ◆ The specified checkpoint directory must already exist. LSF does 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.

Behavior

Specifying a checkpoint directory at the queue level or in an application profile enables checkpointing.

- ◆ All jobs submitted to the queue or application profile 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.

NOTE: LSF does not delete the checkpoint files; you must perform file maintenance manually.

- ◆ If the administrator specifies a checkpoint period, in minutes, LSF creates a checkpoint file every `chkpnt_period` during job execution.

- ◆ If the administrator specifies an initial checkpoint period in an application profile, in minutes, the first checkpoint does not happen until the initial period has elapsed. LSF then creates a checkpoint file every *chkpnt_period* after the initial checkpoint period, during job execution.
- ◆ If a user specifies a checkpoint directory, initial checkpoint period, checkpoint method or checkpoint period at the job level with `bsub -k`, or modifies the job with `bmod`, the job-level values override the queue-level and application profile values.

The `brestart` command restarts checkpointed jobs that have stopped running.

Precedence of checkpointing options

If checkpoint-related configuration is specified in both the queue and an application profile, the 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.

Checkpointing MultiCluster jobs

To enable checkpointing of MultiCluster jobs, define a checkpoint directory in both the send-jobs and receive-jobs queues (`CHKPNT` in `lsb.queues`), or in an application profile (`CHKPNT_DIR`, `CHKPNT_PERIOD`, `CHKPNT_INITPERIOD`, `CHKPNT_METHOD` in `lsb.applications`) of both submission cluster and execution cluster. LSF uses the directory specified in the execution cluster.

Checkpointing is not supported if a job runs on a leased host.

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
...
```

Checkpoint and restart executables

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
```

NOTE: For a detailed description of the commands used with the job checkpoint and restart feature, see the *Platform LSF Configuration Reference*.

Checkpoint and restart executables

LSF controls checkpointing and restart by means of interfaces named `echkpnt` 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, `echkpnt` sends checkpoint instructions to an executable named `echkpnt.default`.

For application-level job checkpoint and restart, you can specify customized checkpoint and restart executables for each application that you use. The optional parameter `LSB_ECHKPNT_METHOD` specifies a checkpoint executable used for all jobs in the cluster. An LSF user can override this value when submitting a job.

NOTE: For a detailed description of how to write and configure application-level checkpoint and restart executables, see the *Platform LSF Configuration Reference*.

Job restart

LSF can restart a checkpointed job on a host other than the original execution host using the information saved in the checkpoint file to recreate the execution environment. Only jobs that have been checkpointed successfully can be restarted from a checkpoint file. When a job restarts, LSF performs the following actions:

- 1 LSF resubmits the job to its original queue as a new job and assigns a new job ID.
- 2 When a suitable host becomes available, LSF dispatches the job.
- 3 LSF recreates the execution environment from the checkpoint file.
- 4 LSF restarts the job from its last checkpoint. You can restart a job manually from the command line using `brestart`, automatically through configuration, or by migrating the job to a different host using `bmig`.

Requirements

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:

- ◆ 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

Job migration

Job migration is 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 manually on demand (`bmig`) or automatically. To initiate job migration automatically, you can configure a migration threshold at job submission, or at the host, queue, or in an application profile.

NOTE: For a detailed description of the job migration feature and how to configure it, see the *Platform LSF Configuration Reference*.

Manual job migration

The `bmig` command migrates checkpointable or rerunnable jobs on demand. Jobs can be manually migrated by the job owner, queue administrator, and LSF administrator.

For example, to migrate a job with job ID 123 to the first available host:

```
bmig 123
Job <123> is being migrated
```

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. Specifying a migration threshold at job submission (`bsub -mig`) or configuring an application profile-level, queue-level or host-level migration threshold allows the job to progress and reduces the load on the host. You can use `bmig` at any time to override a configured migration threshold, or `bmod -mig` to change a job-level migration threshold.

For example, at the queue level, in `lsb.queues`:

```
Begin Queue
...
MIG=30          # Migration threshold set to 30 mins
DESCRIPTION=Migrate suspended jobs after 30 mins
...
End Queue
```

Job migration

At the host level, in `lsb.hosts`:

```
Begin Host
  HOST_NAME    r1m    pg    MIG # Keywords
  ...
  hostA        5.0    18    30
  ...
End Host
```

For example, in an application profile, in `lsb.applications`:

```
Begin Application
  ...
  MIG=30          # Migration threshold set to 30 mins
  DESCRIPTION=Migrate suspended jobs after 30 mins
  ...
End Application
```

If you want to requeue migrated jobs instead of restarting or rerunning them, you can define the following parameters in `lsf.conf`:

- ◆ `LSB_MIG2PEND=1` requeues a job with the original submission time and priority
- ◆ `LSB_REQUEUE_TO_BOTTOM=1` requeues a job at the bottom of the queue, regardless of the submission time and priority

Chunk Job Dispatch

Contents

- ◆ [About Job Chunking](#) on page 519
- ◆ [Configure Chunk Job Dispatch](#) on page 520
- ◆ [Submitting and Controlling Chunk Jobs](#) on page 522

About 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.

Chunk job candidates

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 underutilize 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.

Configure Chunk Job Dispatch

CHUNK_JOB_SIZE (lsb.queues)

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 `lsb.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
```

Postrequisites: After adding `CHUNK_JOB_SIZE` to `lsb.queues`, use `badmin reconfig` to reconfigure your cluster.

Chunk jobs and job throughput

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.

CHUNK_JOB_SIZE (lsb.applications)

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 `lsb.applications`.
Specify `CHUNK_JOB_SIZE=1` to disable job chunking for the application. This value overrides chunk job dispatch configured in the queue.

Postrequisites: After adding `CHUNK_JOB_SIZE` to `lsb.applications`, use `badmin reconfig` to reconfigure your cluster.

CHUNK_JOB_DURATION (lsb.params)

If `CHUNK_JOB_DURATION` is defined in the file `lsb.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 `lsb.applications`), or
- ◆ A queue-level CPU limit or run time limit is specified (CPULIMIT or RUNLIMIT in `lsb.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 `lsb.params`, use `badmin reconfig` to reconfigure your cluster.

By default, `CHUNK_JOB_DURATION` is not enabled.

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 `lsb.queues` or `lsb.applications`). If `CHUNK_JOB_DURATION` is set in `lsb.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 `lsb.queues` or `lsb.applications`). If `CHUNK_JOB_DURATION` is set in `lsb.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 `lsb.applications`)

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).

Submitting and Controlling 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 `sbatchd` 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 `bjobs`. Any jobs in `WAIT` status are included in the count of pending jobs by `bqueues` and `busers`. The `bhosts` 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 `bjobs -l` command does not display a `WAIT` reason in the list of pending jobs.

Controlling 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>bstop</code>)	PEND	Removed from chunk (PSUSP)
	RUN	All jobs in the chunk are suspended (NRUN -1, NSUSP +1)
	USUSP	No change
	WAIT	Removed from chunk (PSUSP)
Kill (<code>bkill</code>)	PEND	Removed from chunk (NJOBS -1, PEND -1)
	RUN	Job finishes, next job in the chunk starts if one exists (NJOBS -1, PEND -1)
	USUSP	Job finishes, next job in the chunk starts if one exists (NJOBS -1, PEND -1, SUSP -1, RUN +1)

Action (Command)	Job State	Effect on Job (State)
	WAIT	Job finishes (NJOBS-1, PEND -1)
Resume (bresume)	USUSP	Entire chunk is resumed (RUN +1, USUSP -1)
Migrate (bmig)	WAIT	Removed from chunk
Switch queue (bswitch)	RUN	Job is removed from the chunk and switched; all other WAIT jobs are requeued to PEND
	WAIT	Only the WAIT job is removed from the chunk and switched, and requeued to PEND
Checkpoint (bchkpnt)	RUN	Job is checkpointed normally
Modify (bmod)	PEND	Removed from the chunk to be scheduled later

Migrating jobs with `bmig` 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.

See Chapter 30, “[Job Requeue and Job Rerun](#)” for more information about rerunnable jobs.

Checkpointing 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.

See Chapter 31, “[Job Checkpoint, Restart, and Migration](#)” for more information about checkpointing jobs.

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

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. 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.

Contents

- ◆ [Create a Job Array](#) on page 525
- ◆ [Handling Input and Output Files](#) on page 527
- ◆ [Job Array Dependencies](#) on page 529
- ◆ [Monitoring Job Arrays](#) on page 529
- ◆ [Controlling Job Arrays](#) on page 531
- ◆ [Requeuing a Job Array](#) on page 534
- ◆ [Job Array Job Slot Limit](#) on page 535

Create a Job Array

A job array is created at job submission time using the `-J` option of `bsub`.

- 1 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>.
```

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]
```

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 can never exceed 1000 jobs.

-
- 1 To make a change to the maximum job array value, 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`.
-

Handling 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.

Methods

- ◆ [Redirecting Standard Input and Output](#) on page 527
- ◆ [Passing Arguments on the Command Line](#) on page 528

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`
-

Redirecting Standard Input and Output

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.

Redirect standard input

- 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 `output.1`, `output.2`, `output.3`, ..., `output.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 `output.123.1`, `output.123.2`, `output.123.3`, ..., `output.123.1000`. The job ID of the job array is 123.

```
bsub -J "myArray[1-1000]" -o "output.%J.%I" myJob
```

Passing 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:

```
input.1, input.2, input.3, ..., input.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 `input.1`,

input.2, input.3, ..., input.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
```

Job Array Dependencies

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.

Set a whole array dependency

- 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
<code>numrun(jobArrayJobId, op num)</code>	Evaluate the number of jobs in RUN state
<code>numpend(jobArrayJobId, op num)</code>	Evaluate the number of jobs in PEND state
<code>numdone(jobArrayJobId, op num)</code>	Evaluate the number of jobs in DONE state
<code>numexit(jobArrayJobId, op num)</code>	Evaluate the number of jobs in EXIT state
<code>numended(jobArrayJobId, op num)</code>	Evaluate the number of jobs in DONE and EXIT state
<code>numhold(jobArrayJobId, op num)</code>	Evaluate the number of jobs in PSUSP state
<code>numstart(jobArrayJobId, op num)</code>	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
```

Monitoring Job Arrays

Use `bjobs` and `bhist` to monitor the current and past status of job arrays.

Individual job status

Display job array status

- 1 To display summary information about the currently running jobs submitted from a job array, use the `-A` option of `bjobs`.

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     3   4     0     0     0     0
```

Display job array dependencies

- 1 To display information for any job dependency information for an array, use the `bjdepinfo` 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
```

Individual job status

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 `bjobs`. 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      hostQ      myArray[8]  Feb 29 12:34
123    user1  PEND  default  hostA      hostQ      myArray[9]  Feb 29 12:34
123    user1  PEND  default  hostA      hostQ      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
```

Specific job status

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
```

Controlling 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`, `bresume`, 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.

Requeuing a Job Array

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. You can requeue:

- ◆ Jobs in `DONE` job state
- ◆ Jobs in `EXIT` job state
- ◆ All jobs regardless of job state in a job array.
- ◆ `EXIT`, `RUN`, `DONE` jobs to `PSUSP` state
- ◆ Jobs in `RUN` job state

`brequeue` is not supported across clusters.

Requeue jobs in `DONE` state

- 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.

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.

Setting a job array job slot limit

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: Submitted from host <hostA>, CWD <${HOME}>;

COUNTERS:
NJOB  PEND  DONE  RUN  EXIT  SSUSP  USUSP  PSUSP
   10    9    0    1    0    0    0    0
```

Running Parallel Jobs

Contents

- ◆ [How LSF Runs Parallel Jobs](#) on page 538
- ◆ [Preparing Your Environment to Submit Parallel Jobs to LSF](#) on page 539
- ◆ [Submitting Parallel Jobs](#) on page 540
- ◆ [Starting Parallel Tasks with LSF Utilities](#) on page 541
- ◆ [Job Slot Limits For Parallel Jobs](#) on page 543
- ◆ [Specifying a Minimum and Maximum Number of Processors](#) on page 544
- ◆ [Specifying a First Execution Host](#) on page 545
- ◆ [Controlling Processor Allocation Across Hosts](#) on page 554
- ◆ [Controlling Job Locality using Compute Units](#) on page 547
- ◆ [Running Parallel Processes on Homogeneous Hosts](#) on page 557
- ◆ [Limiting the Number of Processors Allocated](#) on page 559
- ◆ [Reserving Processors](#) on page 562
- ◆ [Reserving Memory for Pending Parallel Jobs](#) on page 564
- ◆ [Backfill Scheduling: Allowing Jobs to Use Reserved Job Slots](#) on page 565
- ◆ [Parallel Fairshare](#) on page 574
- ◆ [How Deadline Constraint Scheduling Works For Parallel Jobs](#) on page 575
- ◆ [Optimized Preemption of Parallel Jobs](#) on page 576
- ◆ [Processor Binding for Parallel Jobs](#) on page 576
- ◆ [Making Job Allocations Resizable](#) on page 578

How 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 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.

For more information, see [Submitting Parallel Jobs](#) on page 540

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. See [Queue-Level Job Starters](#) on page 628 for more information about job starters.

1 To see if your queue already has a job starter defined, run `bqueues -l`.

Submitting Parallel Jobs

LSF can allocate more than one host or processor to run a job and automatically keeps track of the job status, while a parallel job is running.

Specify the number of processors

When submitting a parallel job that requires multiple processors, you can specify the exact number of processors to use.

- 1 To submit a parallel job, use `bsub -n` and specify the number of processors the job requires.
- 2 To submit jobs based on the number of available job slots instead of the number of processors, use `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`.

For example:

```
bsub -n 4 myjob
```

submits `myjob` as a parallel job. The job is started when 4 job slots are available.

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 processors, however `lshosts` output will continue to show `ncpus` as defined by `EGO_DEFINE_NCPUS` in `lsf.conf`.

Starting Parallel Tasks with 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.

Running parallel tasks with `lsgrun`

The simplest parallel job runs an identical copy of the executable on every host. The `lsgrun` command takes a list of host names and runs the specified task on each host. The `lsgrun -p` command specifies that the task should be run in parallel on each host.

Example

This example submits a job that uses `lsgrun` to run `myjob` on all the selected hosts in parallel:

```
bsub -n 10 'lsgrun -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 `lsrun` in the background to start each component.

Running 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 `lsrun` 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.

See *Using Platform LSF HPC* for more information about using the `blaunch` distributed application framework.

Submitting 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.

Starting Parallel Tasks with LSF Utilities

- ◆ **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.

Specifying 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.

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

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

Specifying a First Execution Host

- ❖ 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]
```

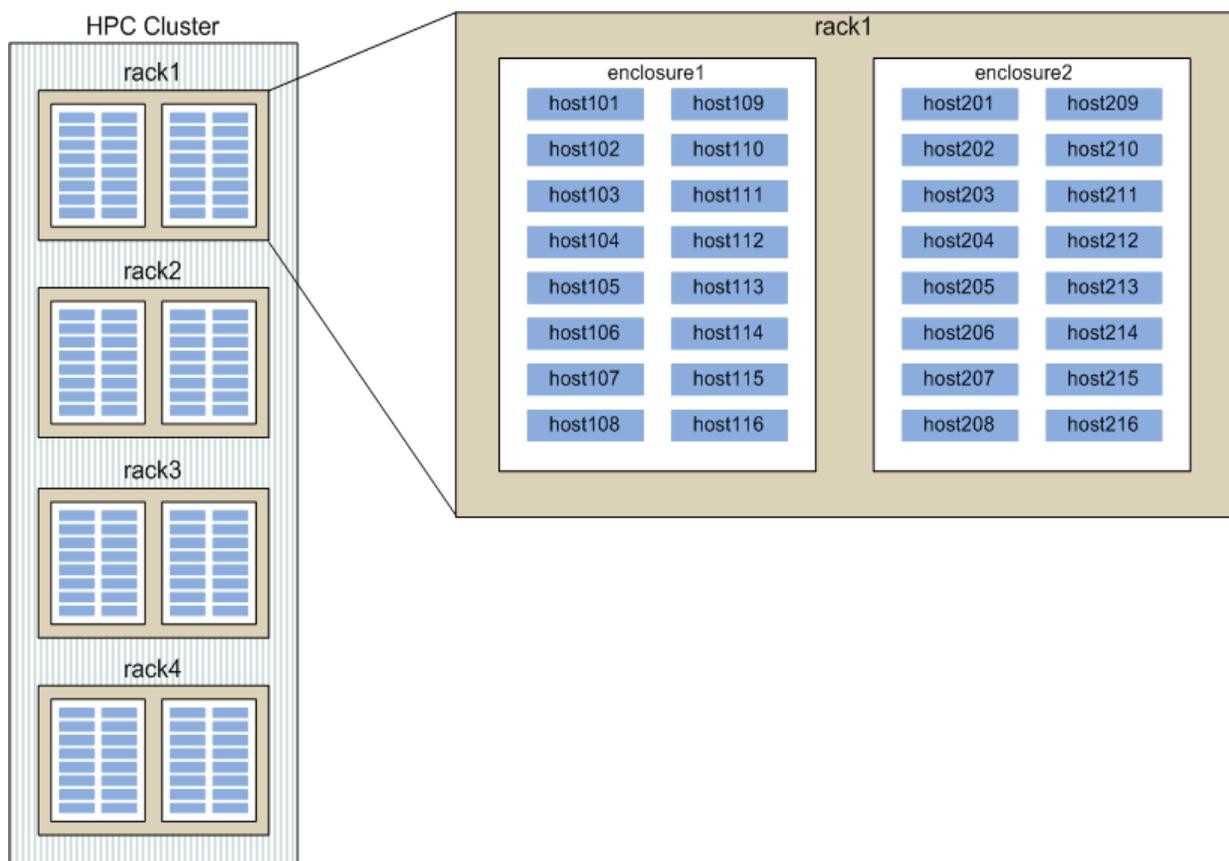
Controlling Job Locality using Compute Units

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.

TIP: 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
```

Controlling Job Locality using Compute Units

! 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):

```
Begin ComputeUnit
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
End ComputeUnit
```

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 `bhosts`, where `HOST_NAME` refers to the compute unit:

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
enclosure1	ok	-	64	34	34	0	0	0
enclosure2	ok	-	64	54	54	0	0	0
enclosure3	ok	-	64	46	46	0	0	0
enclosure4	ok	-	64	44	44	0	0	0
enclosure5	ok	-	64	45	45	0	0	0
enclosure6	ok	-	64	44	44	0	0	0
enclosure7	ok	-	32	0	0	0	0	0
enclosure8	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.

Using compute units

1 `bsub -R "cu[]" -n 64 ./app`

This job is restricted to compute units of the default type `enclosure`. The default `pref=config` applies, with compute units considered in configuration order. The job runs on 30 slots in `enclosure1`, 10 slots in `enclosure2`, 8 slots in `enclosure3`, and 16 slots in `enclosure4` 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 `enclosure` in the order `pref=minavail`. Compute units with the fewest free slots are considered first. The job runs on 10 slots in `enclosure2`, 18 slots in `enclosure3` and 3 slots in `enclosure5` 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 `enclosure` in the order `pref=maxavail`. Compute units with the most free slots are considered first. The job runs on 64 slots in `enclosure8`.

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 `enclosure8` 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
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
```

Controlling Job Locality using Compute Units

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 `enclosure7` and 26 slots in `enclosure8`.

```
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    host1   host
h2    host2   host
...
h50   host50  host
End ComputeUnit
```

Each configured compute unit of default type `host` contains a single host.

Ordering host allocations

Using the compute `host` 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.

Limiting 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 `usablecuslots`, hosts are only considered if they have a minimum number of slots free and usable for this job:

```
1 bsub -n 16 -R "cu[usablecuslots=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[usablecuslots=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.

Controlling 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.

Specifying 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.

Specifying 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]
```

Controlling Processor Allocation Across Hosts

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.

Specifying parallel job locality at the queue level

The queue may also define the locality for parallel jobs using the `RES_REQ` parameter.

Running 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.

Examples

Running 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.

Running 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.

Running 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
```

Running Parallel Processes on Homogeneous Hosts

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 `lsb.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`

Limiting the Number of Processors Allocated

Use the PROCLIMIT parameter in `lsb.queues` or `lsb.applications` to limit the number of processors that can be allocated to a parallel job.

- ◆ [Syntax](#) on page 559
- ◆ [How PROCLIMIT affects submission of parallel jobs](#) on page 559
- ◆ [Changing PROCLIMIT](#) on page 560
- ◆ [MultiCluster](#) on page 560
- ◆ [Resizable jobs](#) on page 560
- ◆ [Automatic queue selection](#) on page 560
- ◆ [Examples](#) on page 561

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:

$1 \leq \text{minimum} \leq \text{default} \leq \text{maximum}$

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 `mbatchd` log file when LSF is started.

Changing 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.

Example

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.

Examples

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 <code>myjob</code> runs on 2 processors.
<code>bsub -n 4 myjob</code>	The job <code>myjob</code> 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 <code>myjob</code> runs on 2 or 3 processors.
<code>bsub -n 2,5 myjob</code>	The job <code>myjob</code> 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 <code>myjob</code> runs on 1 processor.

Minimum and maximum processor limits

PROCLIMIT is specified in `lsb.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
<code>bsub -n 5 myjob</code>	The job <code>myjob</code> runs on 5 processors.
<code>bsub -n 2 myjob</code>	The job <code>myjob</code> is rejected from the queue because the number of processors requested is less than the minimum number of processors configured for the queue (3).
<code>bsub -n 4,5 myjob</code>	The job <code>myjob</code> runs on 4 or 5 processors.
<code>bsub -n 2,6 myjob</code>	The job <code>myjob</code> runs on 3 to 6 processors.
<code>bsub -n 4,9 myjob</code>	The job <code>myjob</code> runs on 4 to 8 processors.
<code>bsub myjob</code>	The default number of processors is equal to the minimum number (3). The job <code>myjob</code> runs on 3 processors.

Minimum, default, and maximum processor limits

PROCLIMIT is specified in `lsb.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
<code>bsub myjob</code>	Because a default number of processors is configured, the job <code>myjob</code> runs on 6 processors.

Reserving 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 `lsb.queues` and specify the reservation time (a job cannot hold any reserved slots after its reservation time expires).

Syntax

```
SLOT_RESERVE=MAX_RESERVE_TIME[n].
```

where *n* is an integer by which to multiply `MBD_SLEEP_TIME`.

`MBD_SLEEP_TIME` is defined in `lsb.params`; the default value is 60 seconds.

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.

Viewing information about reserved job slots

Reserved slots can be displayed with the `bjobs` command. The number of reserved slots can be displayed with the `bqueues`, `bhosts`, `bhpart`, and `busers` commands. Look in the `RSV` column.

Reserving 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.

Configuring memory reservation for pending parallel jobs

Use the `RESOURCE_RESERVE` parameter in `lsb.queues` to reserve host memory for pending jobs, as described in [Memory Reservation for Pending Jobs](#) on page 440.

lsb.queues

- 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: Allowing Jobs to Use Reserved Job Slots

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

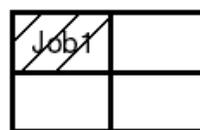
For backfill scheduling, LSF assumes that a job can run until its run limit expires. Backfill scheduling works most efficiently when all the jobs in the cluster have a run limit.

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.

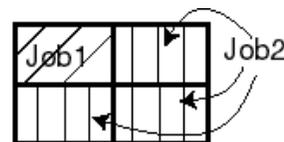
Once the big parallel job has reserved sufficient job slots, LSF calculates the start time of the big job, based on the run limits of the jobs currently running in the reserved slots. LSF cannot backfill if the big job is waiting for a job that has no run limit defined.

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.

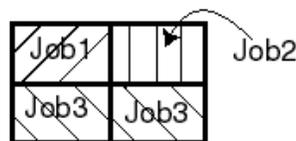
Example



(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.

Backfill Scheduling: Allowing Jobs to Use Reserved Job Slots

- 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 (`job2`) requiring all 4 CPUs is submitted. It cannot start right away because `job1` is using one CPU, so it reserves the remaining 3 processors (figure b).
- 3 At 8:30 am, another parallel job (`job3`) is submitted requiring only two processors and with a run limit of 1 hour. Since `job2` cannot start until 10:00am (when `job1` finishes), its reserved processors can be backfilled by `job3` (figure c). Therefore `job3` can complete before `job2`'s start time, making use of the idle processors.
- 4 `job3` finishes at 9:30am and `job1` at 10:00am, allowing `job2` to start shortly after 10:00am. In this example, if `job3`'s run limit was 2 hours, it would not be able to backfill `job2`'s reserved slots, and would have to run after `job2` finishes.

Limitations

- ◆ A job does not have an estimated start time immediately after `mbatchd` 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 allocation 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 allocation request is 16 hours.

Configuring 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 `lsb.queues`.
- 2 Specify `Y` to enable backfilling. To disable backfilling, specify `N` or blank space.

Example

```
BACKFILL=Y
```

Enforcing 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 `lsb.queues` or `lsb.applications`, 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 `lsb.applications`. 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 `lsb.queues`) or application level (`RUNLIMIT` in `lsb.applications`).

View information about job start time

1 Use `bjobs -l` to view the estimated start time of a job.

Using 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

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
```

```
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`.

Backfill Scheduling: Allowing Jobs to Use Reserved Job Slots

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.

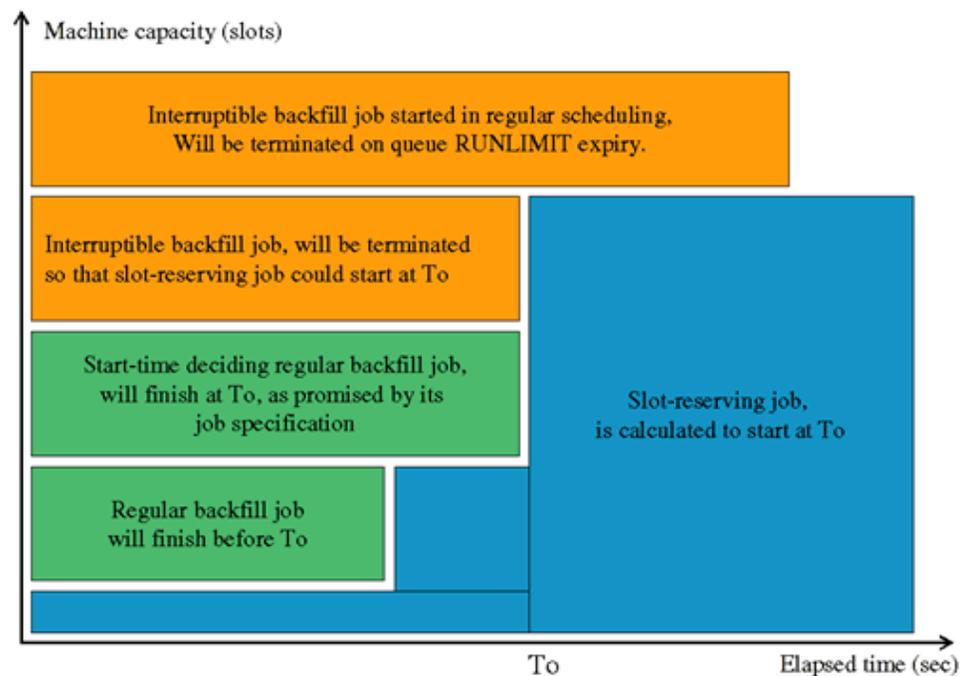
Using interruptible backfill

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-reservation 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 LSF 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: Submitted from host <hostA>, CWD <${HOME}/H
    PC/jobs>;

RUNLIMIT
59.5 min of hostA
Mon Nov 21 11:49:26: Started on <hostA>, Execution Home </home
    /user1>, Execution CWD </home/user1/HPC/jobs>;
```

- 2 Use `bhist` 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>.
bhist displays the job-level run limit:
```

```
bhist -l 135
Job <135>, User <user1>, Project <default>, Command <myjob>
Mon Nov 21 11:49:22: Submitted from host <hostA>, to Queue <la
    zy>, CWD <${HOME}/HPC/jobs>;

RUNLIMIT
180.0 min of hostA
Mon Nov 21 11:49:26: Dispatched to <hostA>;
Mon Nov 21 11:49:26: Starting (Pid 2746);
Mon Nov 21 11:49:27: Interruptible backfill runtime limit is 59.5 minutes;
```

Mon Nov 21 11:49:27: Running with execution home </home/user1>, Execution CWD

...

Displaying available slots for backfill jobs

The `bslots` 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 `bslots` 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, `bslots` displays all available slots, and the available run time for those slots. When no reserved slots are available for backfill, `bslots` displays "No reserved slots available."

The backfill window calculation is based on the snapshot information (current running jobs, slot reservations, advance reservations) obtained from `mbatchd`.

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.

`bslots -R` only supports the `select` 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 ... ..
```

Submitting backfill jobs according to available slots

- 1 Use `bslots` 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 `bslots`.

Submitting a job according to the backfill slot availability shown by `bslots` 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_CPUs* 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 as indicated in [Fairshare Scheduling](#) on page 335.
- 2 To enable parallel fairshare, set the parameter `LSB_NCPU_ENFORCE=1` in `lsf.conf`.
- 3 To make your changes take effect, use 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

For information about deadline constraint scheduling, see [Using Deadline Constraint Scheduling](#) on page 291. 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 `lsb.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 `lsb.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

See also [Processor binding for LSF job processes](#) on page 680.

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 multihost 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 multihost 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.

Making Job Allocations Resizable

By default, if a job specifies a minimum and maximum slot request (`bsub -n min, max`), LSF makes a onetime allocation and schedules the job. You can make jobs *resizable* by submitting them an application profile configured for resizable jobs. You can use the `-ar` option to submit *autoresizable* jobs, where LSF dispatches jobs as long as 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.

For detailed information about the resizable job feature and how to configure it, see the *Platform LSF Configuration Guide*.

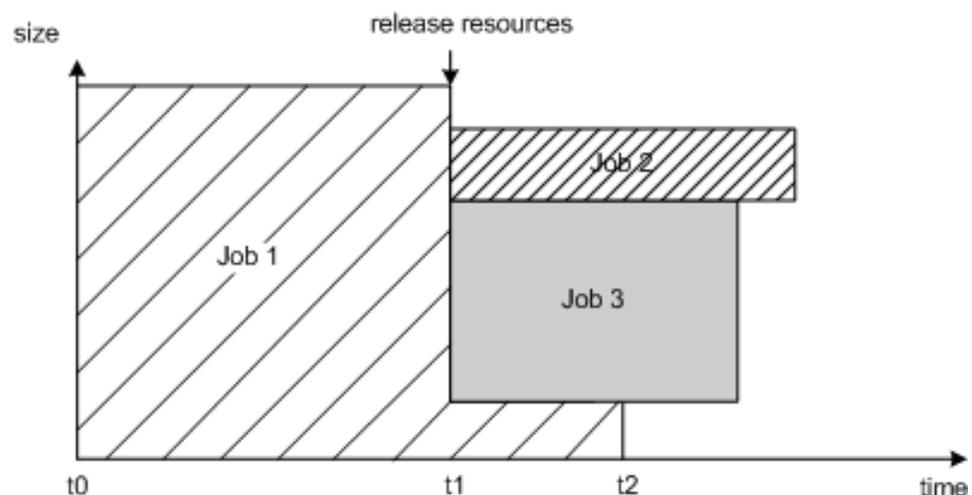
Job scheduling and dispatch

The `JOB_ACCEPT_INTERVAL` parameter in `lsb.params` or `lsb.queue` 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.

Release job allocations

Some parallel jobs must release idle resources before the job completes. For instance, an embarrassingly parallel application typically has a "long tail" issue. In this case, most of short running tasks have already finished leaving many nodes idle. But there are still a few long running tasks left which occupy a small number of nodes. Ideally, the application should release these idle resources and let other jobs make use of them.

Traditional HPC applications also have this requirement, where a long running parallel portion finishes and job waits for a serial job to run to complete file staging. In this case, users would like to release every node except the first one for serial portion of the job run. The following figure illustrates this process:



Release idle resources from the application:

To release resources from running job, the job must be submitted to an application profile configured as resizable. Run `bresize release` to release allocated resources from a running resizable job:

- ◆ Release all slots except one slot from the first execution node
- ◆ Release all hosts except the first execution node
- ◆ Release a list of hosts and different slots for each explicitly

Only the job owner, cluster administrators, queue administrators, user group administrators, and root are allowed to release resources.

Specify a resize notification command

You can also use `bresize release` to specify a resize notification command to be invoked on the first execution host of the job. The resize notification command only applies to the release request and overrides the corresponding resize notification parameters defined in either the application profile (`RESIZE_NOTIFY_CMD` in `lsb.applications`) or at job level (`bsub -rnc resize_notification_cmd`).

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.

By default, if the job has an active pending allocation request, you cannot release the resources. You must run `bresize release -c` to cancel the active pending allocation request and release the resources.

Cancel an active pending request

If a job still has an active pending allocation request, but you do not want to allocate more resources to the job, use `bresize cancel` to cancel the allocation request.

Only the job owner, cluster administrators, queue administrators, user group administrators, and root are allowed to cancel pending resource allocation requests.

Feature interactions

- Chunk Jobs** Because candidate jobs for the chunk job feature are short-running sequential jobs, the resizable job feature does not support job chunking:
 - ◆ Autoresizable jobs in a chunk queue or application profile cannot be chunked together
 - ◆ `bresize` commands to resize job allocations do not apply to running chunk job members
- brequeue** Jobs requeued with `brequeue` start from the beginning. After requeue, LSF restores the original allocation request for the job.
- bswitch** `bswitch` can switch resizable jobs between queues regardless of job state (including job is 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.
- User group administrators** User group administrators are allowed to issue `bresize` commands to release a part of resources from job allocation (`bresize release`) or cancel active pending allocation request (`bresize cancel`).

Making Job Allocations Resizable

- Requeue exit values** 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.
- Automatic job rerun** 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.

Submitting Jobs Using JSDL

Contents

- ◆ [Why Use JSDL?](#) on page 581
- ◆ [Using JSDL Files with LSF](#) on page 581
- ◆ [Collecting resource values using elim.jsdl](#) on page 590

Why Use 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>.

Using 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 (`jsdl.xsd`), the POSIX extension (`jsdl-posix.xsd`), and the LSF extension (`jsdl-lsf.xsd`) are located in the `LSF_LIBDIR` 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 1: Supported JSDL and POSIX extension elements

Element	bsub Option	Description	Example
Job Structure Elements			
JobDefinition	N/A	Root element of the JSDL document. Contains the mandatory child element JobDescription.	<pre><JobDefinition> <JobDescription> ... </JobDescription> </JobDefinition></pre>
JobDescription	-P	High-level container element that holds more specific job description elements.	
Job Identity Elements			
JobName	-J	String used to name the job.	<code><jsd1:JobName>myjob</jsdl:JobName></code>
JobProject	-P	String that specifies the project to which the job belongs.	<code><jsd1:JobProject>myproject</jsdl:JobProject></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 <code>lsb.applications</code> .	<pre><jsd1:Application> <jsd1:ApplicationName>ApplicationX </jsdl:ApplicationName> </jsdl:Application></pre>
ApplicationVersion	-app	String that defines the version of the application defined in <code>lsb.applications</code> .	<pre><jsd1:Application> <jsd1:ApplicationName> ApplicationX</jsdl:ApplicationName> <jsd1:ApplicationVersion>5.5 </jsdl:ApplicationVersion> ... </jsdl:Application></pre>
Resource Elements			
CandidateHosts	-m	Complex type element that specifies the set of named hosts that can be selected to run the job.	<pre><jsd1:CandidateHosts> <jsd1:HostName>host1</jsdl:HostName> <jsd1:HostName>host2</jsdl:HostName> </jsdl:CandidateHosts></pre>
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.	<code><jsd1:ExclusiveExecution>>true </jsd1:ExclusiveExecution></code>
OperatingSystemName	-R	A token type that contains the operating system name. LSF uses the external resource "osname."	<code><jsd1:OperatingSystemName>LINUX </jsd1:OperatingSystemName></code>
OperatingSystemVersion	-R	A token type that contains the operating system version. LSF uses the external resource "osver."	<code><jsd1:OperatingSystemVersion>5.7 </jsd1:OperatingSystemVersion></code>
CPUArchitectureName	-R	Token that specifies the CPU architecture required by the job in the execution environment. LSF uses the external resource "cpuarch."	<code><jsd1:CPUArchitectureName>sparc </jsd1:CPUArchitectureName></code>
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."	<code><jsd1:IndividualCPUSpeed> <jsd1:LowerBoundedRange>1073741824.0 </jsd1:LowerBoundedRange> </jsd1:IndividualCPUSpeed></code>
IndividualCPUCount	-n	Range value that specifies the number of CPUs for each resource.	<code><jsd1:IndividualCPUCount> <jsd1:exact>2.0</jsd1:exact> </jsd1:IndividualCPUCount></code>
IndividualPhysicalMemory	-R	Range value that specifies the amount of physical memory required on each resource, in bytes.	<code><jsd1:IndividualPhysicalMemory> <jsd1:LowerBoundedRange>1073741824.0 </jsd1:LowerBoundedRange> </jsd1:IndividualPhysicalMemory></code>
IndividualVirtualMemory	-R	Range value that specifies the amount of virtual memory required for each resource, in bytes.	<code><jsd1:IndividualVirtualMemory> <jsd1:LowerBoundedRange>1073741824.0 </jsd1:LowerBoundedRange> </jsd1:IndividualVirtualMemory></code>

Using JSDL Files with LSF

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."	<code><jsd1:IndividualNetworkBandwidth> <jsd1:LowerBoundedRange>104857600.0 </jsdl:LowerBoundedRange> </jsdl:IndividualNetworkBandwidth></code>
TotalCPUCount	-n	Range value that specifies the total number of CPUs required for the job.	<code><jsd1:TotalCPUCount><jsd1:exact>2.0 </jsdl:exact></jsdl:TotalCPUCount></code>
TotalPhysicalMemory	-R	Range value that specifies the required amount of physical memory for all resources for the job, in bytes.	<code><jsd1:TotalPhysicalMemory> <jsd1:LowerBoundedRange> 10737418240.0 </jsdl:LowerBoundedRange> </jsdl:TotalPhysicalMemory></code>
TotalVirtualMemory	-R	Range value that specifies the required amount of virtual memory for the job, in bytes.	<code><jsd1:TotalVirtualMemory> <jsd1:LowerBoundedRange>1073741824.0 </jsdl:LowerBoundedRange> </jsdl:TotalVirtualMemory></code>
TotalResourceCount	-n	Range value that specifies the total number of resources required by the job.	<code><jsd1:Resources>... <jsd1:TotalResourceCount> <jsd1:exact>5.0</jsdl:exact> </jsdl:TotalResourceCount></code>
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.	<code><jsd1:DataStaging><jsd1:FileName> job1/input/control.txt </jsdl:FileName> ...</jsdl:DataStaging></code>
CreationFlag	-f	Specifies whether the file created on the local execution system overwrites or append to an existing file.	<code><jsd1:DataStaging> <jsd1:CreationFlag>overwrite </jsdl:CreationFlag> ...</jsdl:DataStaging></code>
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.	

Element	bsub Option	Description	Example
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><jsd1:DataStaging><jsd1:Source> <jsd1:URI>//input/myjobs/control.txt </jsdl:URI></jsdl:Source> <jsd1:Target><jsd1:URI> //output/myjobs/control.txt </jsdl:URI></jsdl:Target> ...</jsdl:DataStaging></pre>
POSIX Extension Elements			
Executable	N/A	String that specifies the command to execute.	<pre><jsd1-posix:Executable>myscript </jsdl-posix:Executable></pre>
Argument	N/A	Constrained normalized string that specifies an argument for the application or command.	<pre><jsd1-posix:Argument>10 </jsdl-posix:Argument></pre>
Input	-l	String that specifies the Standard Input for the command.	<pre>...<jsd1-posix:Input>input.txt </jsdl-posix:Input>...</pre>
Output	-o	String that specifies the Standard Output for the command.	<pre>...<jsd1-posix:Output>output.txt </jsdl-posix:Output>...</pre>
Error	-e	String that specifies the Standard Error for the command.	<pre>...<jsd1-posix:Error>error.txt </jsdl-posix:Error>...</pre>
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>...<jsd1-posix:WorkingDirectory> /home</jsdl-posix:WorkingDirectory> ...</pre>

Using JSDL Files with LSF

Element	bsub Option	Description	Example
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.	<code><jsd1-posix:Environment name="SHELL"> /bin/bash</jsdl-posix:Environment></code>
WallTimeLimit	-W	Positive integer that specifies the soft limit on the duration of the application's execution, in seconds.	<code><jsd1-posix:WallTimeLimit>60 </jsdl-posix:WallTimeLimit></code>
FileSizeLimit	-F	Positive integer that specifies the maximum size of any file associated with the job, in bytes.	<code><jsd1-posix:FileSizeLimit>1073741824 </jsdl-posix:FileSizeLimit></code>
CoreDumpLimit	-C	Positive integer that specifies the maximum size of core dumps a job may create, in bytes.	<code><jsd1-posix:CoreDumpLimit>0 </jsdl-posix:CoreDumpLimit></code>
DataSegmentLimit	-D	Positive integer that specifies the maximum data segment size, in bytes.	<code><jsd1-posix:DataSegmentLimit>32768 </jsdl-posix:DataSegmentLimit></code>
MemoryLimit	-M	Positive integer that specifies the maximum amount of physical memory that the job can use during execution, in bytes.	<code><jsd1-posix:MemoryLimit>67108864 </jsdl-posix:MemoryLimit></code>
StackSizeLimit	-S	Positive integer that specifies the maximum size of the execution stack for the job, in bytes.	<code><jsd1-posix:StackSizeLimit>1048576 </jsdl-posix:StackSizeLimit></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><jsd1-posix:CPUTimeLimit>30 </jsdl-posix:CPUTimeLimit></code>

Element	bsub Option	Description	Example
ProcessCountLimit	-p	Positive integer that specifies the maximum number of processes the job can spawn.	<code><jsd1-posix:ProcessCountLimit>8</jsdl-posix:ProcessCountLimit></code>
VirtualMemoryLimit	-v	Positive integer that specifies the maximum amount of virtual memory the job can allocate, in bytes.	<code><jsd1-posix:VirtualMemoryLimit>134217728</jsdl-posix:VirtualMemoryLimit></code>
ThreadCountLimit	-T	Positive integer that specifies the number of threads the job can create.	<code><jsd1-posix:ThreadCountLimit>8</jsdl-posix:VirtualMemoryLimit></code>

LSF extension elements

To use all available LSF features, add the elements described in the following table to your JSDL file.

Table 2: 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><jsd1-lsf:SchedulerParams> <jsd1-lsf:Queue>normal</jsdl-lsf:Queue> <jsd1-lsf:ResourceRequirements>"select[swp>15 && hpux] order[ut]" </jsdl-lsf:ResourceRequirements> <jsd1-lsf:Start>12:06:09:55</jsdl-lsf:Start> <jsd1-lsf:Deadline>8:22:15:50</jsdl-lsf:Deadline> <jsd1-lsf:ReservationID>"user1#0"</jsdl-lsf:ReservationID> <jsd1-lsf:Dependencies>'done myjob1'</jsdl-lsf:Dependencies> <jsd1-lsf:Rerunnable>true</jsdl-lsf:Rerunnable> <jsd1-lsf:UserPriority>3</jsdl-lsf:UserPriority> <jsd1-lsf:ServiceClass>platinum</jsdl-lsf:ServiceClass> <jsd1-lsf:Group>sysadmin</jsdl-lsf:Group> <jsd1-lsf:ExternalScheduler></jsdl-lsf:ExternalScheduler><jsd1-lsf:Hold>true </jsdl-lsf:Hold> <jsd1-lsf:JobGroup>/risk_group/portfolio1/current</jsdl-lsf:JobGroup> </jsdl-lsf:SchedulerParams></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 <code>brsvadd</code> 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.

Using JSDL Files with LSF

Element	bsub Option	Description
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.
NotificationParams	N/A	Complex type element that tells LSF when and where to send notification emails for a job. See the following example:
		<pre> <jSDL-lsf:NotificationParams> <jSDL-lsf:NotifyAtStart>true</jSDL-lsf:NotifyAtStart> <jSDL-lsf:NotifyAtFinish>true</jSDL-lsf:NotifyAtFinish> <jSDL-lsf:NotificationEmail>-u user10</jSDL-lsf:NotificationEmail> </jSDL-lsf:NotificationParams> </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 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> <jSDL-lsf:RuntimeParams> <jSDL-lsf:Interactive>1</jSDL-lsf:Interactive> <jSDL-lsf:Block>true</jSDL-lsf:Block> <jSDL-lsf:PreExec>myscript</jSDL-lsf:PreExec> <jSDL-lsf:Checkpoint>myjobs/checkpointdir</jSDL-lsf:Checkpoint> <jSDL-lsf:LoginShell>/csh</jSDL-lsf:LoginShell> <jSDL-lsf:SignalJob>18</jSDL-lsf:SignalJob> <jSDL-lsf:WarningAction>'URG'</jSDL-lsf:WarningAction> <jSDL-lsf:WarningTime>'2'</jSDL-lsf:WarningTime> <jSDL-lsf:SpoolCommand>true</jSDL-lsf:SpoolCommand> <jSDL-lsf:Checkpoint></jSDL-lsf:RuntimeParams> </pre>
Interactive	-l[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.

Element	bsub Option	Description
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 schemas

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.

Collecting 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.

Collecting resource values using elim.jsdl



Controlling Job Execution

- ◆ [Runtime Resource Usage Limits](#) on page 595
- ◆ [Load Thresholds](#) on page 609
- ◆ [Pre-Execution and Post-Execution Commands](#) on page 615
- ◆ [Job Starters](#) on page 625
- ◆ [External Job Submission and Execution Controls](#) on page 631
- ◆ [Configuring Job Controls](#) on page 643

Runtime Resource Usage Limits

Contents

- ◆ [About Resource Usage Limits](#) on page 595
- ◆ [Specifying Resource Usage Limits](#) on page 599
- ◆ [Supported Resource Usage Limits and Syntax](#) on page 601
- ◆ [CPU Time and Run Time Normalization](#) on page 607
- ◆ [PAM resource limits](#) on page 608

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. See Chapter 23, “[Working with Application Profiles](#)” for information about resource usage limits in application profiles.

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.

See Chapter 24, “[Resource Allocation Limits](#)” for more information.

Summary of resource usage limits

Limit	Job syntax (bsub)	Syntax (lsb.queues and lsb.applications)	Format/Default Units
Core file size limit	-C <i>core_limit</i>	CORELIMIT= <i>limit</i>	<i>integer</i> KB
CPU time limit	-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>]
Data segment size limit	-D <i>data_limit</i>	DATALIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i> KB
File size limit	-F <i>file_limit</i>	FILELIMIT= <i>limit</i>	<i>integer</i> KB
Memory limit	-M <i>mem_limit</i>	MEMLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i> KB
Process limit	-p <i>process_limit</i>	PROCESLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i>
Run time limit	-W <i>run_limit</i>	RUNLIMIT=[<i>default</i>] <i>maximum</i>	[<i>hours</i> :] <i>minutes</i> [/ <i>host_name</i> / <i>host_model</i>]
Stack segment size limit	-S <i>stack_limit</i>	STACKLIMIT= <i>limit</i>	<i>integer</i> KB
Virtual memory limit	-v <i>swap_limit</i>	SWAPLIMIT= <i>limit</i>	<i>integer</i> KB
Thread limit	-T <i>thread_limit</i>	THREADLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i>

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`).

About Resource Usage Limits

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`, `lslload`, 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 rusage 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
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
lslload	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.

Specifying 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.

Specify queue-level resource usage limits using parameters in `lsb.queues`.

Specifying queue-level resource usage limits

Limits configured in `lsb.queues` apply to all jobs submitted to the queue. Job-level resource usage limits specified at job submission override the queue definitions.

Maximum value only

Specify only a maximum value for the resource.

For example, to specify a maximum run limit, use one value for the `RUNLIMIT` parameter in `lsb.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 `lsb.queues`:

```
RUNLIMIT = 10
```

The maximum run limit for the queue is 10 minutes. Jobs cannot run for more than 10 minutes.

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 `lsb.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 `lsb.queues`:

- ◆ `CPULIMIT`
- ◆ `DATALIMIT`
- ◆ `MEMLIMIT`

Specifying Resource Usage Limits

- ◆ 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
<code>-C core_limit</code>	<code>CORELIMIT=limit</code>	<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 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
<code>-c cpu_limit</code>	<code>CPULIMIT=[default] maximum</code>	<i>[hours:]minutes[/host_name /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 `myjob` can run for 10 minutes on a DEC3000 host, or the corresponding time on any other host:

```
bsub -c 10/DEC3000 myjob
```

See [CPU Time and Run Time Normalization](#) on page 607 for more information.

Data segment size limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
<code>-D data_limit</code>	<code>DATALIMIT=[default] maximum</code>	<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.queues)	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.queues)	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.

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.queues)	Format/Default Units
<code>-p <i>process_limit</i></code>	<code>PROCESSLIMIT=[<i>default</i>] <i>maximum</i></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.queues)	Format/Default Units
<code>-W <i>run_limit</i></code>	<code>RUNLIMIT=[<i>default</i>] <i>maximum</i></code>	<code>[<i>hours</i>:]<i>minutes</i>[/<i>host_name</i> /<i>host_model</i>]</code>

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.

See [CPU Time and Run Time Normalization](#) on page 607 for more information.

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.queues)	Format/Default Units
-T <i>thread_limit</i>	THREADLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i>

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

Examples

Sets a per-process (soft) stack segment size 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).

An `sbrk()` call to extend the stack segment beyond the stack limit causes the process to be terminated. The default is no soft limit.

Virtual memory (swap) limit

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
<code>-v swap_limit</code>	<code>SWAPLIMIT=limit</code>	<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 `myjob` with a memory limit of 5000 KB.

```
bsub -W 14 myjob
```

`myjob` 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 `myjob` 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 `lsb.params` or in `lsb.applications` 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 `lsb.queues`) if it has been configured, otherwise uses the default CPU time normalization host defined at the cluster level (`DEFAULT_HOST_SPEC` in `lsb.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 `lsb.queues`

PAM resource limits

- 2 DEFAULT_HOST_SPEC is configured in `lsb.params`
- 3 If DEFAULT_HOST_SPEC is not configured in `lsb.queues` or `lsb.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 if `bacct` and `bhist`.

PAM resource limits

PAM limits are system resource limits defined in `limits.conf`.

- ◆ Windows: Not applicable
- ◆ Linux: `/etc/pam.d/lsf`

When `USE_PAM_CREDS` is enabled in `lsb.queues` or `lsb.applications`, 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.applications`, 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/lsf/limits.conf`
- 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.applications`, applies resource limits on jobs running on the execution host.

For more information about configuring a PAM file, check Linux documentation.

Load Thresholds

Contents

- ◆ [Automatic Job Suspension](#) on page 610
- ◆ [Suspending Conditions](#) on page 612

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 `lsb.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 `lsload`, 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 (`lsb.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 normalised 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.

Configuring 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.

Configuring 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.

Example

This queue suspends a job based on the idle time for desktop machines and based on 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
```

Viewing host-level and queue-level suspending conditions

The suspending conditions are displayed by the `bhosts -l` and `bqueues -l` commands.

Viewing job-level suspending conditions

The thresholds that apply to a particular job are the more restrictive of the host and queue thresholds, and are displayed by the `bjobs -l` command.

Viewing suspend reason

The `bjobs -lp` command shows the load threshold that caused LSF to suspend a job, together with the scheduling parameters.

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.

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.

Viewing resume thresholds

The `bjobs -l` command displays the scheduling thresholds that control when a job is resumed.

Pre-Execution and Post-Execution Commands

Jobs can be submitted with optional pre- and post-execution commands. A pre- or post-execution command is an arbitrary command to run before the job starts or after the job finishes. Pre- and post-execution commands are executed in a separate environment from the job.

Contents

- ◆ [About Pre-Execution and Post-Execution Commands](#) on page 616
- ◆ [Configuring Pre- and Post-Execution Commands](#) on page 618

About Pre-Execution and Post-Execution Commands

Each batch job can be submitted with optional pre- and post-execution commands. Pre- and post-execution commands can be any executable command lines to be run before a job is started or after a job finishes.

Some batch jobs require resources that LSF does not directly support. For example, appropriate pre- and/or post-execution commands can be used to handle various situations:

- ◆ Reserving devices like tape drives
- ◆ Creating and deleting scratch directories for a job
- ◆ Customized scheduling
- ◆ Checking availability of software licenses
- ◆ Assigning jobs to run on specific processors on SMP machines
- ◆ Transferring data files needed for job processing
- ◆ Modifying system configuration files before and after running a job

By default, the pre- and post-execution commands are run under the same user ID, environment, and home and working directories as the batch job. If the command is not in your normal execution path, the full path name of the command must be specified.

For parallel jobs, the command is run on the first selected host.

The command path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.

Pre-execution commands

Pre-execution commands support job starting decisions which cannot be configured directly in LSF. LSF supports job-level, queue-level, and application-level (`lsb.applications`) pre-execution.

The pre-execution command returns information to LSF using its exit status. When a pre-execution command is specified, the job is held in the queue until the specified pre-execution command returns exit status zero (0).

If the pre-execution command exits with non-zero status, the batch job is not dispatched. The job goes back to the PEND state, and LSF tries to dispatch another job to that host. While the job is pending, other jobs can proceed ahead of the waiting job. The next time LSF tries to dispatch jobs this process is repeated.

If the pre-execution command exits with a value of 99, the job does not go back to the PEND state, it exits. This gives you flexibility to abort the job if the pre-execution command fails.

LSF assumes that the pre-execution command runs without side effects. For example, if the pre-execution command reserves a software license or other resource, you must not reserve the same resource more than once for the same batch job.

Post-execution commands

If a post-execution command is specified, then the command is run after the job is finished regardless of the exit state of the job.

Post-execution commands are typically used to clean up some state left by the pre-execution and the job execution. LSF supports job-level, queue-level, and application-level (`lsb.applications`) post-execution.

Job-level commands

The `bsub -E` option specifies an arbitrary command to run before starting the batch job. When LSF finds a suitable host on which to run a job, the pre-execution command is executed on that host. If the pre-execution command runs successfully, the batch job is started.

The `bsub -Ep` option specifies job-level post-execution commands to run on the execution host after the job finishes.

Queue-level and application-level commands

In some situations (for example, license checking), it is better to specify a queue-level or application-level pre-execution command instead of requiring every job be submitted with the `-E` option of `bsub`.

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.

Application level pre-execution commands run on the execution host before the job associated with the application profile is dispatched on an execution host.

When a job finishes, the application-level post-execution commands run, followed by queue-level post-execution commands if any.

Application-level post-execution commands run on the execution host after the job associated with the application profile has finished running on the execution host. They also run if the `PRE_EXEC` command exits with a 0 exit status, but the job execution environment failed to be set up.

Post-execution job 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.

By default, 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.

The `bhist` command displays the `POST_DONE` and `POST_ERR` states. The resource usage of post-processing is not included in the job resource usage.

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

Configuring Pre- and Post-Execution Commands

Pre-execution commands can be configured at the job level, in queues, or in application profiles.

Post-execution commands can be configured at the job level, in queues or in application profiles.

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	1 Queue level 2 Job level
Queue and application levels	1 Queue level 2 Application level
Queue and job levels	1 Queue level 2 Job level
Application and job levels	1 Job level

Post-execution commands run in the following order:

- 1 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.
- 2 The queue-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	1 Job level 2 Queue level
Queue and application levels	1 Application level 2 Queue level
Queue and job levels	1 Job level 2 Queue level

Job-level commands

Job-level pre-execution and post-execution commands require no configuration. Use the `bsub -E` option to specify an arbitrary command to run before the job starts. Use the `bsub -Ep` option to specify an arbitrary command to run after the job finishes running.

Example

The following example shows a batch job that requires a tape drive. The user program `tapeCheck` exits with status zero if the specified tape drive is ready:

```
bsub -E "/usr/share/bin/tapeCheck /dev/rmt01" myJob
```

Queue-level commands

Use the `PRE_EXEC` and `POST_EXEC` keywords in the queue definition (`lsb.queues`) to specify pre- and post-execution commands.

The following points should be considered when setting up pre- and post-execution commands at the queue level:

- ◆ If the pre-execution command exits with a non-zero exit code, then it is considered to have failed and the job is requeued to the head of the queue. This feature can be used to implement customized scheduling by having the pre-execution command fail if conditions for dispatching the job are not met.
- ◆ Other environment variables set for the job are also set for the pre- and post-execution commands.
- ◆ When a job is dispatched from a queue which has a post-execution command, LSF remembers the post-execution command defined for the queue from which the job is dispatched. If the job is later switched to another queue or the post-execution command of the queue is changed, LSF still runs the original post-execution command for this job.
- ◆ When the post-execution command is run, the environment variable, `LSB_JOBEXIT_STAT`, is set to the exit status of the job. See the man page for the `wait(2)` command for the format of this exit status.
- ◆ The post-execution command is also run if a job is requeued because the job's execution environment fails to be set up, or if the job exits with one of the queue's `REQUEUE_EXIT_VALUES`. The `LSB_JOBPEND` environment variable is set if the job is requeued. If the job's execution environment could not be set up, `LSB_JOBEXIT_STAT` is set to 0.
- ◆ 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. Use `LSB_DISABLE_RERUN_POST_EXEC=Y` in `lsf.conf` to prevent the post-exec from running when a job is rerun.
- ◆ If both queue and job-level pre-execution commands are specified, the job-level pre-execution is run after the queue-level pre-execution command.
- ◆ If both application-level and job-level post-execution commands 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

Example

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
```

Configuring Pre- and Post-Execution Commands

```
PRE_EXEC      = /usr/share/lsf/pri_prexec
POST_EXEC     = /usr/share/lsf/pri_postexec
End Queue
```

See the `lsb.queues` template file for additional queue examples.

Application-level commands

Use the `PRE_EXEC` and `POST_EXEC` keywords in the application profile definition (`lsb.applications`) to specify pre- and post-execution commands.

The following points should be considered when setting up pre- and post-execution commands at the application level:

- ◆ When a job finishes, the application-level post-execution commands run, followed by queue-level post-execution commands if any.
- ◆ Environment variables set for the job are also set for the pre- and post-execution commands.
- ◆ 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.
- ◆ When a job is submitted to an application profile that has a pre-execution command, the system will remember the post-execution command defined for the application profile from which the job is dispatched. If the job is later moved to another application profile or the post-execution command of the application profile is changed, the original post-execution command will be run.
- ◆ When the post-execution command is run, the environment variable `LSB_JOBEXIT_STAT` is set to the exit status of the job. Refer to the man page for `wait(2)` for the format of this exit status.
- ◆ The post-execution command is also run if a job is requeued because the job's execution environment fails to be set up or if the job exits with one of the application profile's `REQUEUE_EXIT_VALUES`. The environment variable `LSB_JOBPEND` is set if the job is requeued. If the job's execution environment could not be set up, `LSB_JOBEXIT_STAT` is set to 0 (zero).
- ◆ If the pre-execution command exits with a non-zero exit code, it is considered to have failed, and the job is requeued to the head of the queue. Use this feature to implement customized scheduling by having the pre-execution command fail if conditions for dispatching the job are not met.

Example

```
Begin Application
NAME          = catia
DESCRIPTION   = CATIA V5
CPULIMIT      = 24:0/hostA      # 24 hours of host hostA
FILELIMIT     = 20000
DATALIMIT     = 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
```

See the `lsb.applications` template file for additional application profile examples.

Pre- and post-execution on UNIX and Linux

The entire contents of the configuration line of the pre- and post-execution commands are run under `/bin/sh -c`, so shell features can be used in the command.

For example, the following is valid:

```
PRE_EXEC = /usr/share/lsf/misc/testq_pre >> /tmp/pre.out
POST_EXEC = /usr/share/lsf/misc/testq_post | grep -v "Hey!"
```

The pre- and post-execution commands are run in `/tmp`.

Standard input and standard output and error are set to `/dev/null`. The output from the pre- and post-execution commands can be explicitly redirected to a file for debugging purposes.

The `PATH` environment variable is set to:

```
PATH='/bin /usr/bin /sbin /usr/sbin'
```

Pre- and post-execution on Windows

The pre- and post-execution commands are run under `cmd.exe /c`.

NOTE: For pre- and post-execution commands that execute on a Windows Server 2003, x64 Edition platform, users must have "Read" and "Execute" privileges for `cmd.exe`.

Standard input and standard output and error are set to `NULL`. The output from the pre- and post-execution commands can be explicitly redirected to a file for debugging purposes.

Setting a pre- and post-execution user ID

By default, both the pre- and post-execution commands are run as the job submission user. Use the `LSB_PRE_POST_EXEC_USER` parameter in `lsf.sudoers` to specify a different user ID for queue-level and application-level pre- and post-execution commands.

Example

If the pre- or post-execution commands perform privileged operations that require root permission, specify:

```
LSB_PRE_POST_EXEC_USER=root
```

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

Including job post-execution in job finish status reporting

By default, LSF releases resources for a job as soon as the job is finished and when `sbatchd` reports job finish status (`DONE` or `EXIT`) to `mbatchd`. Post-execution processing is not considered part of job processing. This makes it possible for a new job to be started before post-execution processing for a previous job is complete.

There are situations where you do not want the first job's post-execution affecting the second job's execution. Or the execution of a second job might crucially depend on the completion of post-execution of the previous job.

In other cases, you may want to include job post-execution in job accounting processes, or if the post-execution is CPU intensive, you might not want a second job running at the same time. Finally, system configuration required by the original job may be changed or removed by a new job, which could prevent the first job from finishing normally.

To enable all associated processing to complete before LSF reports job finish status, configure `JOB_INCLUDE_POSTPROC=Y` in an application profile in `lsb.applications` or cluster wide in `lsb.params`.

When `JOB_INCLUDE_POSTPROC` is set:

- ◆ `sbatchd` sends both job finish status (`DONE` or `EXIT`) and post-execution status (`POST_DONE` or `POST_ERR`) to `mbatchd` at the same time
- ◆ The job remains in `RUN` state and holds its job slot until the job post-execution processing has finished
- ◆ Jobs can now depend on the completion of post-execution processing
- ◆ `bjobs`, `bhist`, and `bacct` will show the same time for both job finish and post-execution finish
- ◆ Job requeue will happen after post-execution processing, not when the job finishes

For job history and job accounting, the job CPU time and run time will also include the post-execution CPU time and run time.

Limitations and side-effects

Job query commands (`bjobs`, `bhist`) show that the job remains in `RUN` state until the post-execution processing is finished, even though the job itself has finished. Job control commands (`bstop`, `bkill`, `bresume`) will have no effect.

Rerunnable jobs may rerun after they have actually finished because the host became unavailable before post-execution processing finished, but the `mbatchd` considers the job still in `RUN` state.

Job preemption is delayed until post-execution processing is finished.

Post-execution on SGI cpusets

Post-execution processing on SGI cpusets behave differently from previous releases. If `JOB_INCLUDE_POSTPROC=Y` is specified in `lsb.applications` or cluster wide in `lsb.params`, post-execution processing is not attached to the job cpuset, and Platform LSF does not release the cpuset until post-execution processing has finished.

Preventing job overlap on hosts

You can use `JOB_INCLUDE_POSTPROC` to ensure that there is no execution overlap among running jobs. For example, you may have pre-execution processing to create a user execution environment at the desktop (mount a disc for the user, create `rlogin` permissions, etc.) Then you configure post-execution processing to clean up the user execution environment set by the pre-exec.

If the post-execution for one job is still running when a second job is dispatched, pre-execution processing that sets up the user environment for the next job may not be able to run correctly because the previous job's environment has not yet been cleaned up by its post-exec.

You should configure jobs to run exclusively to prevent the actual jobs from not overlapping, but in this case, you also need to configure post-execution to be included in job finish status reporting.

Setting a post-execution timeout

Configure `JOB_POSTPROC_TIMEOUT` in an application profile in `lsb.applications` or cluster wide in `lsb.params` to control how long post-execution processing is allowed to run.

`JOB_POSTPROC_TIMEOUT` specifies a timeout in minutes for job post-execution processing. If post-execution processing takes longer than the timeout, `sbatchd` reports the post-execution has failed (`POST_ERR` status), and kills the process group of the job's post-execution processes.

The specified timeout must be greater than zero.

If `JOB_INCLUDE_POSTPROC` is enabled in the application profile or cluster wide in `lsb.params`, and `sbatchd` kills the post-execution processes because the timeout has been reached, the CPU time of the post-execution processing is set to 0, and the job CPU time will not include the CPU time of the post-execution processing.

Controlling how many times pre-execution commands are retried

By default, if job pre-execution fails, LSF retries the job automatically.

Configure `MAX_PREEEXEC_RETRY`, `LOCAL_MAX_PREEEXEC_RETRY`, or `REMOTE_MAX_PREEEXEC_RETRY` to limit the number of times LSF retries job pre-execution. Pre-execution retry is configured cluster-wide (`lsb.params`), at the queue level (`lsb.queues`), and at the application level (`lsb.applications`).

Pre-execution retry configured in `lsb.applications` overrides `lsb.queues`, and `lsb.queues` overrides `lsb.params` configuration.

Configuring Pre- and Post-Execution Commands

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.

Contents

- ◆ [About Job Starters](#) on page 625
- ◆ [Command-Level Job Starters](#) on page 626
- ◆ [Queue-Level Job Starters](#) on page 628
- ◆ [Controlling Execution Environment Using Job Starters](#) on page 629

About Job Starters

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 `lsrun`, `lsgrun`, 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. See [Controlling Execution Environment Using Job Starters](#) on page 629 for detailed information.

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`. See [Queue-Level Job Starters](#) on page 628 for detailed information.

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.

See Chapter 38, “[Pre-Execution and Post-Execution Commands](#)” for more information.

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`.
- ◆ 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. See Chapter 42, “[Interactive Jobs with bsub](#)” for information on 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. See [Queue-Level Job Starters](#) on page 628 for more information.

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 `lshrun`, `lshgrun`, 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:

```
% lshrun "'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:\>lshrun dir /p
```

The command that actually runs is:

```
C:\cmd.exe /C dir /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. See Chapter 42, “[Interactive Jobs with `bsub`](#)” for information on interactive batch jobs.

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. See [Command-Level Job Starters](#) on page 626 for more information.

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"
```

For more information

See the *Platform LSF Configuration Reference* for information about the `JOB_STARTER` parameter in the `lsb.queues` file.

Controlling Execution Environment Using 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`.

Adding 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`

Controlling Execution Environment Using Job Starters

- ◆ 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.

External Job Submission and Execution Controls

This document describes the use of external job submission and execution controls called `esub` and `eexec`. These site-specific user-written executables are used to validate, modify, and reject job submissions, pass data to and modify job execution environments.

Contents

- ◆ [Understanding External Executables](#) on page 631
- ◆ [Using `esub`](#) on page 632
- ◆ [Working with `eexec`](#) on page 641

Understanding External Executables

About `esub` and `eexec`

LSF provides the ability to validate, modify, or reject job submissions, modify execution environments, and pass data from the submission host directly to the execution host through the use of the `esub` and `eexec` executables. Both are site-specific and user written and must be located in `LSF_SERVERDIR`.

Validate, modify, or reject a job

To validate, modify, or reject a job, an `esub` needs to be written. See [Using `esub`](#) on page 632

Modifying execution environments

To modify the execution environment on the execution host, an `eexec` needs to be written. See [Working with `eexec`](#) on page 641

Passing data

To pass data directly to the execution host, an `esub` and `eexec` need to be written. See [Using `esub` and `eexec` to pass data to execution environments](#) on page 641

Interactive remote execution

Interactive remote execution also runs `esub` and `eexec` if they are found in `LSF_SERVERDIR`. For example, `lsrun` invokes `esub`, and `RES` runs `eexec` before starting the task. `esub` is invoked at the time of the `ls_connect(3)` call, and `RES` invokes `eexec` each time a remote task is executed. `RES` runs `eexec` only at task startup time.

DCE credentials and AFS tokens

`esub` and `eexec` are also used for processing DCE credentials and AFS tokens. See the following documents on the Platform Web site for more information:

- ◆ [“Installing LSF on AFS”](#)
- ◆ [“Installing LSF on DCE/DFS”](#)

Using esub

About esub

An `esub`, short for *external submission*, is a user-written executable (binary or script) that can be used to validate, modify, or reject jobs. The `esub` is put into `LSF_SERVERDIR` (defined in `lsf.conf`) where LSF checks for its existence when a job is submitted, restarted, and modified. If LSF finds an `esub`, it is run by LSF. Whether the job is submitted, modified, or rejected depends on the logic built into the `esub`.

Any messages that need to be provided to the user should be directed to the standard error (`stderr`) stream and not the standard output (`stdout`) stream.

In this section

- ◆ [Environment variables to bridge esub and LSF](#) on page 632
- ◆ [General esub logic](#) on page 636
- ◆ [Rejecting jobs](#) on page 637
- ◆ [Validating job submission parameters](#) on page 637
- ◆ [Modifying job submission parameters](#) on page 637
- ◆ [Using bmod and brestart commands with mesub](#) on page 638
- ◆ [Use multiple esub \(mesub\)](#) on page 638
- ◆ [How master esub invokes application-specific esubs](#) on page 639
- ◆ [Configure master esub and your application-specific esub](#) on page 640

Environment variables to bridge esub and LSF

LSF provides the following environment variables in the `esub` execution environment:

LSB_SUB_PARM_FILE

This variable points to a temporary file containing the job parameters that `esub` reads when the job is submitted. The submission parameters are a set of name-value pairs on separate lines in the format "`option_name=value`".

The following option names are supported:

Option	Description
LSB_SUB_ADDITIONAL	String format parameter containing the value of the <code>-a</code> option to <code>bsub</code> The value of <code>-a</code> is passed to <code>esub</code> , but it does not directly affect the other <code>bsub</code> parameters or behavior. The value of <code>-a</code> must correspond to an actual <code>esub</code> file. For example, to use <code>bsub -a fluent</code> , the file <code>esub.fluent</code> must exist in <code>LSF_SERVERDIR</code> . LSB_SUB_ADDITIONAL cannot be changed in or added to <code>LSB_SUB_MODIFY_FILE</code> .
LSB_SUB_BEGIN_TIME	Begin time, in seconds since 00:00:00 GMT, Jan. 1, 1970
LSB_SUB_CHKPNT_DIR	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	<code>bsub</code> job command argument LSB_SUB_COMMANDNAME must be set in <code>lsf.conf</code> to enable <code>esub</code> to use this variable
LSB_SUB_CHKPNT_PERIOD	Checkpoint period in seconds
LSB_SUB_DEPEND_COND	Dependency condition
LSB_SUB_ERR_FILE	Standard error file name
LSB_SUB_EXCEPTION	Exception condition
LSB_SUB_EXCLUSIVE	"Y" specifies exclusive execution
LSB_SUB_EXTSCHED_PARAM	Validate or modify <code>bsub -extsched</code> option
LSB_SUB_HOLD	Hold job (<code>bsub -H</code> option)
LSB_SUB_HOSTS	List of execution host names
LSB_SUB_HOST_SPEC	Host specifier
LSB_SUB_IN_FILE	Standard input file name
LSB_SUB_INTERACTIVE	"Y" specifies an interactive job
LSB_SUB_LOGIN_SHELL	Login shell
LSB_SUB_JOB_NAME	Job name
LSB_SUB_JOB_WARNING_ACTION	Job warning action specified by <code>bsub -wa</code>
LSB_SUB_JOB_ACTION_WARNING_TIME	Job warning time period specified by <code>bsub -wt</code>
LSB_SUB_MAIL_USER	Email address used by LSF for sending job email
LSB_SUB_MAX_NUM_PROCESSORS	Maximum number of processors requested
LSB_SUB_MODIFY	"Y" specifies a modification request
LSB_SUB_MODIFY_ONCE	"Y" specifies a modification-once request
LSB_SUB_NOTIFY_BEGIN	"Y" specifies email notification when job begins
LSB_SUB_NOTIFY_END	"Y" specifies email notification when job ends
LSB_SUB_NUM_PROCESSORS	Minimum number of processors requested
LSB_SUB_OTHER_FILES	The value is <code>SUB_RESET</code> if defined to indicate a <code>bmod</code> is being performed to reset the number of files to be transferred. The file path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.

Using esub

Option	Description
LSB_SUB_OTHER_FILES_ <i>number</i>	<i>number</i> is an index number indicating the particular file transfer value is the specified file transfer expression. For example, for <code>bsub -f "a > b" -f "c < d"</code> , the following would be defined: LSB_SUB_OTHER_FILES_0="a > b" LSB_SUB_OTHER_FILES_1="c < d"
LSB_SUB_OUT_FILE	Standard output file name
LSB_SUB_PRE_EXEC	Pre-execution command. The command path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.
LSB_SUB_PROJECT_NAME	Project name
LSB_SUB_PTY	"Y" specifies an interactive job with PTY support
LSB_SUB_PTY_SHELL	"Y" specifies an interactive job with PTY shell support
LSB_SUB_QUEUE	Submission queue name
LSB_SUB_RERUNNABLE	"Y" specifies a rerunnable job "N" specifies a nonrerunnable job (specified with <code>bsub -rn</code>). The job is not rerunnable even it was submitted to a rerunable queue or application profile For <code>bmod -rn</code> , the value is SUB_RESET.
LSB_SUB_RES_REQ	Resource requirement string— <i>does not</i> support multiple resource requirement strings
LSB_SUB_RESTART	"Y" specifies a restart job
LSB_SUB_RESTART_FORCE	"Y" specifies forced restart job
LSB_SUB_RLIMIT_CORE	Core file size limit
LSB_SUB_RLIMIT_CPU	CPU limit
LSB_SUB_RLIMIT_DATA	Data size limit
LSB_SUB_RLIMIT_FSIZE	File size limit
LSB_SUB_RLIMIT_PROCESS	Process limit
LSB_SUB_RLIMIT_RSS	Resident size limit
LSB_SUB_RLIMIT_RUN	Wall-clock run limit
LSB_SUB_RLIMIT_STACK	Stack size limit
LSB_SUB_RLIMIT_SWAP	Virtual memory limit (swap space)
LSB_SUB_RLIMIT_THREAD	Thread limit
LSB_SUB_TERM_TIME	Termination time, in seconds, since 00:00:00 GMT, Jan. 1, 1970
LSB_SUB_TIME_EVENT	Time event expression
LSB_SUB_USER_GROUP	User group name
LSB_SUB_WINDOW_SIG	Window signal number
LSB_SUB2_JOB_GROUP	Options specified by <code>bsub -g</code>
LSB_SUB2_LICENSE_PROJECT	LSF License Scheduler project name specified by <code>bsub -Lp</code>
LSB_SUB2_IN_FILE_SPOOL	Spooled input file (<code>bsub -is</code>)
LSB_SUB2_JOB_CMD_SPOOL	Spooled job command file (<code>bsub -Zs</code>)
LSB_SUB2_JOB_PRIORITY	Job priority (<code>bsub -sp</code> and <code>bmod -sp</code>) For <code>bmod -spn</code> , the value is SUB_RESET
LSB_SUB2_SLA	SLA scheduling options
LSB_SUB2_USE_RSV	Advance reservation ID specified by <code>bsub -U</code>

Option	Description
LSB_SUB3_ABSOLUTE_PRIORITY	For <code>bmod -aps</code> , the value equal to the APS string given with the <code>bmod -aps</code> . For <code>bmod -apn</code> , the value is <code>SUB_RESET</code> .
LSB_SUB3_APP	Options specified by <code>bsub -app</code> and <code>bmod -app</code> . For <code>bmod -apn</code> , the value is <code>SUB_RESET</code> .
LSB_SUB3_AUTO_RESIZABLE	Defines the job autoresizable attribute. LSB_SUB3_AUTO_RESIZABLE=Y if <code>bsub -ar</code> or <code>bmod -ar</code> is specified. LSB_SUB3_AUTO_RESIZABLE=SUB_RESET if <code>bmod -arn</code> is used.
LSB_SUB3_RESIZE_NOTIFY_CMD	Define the job resize notification command. LSB_SUB3_RESIZE_NOTIFY_CMD=<cmd> if <code>bsub -rnc</code> or <code>bmod -rnc</code> is specified. LSB_SUB3_RESIZE_NOTIFY_CMD=SUB_RESET if <code>bmod -rnc</code> is used.
LSB_SUB3_JOB_QUEUE	String format parameter containing the value of the <code>-Q</code> option to <code>bsub</code> . For <code>bmod -Qn</code> , the value is <code>SUB_RESET</code> .
LSB_SUB3_CWD	Current working directory specified on the command line with <code>bsub -cwd</code>
LSB_SUB_INTERACTIVE LSB_SUB3_INTERACTIVE_SSH	If both are specified by "Y", the session of the interactive job is encrypted with SSH. <code>bsub -IS</code> <code>bsub -ISs</code>
LSB_SUB_INTERACTIVE LSB_SUB_PTY LSB_SUB3_INTERACTIVE_SSH	If <code>LSB_SUB_INTERACTIVE</code> is specified by "Y", <code>LSB_SUB_PTY</code> is specified by "Y" and <code>LSB_SUB3_INTERACTIVE_SSH</code> is specified by "Y", the session of interactive job with PTY support will be encrypted by SSH. <code>bsub -ISp</code>
LSB_SUB_INTERACTIVE LSB_SUB_PTY LSB_SUB_PTY_SHELL LSB_SUB3_INTERACTIVE_SSH	If <code>LSB_SUB_INTERACTIVE</code> is specified by "Y", <code>LSB_SUB_PTY</code> is specified by "Y", <code>LSB_SUB_PTY_SHELL</code> is specified by "Y", and <code>LSB_SUB3_INTERACTIVE_SSH</code> is specified by "Y", the session of interactive job with PTY shell support will be encrypted by SSH. <code>bsub -ISs</code>
LSB_SUB3_POST_EXEC	Run the specified post-execution command on the execution host after the job finishes. Specified by <code>bsub -Ep</code> . The command path directory can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.
LSB_SUB3_RUNTIME_ESTIMATION	Runtime estimate specified by <code>bsub -We</code>
LSB_SUB3_USER_SHELL_LIMITS	Pass user shell limits to execution host. Specified by <code>bsub -ul</code> .
LSB_SUB_INTERACTIVE LSB_SUB3_XJOB_SSH	If both are specified by "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. <code>bsub -IX</code>

Example submission parameter file

If a user submits the following job:

```
bsub -q normal -x -P my_project -R "r1m rusage[dummy=1]" -n 90 sleep
10
```

Using esub

The contents of the `LSB_SUB_PARM_FILE` will be:

```
LSB_SUB_QUEUE="normal"  
LSB_SUB_EXCLUSIVE=Y  
LSB_SUB_RES_REQ="r1m rusage[dummy=1]"  
LSB_SUB_PROJECT_NAME="my_project"  
LSB_SUB_COMMAND_LINE="sleep 10"  
LSB_SUB_NUM_PROCESSORS=90  
LSB_SUB_MAX_NUM_PROCESSORS=90
```

LSB_SUB_ABORT_VALUE

This variable indicates the value `esub` should exit with if LSF is to reject the job submission.

LSB_SUB_MODIFY_ENVFILE

The file in which `esub` should write any changes to the job environment variables. `esub` writes the variables to be modified to this file in the same format used in `LSB_SUB_PARM_FILE`. The order of the variables does not matter.

After `esub` runs, LSF checks `LSB_SUB_MODIFY_ENVFILE` for changes and if found, LSF will apply them to the job environment variables.

LSB_SUB_MODIFY_FILE

The file in which `esub` should write any submission parameter changes.

`esub` writes the job options to be modified to this file in the same format used in `LSB_SUB_PARM_FILE`. The order of the options does not matter. After `esub` runs, LSF checks `LSB_SUB_MODIFY_FILE` for changes and if found LSF will apply them to the job.

TIP: `LSB_SUB_ADDITIONAL` cannot be changed in or added to `LSB_SUB_MODIFY_FILE`.

LSF_INVOKE_CMD

Indicates the name of the last LSF command that invoked an external executable (for example, `esub`).

External executables get called by several LSF commands (`bsub`, `bmod`, `lrun`). This variable contains the name of the last LSF command to call the executable.

General esub logic

After `esub` runs, LSF checks:

- 1 Is the `esub` exit value `LSB_SUB_ABORT_VALUE`?
 - a Yes, step 2
 - b No, step 4
- 2 Reject the job
- 3 Go to step 5
- 4 Does `LSB_SUB_MODIFY_FILE` or `LSB_SUB_MODIFY_ENVFILE` exist?
 - ❖ Apply changes
- 5 Done

Rejecting jobs

Depending on your policies you may choose to reject a job. To do so, have `esub` exit with `LSB_SUB_ABORT_VALUE`.

If `esub` rejects the job, it should not write to either `LSB_SUB_MODIFY_FILE` or `LSB_SUB_MODIFY_ENVFILE`.

Example

The following Bourne shell `esub` rejects all job submissions by exiting with `LSB_SUB_ABORT_VALUE`:

```
#!/bin/sh

# Redirect stderr to stdout so echo can be used for
# error messages
exec 1>&2

# Reject the submission
echo "LSF is Rejecting your job submission..."
exit $LSB_SUB_ABORT_VALUE
```

Validating job submission parameters

One use of validation is to support project-based accounting. The user can request that the resources used by a job be charged to a particular project. Projects are associated with a job at job submission time, so LSF will accept any arbitrary string for a project name. In order to ensure that only valid projects are entered and the user is eligible to charge to that project, an `esub` can be written.

Example

The following Bourne shell `esub` validates job submission parameters:

```
#!/bin/sh

. $LSB_SUB_PARM_FILE

# Redirect stdout to stderr so echo can be used for error messages
exec 1>&2

# Check valid projects
if [ $LSB_SUB_PROJECT_NAME != "proj1" -o $LSB_SUB_PROJECT_NAME != "proj2" ]; then
    echo "Incorrect project name specified"
    exit $LSB_SUB_ABORT_VALUE
fi

USER=`whoami`
if [ $LSB_SUB_PROJECT_NAME = "proj1" ]; then
    # Only user1 and user2 can charge to proj1
    if [ $USER != "user1" -a $USER != "user2" ]; then
        echo "You are not allowed to charge to this project"
        exit $LSB_SUB_ABORT_VALUE
    fi
fi
```

Modifying job submission parameters

`esub` can be used to modify submission parameters and the job environment before the job is actually submitted.

The following example writes modifications to `LSB_SUB_MODIFY_FILE` for the following parameters:

- ◆ `LSB_SUB_QUEUE`
- ◆ `USER`
- ◆ `SHELL`

In the example, user `userA` can only submit jobs to queue `queueA`. User `userB` must use Bourne shell (`/bin/sh`), and user `userC` should never be able to submit a job.

```
#!/bin/sh
. $LSB_SUB_PARM_FILE

# Redirect stderr to stdout so echo can be used for error messages
exec 1>&2

USER=`whoami`
# Ensure 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

# Ensure 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
```

Using `bmod` and `brestart` commands with `mesub`

You can use the `bmod` command to modify job submission parameters, and `brestart` to restart checkpointed jobs. Like `bsub`, `bmod` and `brestart` also call `mesub`, which in turn invoke any existing `esub` executables in `LSF_SERVERDIR`. `bmod` and `brestart` cannot make changes to the job environment through `mesub` and `esub`. Environment changes only occur when `mesub` is called by the original job submission with `bsub`.

Use multiple `esub` (`mesub`)

LSF provides a master `esub` (`LSF_SERVERDIR/mesub`) to handle the invocation of individual application-specific `esub` executables and the job submission requirements of your applications.

- 1 Use the `-a` option of `bsub` to specify the application you are running through LSF.

For example, to submit a FLUENT job:

```
bsub -a fluent bsub_options fluent_command
```

The method name `fluent`, uses the `esub` for FLUENT jobs (`LSF_SERVERDIR/esub.fluent`), which sets the checkpointing method `LSB_ECHKPNT_METHOD="fluent"` to use the `echkpnt.fluent` and `erestart.fluent`.

LSB_ESUB_METHOD (lsf.conf)

To specify a mandatory `esub` method that applies to all job submissions, you can configure `LSB_ESUB_METHOD` in `lsf.conf`.

`LSB_ESUB_METHOD` specifies the name of the `esub` method used in addition to any methods specified in the `bsub -a` option.

For example, `LSB_ESUB_METHOD="dce fluent"` defines DCE as the mandatory security system, and FLUENT as the mandatory application used on all jobs.

Compatibility note

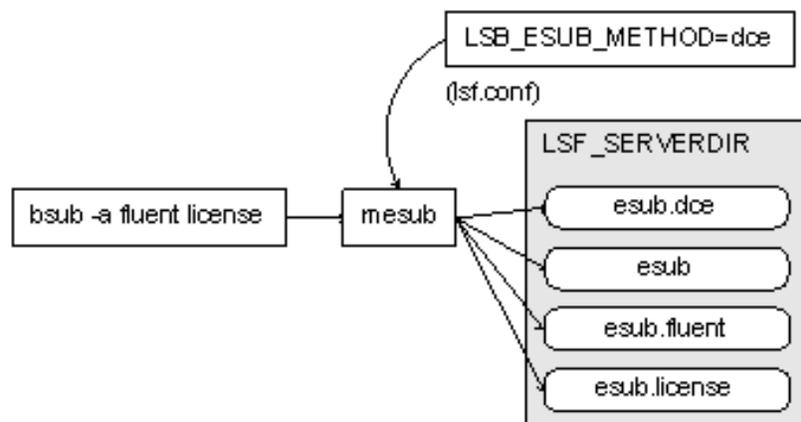
RESTRICTION: After LSF version 5.1, the value of `-a` and `LSB_ESUB_METHOD` must correspond to an actual `esub` file in `LSF_SERVERDIR`. For example, to use `bsub -a fluent`, the file `esub.fluent` must exist in `LSF_SERVERDIR`.

How master esub invokes application-specific esubs

`bsub` invokes `mesub` at job submission, which calls `esub` programs in this order:

- 1 Mandatory `esub` programs defined by `LSB_ESUB_METHOD`
- 2 Any existing executable named `LSF_SERVERDIR/esub`
- 3 Application-specific `esub` programs in the order specified in the `bsub -a` option

Example



In this example:

- ◆ `esub.dce` is defined as the only mandatory `esub`
- ◆ An executable named `esub` already exists in `LSF_SERVERDIR`
- ◆ Executables named `esub.fluent` and `esub.license` exist in `LSF_SERVERDIR`

Existing esub

- ◆ `bsub -a fluent license` submits the job as a FLUENT job, and `mesub` invokes the following esub executables in `LSF_SERVERDIR` in this order:
 - ❖ `esub.dce`
 - ❖ `esub`
 - ❖ `esub.fluent`
 - ❖ `esub.license`
- ◆ `bsub` without the `-a` option submits the job, and `mesub` invokes only the mandatory `esub.dce` and the existing `esub` in `LSF_SERVERDIR`, not the application-specific esub programs.

Configure master esub and your application-specific esub

The master esub is installed as `LSF_SERVERDIR/mesub`. After installation:

- 1 Create your own application-specific esub.
- 2 Optional. Configure `LSB_ESUB_METHOD` in `lsf.conf` to specify a mandatory esub for all job submissions.

Name your esub

- 1 Use the following naming conventions:
 - ❖ On UNIX, `LSF_SERVERDIR/esub.application`
 - ❖ On Windows, `LSF_SERVERDIR\esub.application.[exe |bat]`

For FLUENT jobs, for example:

- ◆ UNIX: `esub.fluent`
- ◆ Windows: `esub.fluent.exe`

The name of the esub program must be a valid file name. It can contain only alphanumeric characters, underscore (`_`) and hyphen (`-`).

CAUTION: The file name `esub.user` is reserved for backward compatibility. Do not use the name `esub.user` for your application-specific esub.

Existing esub

Your existing esub does not need to follow this convention and does not need to be renamed. However, since `mesub` invokes any esub that follows this convention, you should move any backup copies of your esubs out of `LSF_SERVERDIR` or choose a name that does not follow the convention (for example, use `esub_bak` instead of `esub.bak`).

Working with eexec

About eexec

The `eexec` program runs on the execution host at job start-up and completion time and when checkpointing is initiated. It is run as the user after the job environment variables have been set. The environment variable `LS_EXEC_T` is set to `START`, `END`, and `CHKPNT`, respectively, to indicate when `eexec` is invoked.

If you need to run `eexec` as a different user, such as `root`, you must properly define `LSF_EEXEC_USER` in the file `/etc/lsf.sudoers`. See the *Platform LSF Configuration Reference* for information about the `lsf.sudoers` file.

`eexec` is expected to finish running because the parent job process waits for `eexec` to finish running before proceeding. The environment variable `LS_JOBPID` stores the process ID of the process that invoked `eexec`. If `eexec` is intended to monitor the execution of the job, `eexec` must fork 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.

Using `esub` and `eexec` to pass data to execution environments

If `esub` needs to pass some data to `eexec`, it can write the data to its standard output for `eexec` to read from its standard input (`stdin`). LSF effectively acts as the pipe between `esub` and `eexec` (e.g., `esub | eexec`).

Standard output (`stdout`) from any `esub` is automatically sent to `eexec`.

Limitation

Since `eexec` cannot handle more than one standard output stream, only one `esub` can use standard output to generate data as standard input to `eexec`.

For example, the `esub` for AFS (`esub.afs`) sends its authentication tokens as standard output to `eexec`. If you use AFS, no other `esub` can use standard output.

Working with eexec

Configuring 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.

Contents

- ◆ [Default Job Control Actions](#) on page 643
- ◆ [Configuring Job Control Actions](#) on page 645
- ◆ [Customizing Cross-Platform Signal Conversion](#) on page 648

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.

See [Killing Jobs](#) on page 131 for more information about job controls and the LSF commands that perform them.

SUSPEND action

Change a running job from RUN state to one of the following states:

- ◆ USUSP or PSUSP in response to `bstop`
- ◆ SSUSP state when the LSF system suspends the job

The default action is to send the following signals to the job:

Default Job Control Actions

- ◆ 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 `bstop` 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 `bresume` 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 `bkill` or `bqueue` 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.

Configuring 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`.

See the *Platform LSF Configuration Reference* for information about the `lsb.queues` file.

JOB_CONTROLS parameter (lsb.queues)

The `JOB_CONTROLS` parameter has the following format:

```
Begin Queue
...
JOB_CONTROLS = SUSPEND[signal | CHPNT | command] \
                RESUME[signal | command] \
                TERMINATE[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 CHKPNT, the job is checkpointed and then stopped by sending the SIGSTOP signal to the job automatically.
- ◆ If the TERMINATE action is CHKPNT, 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=TERMINATE[bkill]` or `JOB_CONTROLS=TERMINATE[brequeue]`). This will cause a deadlock between the signal and the action.

Using 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 `lsbatch.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

```
TERMINATE_WHEN = [LOAD] [PREEMPT] [WINDOW]
```

Example

The following defines a night queue that will kill jobs if the run window closes.

```
Begin Queue
NAME          = night
RUN_WINDOW    = 20:00-08:00
TERMINATE_WHEN = WINDOW
JOB_CONTROLS  = TERMINATE[ kill -KILL $LSB_JOBPIIDS;
                  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.

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

Avoiding 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 `kill` command uses the TERMINATE action, so a deadlock results when the TERMINATE action itself contains the `kill` command.

Any of the following job control specifications will cause a deadlock:

- ◆ `JOB_CONTROLS=TERMINATE[kill]`
- ◆ `JOB_CONTROLS=TERMINATE[bqueue]`
- ◆ `JOB_CONTROLS=RESUME[bresume]`
- ◆ `JOB_CONTROLS=SUSPEND[bstop]`

Customizing 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 `lshrun` 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 `lshrun` (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=SIGXXXX
- ◆ LSF_NT2UNIX_CTRLB=SIGYYYY

Here, SIGXXXX/SIGYYYY are UNIX signal names such as SIGQUIT, SIGTINT, etc. The conversions will then be: CTRL+C=SIGXXXX and CTRL+BREAK=SIGYYYY.

If both LSF_NT2UNIX_CTRLC and LSF_NT2UNIX_CTRLB are set to the same value (LSF_NT2UNIX_CTRLC=SIGXXXX and LSF_NT2UNIX_CTRLB=SIGXXXX), CTRL+C will be generated on the Windows execution host.

For `bsub -I`, there is no conversion other than the default conversion.

VI

Interactive Jobs

- ◆ [Interactive Jobs with bsub](#) on page 651
- ◆ [Running Interactive and Remote Tasks](#) on page 663

Interactive Jobs with bsub

Contents

- ◆ [About Interactive Jobs](#) on page 651
- ◆ [Submitting Interactive Jobs](#) on page 652
- ◆ [Performance Tuning for Interactive Batch Jobs](#) on page 654
- ◆ [Interactive Batch Job Messaging](#) on page 657
- ◆ [Running X Applications with bsub](#) on page 658
- ◆ [Writing Job Scripts](#) on page 659
- ◆ [Registering utmp File Entries for Interactive Batch Jobs](#) on page 661

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`.

See the *Platform LSF Configuration Reference* for information about configuring interactive queues in the `lsb.queues` file.

Interactive jobs with non-batch utilities

Non-batch utilities such as `lshrun`, `lsgun`, etc., use LIM simple placement advice for host selection when running interactive tasks. For more details on using non-batch utilities to run interactive tasks, see [Running Interactive and Remote Tasks](#) on page 663.

Submitting 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.

Finding 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 -I ls
```

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 `lshrun` or `bsub` LSF commands.

`bsub -lp`

- 1 To submit a batch interactive job by using a pseudo-terminal, use the `bsub -lp` option.

For example:

```
% 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. Some applications such as `vi` for example, require a pseudo-terminal in order to run correctly.

`bsub -ls`

- 1 To submit a batch interactive job and create a pseudo-terminal with shell mode support, use the `bsub -ls` option.

For example:

```
% 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. This option should be specified for submitting interactive shells, or applications which redefine the CTRL-C and CTRL-Z keys (for example, `jove`).

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 lsmake
```

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 -I -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 `lsb.queues`. At the host level, the scheduling conditions are defined as scheduling load thresholds, as described in `lsb.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 `lsb.queues` or as suspending load threshold. At the host level, suspending conditions are defined as stop load threshold as described in `lsb.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 `lsb.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.

See [Load Indices](#) on page 253 for the concept of effective run queue length.

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 and the configuration files take a fraction in the range from 0 to 1, while the `bsub -R` resource requirement string takes an integer from 1-100.

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.

Cluster level

- 1 To enable interactive batch job messaging for all users in the cluster, the LSF administrator configures the following parameters 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.

User level

- 1 To enable messaging for interactive batch jobs, LSF users can define `LSB_INTERACT_MSG_ENH` and `LSB_INTERACT_MSG_INTVAL` as environment variables.

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 -Is -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 -Is sh
Job <2015> is submitted to default queue <normal>.
Job will be scheduled after Fri Nov 19 13:00:00 1999
<<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 -Is 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 >>
```

Running 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 `lsrun` 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.

Writing 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.

Writing 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.

Specifying 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>.
```

UNIX

On UNIX, the `options_file` must be a text file that contains Bourne shell command lines. It cannot be a binary executable file.

Windows

On Windows, the `options_file` must be a text file containing Windows batch file command lines.

Spooling 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.

Redirecting 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.

Specifying 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 that:

- ◆ 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.

Running 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. See [Controlling Execution Environment Using Job Starters](#) on page 629 for more details.

Registering 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 -Ip` or `bsub -Is`. 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.

Registering utmp File Entries for Interactive Batch Jobs

Running Interactive and Remote Tasks

This chapter provides instructions for running tasks interactively and remotely with non-batch utilities such as `lsrun`, `lsgun`, and `lslogin`.

Contents

- ◆ [Running Remote Tasks](#) on page 663
- ◆ [Interactive Tasks](#) on page 666
- ◆ [Load Sharing Interactive Sessions](#) on page 668
- ◆ [Load Sharing X Applications](#) on page 668

Running Remote Tasks

`lsrun` is a non-batch utility to run tasks on a remote host. `lsgun` 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 `lsrun` 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 `lsrun` 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 `lsrun` and `lsgun` options, see the `lsrun(1)` and `lsgun(1)` man pages.

In this section

- ◆ [Run a task on the best available host](#) on page 664
- ◆ [Run a task on a host with specific resources](#) on page 664
- ◆ [Run a task on a specific host](#) on page 665
- ◆ [Run a task by using a pseudo-terminal](#) on page 665
- ◆ [Run the same task on many hosts in sequence](#) on page 665
- ◆ [Run parallel tasks](#) on page 665

- ◆ [Run tasks on hosts specified by a file](#) on page 666

Run a task on the best available host

- 1 To run `mytask` on the best available host, enter:

```
lsrun 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 `lsrun`.

- 1 `lsrun -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 `lsrun` on the command-line. If you do specify resource requirements on the command line, they override the configured resource requirements.

See the *Platform 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 `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.

Run a task on a specific host

- 1 If you want to run your task on a particular host, use the `lsrun -m` option:

```
lsrun -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 `lsrun` 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 `lsrun` specifies that the job should be run using a pseudo-terminal:

```
lsrun -P vi
```

Run the same task on many hosts in sequence

The `lsgrun` 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:

```
lsgrun -m "hostA hostD hostB" cat /tmp/out >> gout
```

Run parallel tasks

`lsgrun -p`

The `-p` option tells `lsgrun` that the task specified should be run in parallel. See `lsgrun(1)` for more details.

- 1 To remove the `/tmp/core` file from all 3 hosts, enter:

```
lsgrun -m "hostA hostD hostB" -p rm -r /tmp/core
```

Run tasks on hosts specified by a file

`lsgrun -f host_file` 1 The `lsgrun -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 `lsrun`, `lsgrun`, etc. To run interactive tasks with these utilities, use the `-i` option.

For more details, see the `lsrun(1)` and `lsgrun(1)` man pages.

In this section

- ◆ [Interactive tasks on remote hosts](#) on page 666
- ◆ [Interactive processing and scheduling policies](#) on page 667
- ◆ [Shared files and user IDs](#) on page 667
- ◆ [Shell mode for remote execution](#) on page 667
- ◆ [Run windows](#) on page 667
- ◆ [Redirect streams to files](#) on page 667

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 `lsrun(1)` man page.

Hiding 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 lsrun -m hostD mytask

or

alias mytask "lsrun -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, `jove`).

The `-S` option of `lshrun`, `ch` or `lshrun` 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 *Platform 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 `lslogin` command.

When you use `lslogin`, LSF automatically chooses the best host and does an `rlogin` to that host.

With no argument, `lslogin` 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 `lslogin -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:

```
lsrun 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 `lsrun` 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.

Setting up an X terminal to start an X session 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`.

Set up Exceed to log on the least loaded host

To set up Exceed to log on to the least loaded host:

- 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:
- 4 Set description to be `Best`.
- 5 Click the Install button 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.

```
lslrun -R "type==any order[cpu:mem:login]" xterm -sb -ls -display your_PC:0.0
```

Start an xterm in Exceed

To start an xterm:

- 1 Double-click the Best icon.

An xterm starts on the least loaded host in the cluster and is displayed on your screen.

Examples

Running 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.

Starting 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
```

VII

Monitoring Your Cluster

- ◆ [Achieving Performance and Scalability](#) on page 673
- ◆ [Reporting](#) on page 691
- ◆ [Event Generation](#) on page 719
- ◆ [Tuning the Cluster](#) on page 723
- ◆ [Authentication and Authorization](#) on page 737
- ◆ [Job Email and Job File Spooling](#) on page 745
- ◆ [Non-Shared File Systems](#) on page 751
- ◆ [Error and Event Logging](#) on page 757
- ◆ [Troubleshooting and Error Messages](#) on page 769

Achieving Performance and Scalability

Contents

- ◆ [Optimizing Performance in Large Sites](#) on page 673
- ◆ [Tuning UNIX for Large Clusters](#) on page 674
- ◆ [Tuning LSF for Large Clusters](#) on page 675
- ◆ [Monitoring Performance Metrics in Real Time](#) on page 686

Optimizing 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.

To target performance optimization to a cluster with 5000 hosts and 100,000 jobs, you must:

- ◆ Configure your operating system. See [Tuning UNIX for Large Clusters](#) on page 674
- ◆ Fine-tune LSF. See [Tuning LSF for Large Clusters](#) on page 675

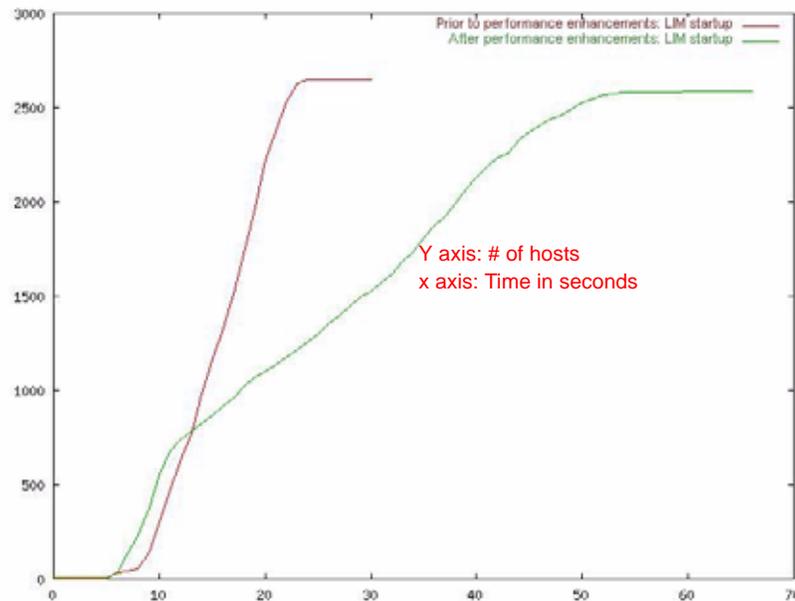
What's new in LSF performance?

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:



Tuning 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.

In this section

- ◆ [Hardware recommendation](#) on page 674
- ◆ [Software requirement](#) on page 674

Hardware recommendation

LSF master host:

- ◆ 4 processors, one each for:
 - ◆ mbatchd
 - ◆ 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
```

Tuning LSF for Large Clusters

To enable and sustain large clusters, you need to tune LSF for efficient querying, dispatching, and event log management.

In this section

- ◆ [Managing scheduling performance](#) on page 675
- ◆ [Limiting the number of batch queries](#) on page 677
- ◆ [Improving the speed of host status updates](#) on page 677
- ◆ [Managing your user's ability to move jobs in a queue](#) on page 678
- ◆ [Managing the number of pending reasons](#) on page 678
- ◆ [Achieving efficient event switching](#) on page 678
- ◆ [Automatic load updating](#) on page 679
- ◆ [Managing the I/O performance of the info directory](#) on page 679
- ◆ [Processor binding for LSF job processes](#) on page 680
- ◆ [Increasing the job ID limit](#) on page 685

Managing 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 `mbatch` 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:

```
badadmin hshutdown
```
- 7 Run `badadmin mbdrestart` to restart the LSF batch daemons on the master host.
- 8 Run `badadmin 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 `mbatchd` is later started by `sbatchd`, 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 `lsb.params`.

Limiting the number of batch queries

In large clusters, job querying can grow very quickly. If your site sees a lot of high traffic job querying, you can tune LSF to limit the number of job queries that `mbatchd` 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 is displayed and `mbatchd` keeps retrying, in one second intervals. If the number of job queries later drops below the limit, `mbatchd` handles the query.

You define the maximum number of concurrent jobs queries to be handled by `mbatchd` in the parameter `MAX_CONCURRENT_JOB_QUERY` in `lsb.params`:

- ◆ If `mbatchd` is using multithreading, a dedicated query port is defined by the parameter `LSB_QUERY_PORT` in `lsf.conf`. When `mbatchd` has a dedicated query port, the value of `MAX_CONCURRENT_JOB_QUERY` sets the maximum number of queries that can be handled by each child `mbatchd` that is forked by `mbatchd`. This means that the total number of job queries handled can be more than the number specified by `MAX_CONCURRENT_JOB_QUERY` (`MAX_CONCURRENT_JOB_QUERY` multiplied by the number of child daemons forked by `mbatchd`).
- ◆ If `mbatchd` is not using multithreading, the value of `MAX_CONCURRENT_JOB_QUERY` sets the maximum total number of job queries that can be handled by `mbatchd`.

Syntax

```
MAX_CONCURRENT_JOB_QUERY=max_query
```

Where:

max_query

Specifies the maximum number of job queries that can be handled by `mbatchd`. 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 `mbatchd`.

```
MAX_CONCURRENT_JOB_QUERY=101
```

Incorrect value. The default value will be used. An unlimited number of job queries will be handled by `mbatchd`.

Improving the speed of host status updates

To improve the speed with which `mbatchd` obtains and reports host status, configure the parameter `LSB_SYNC_HOST_STAT_LIM` in the file `lsb.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, `mbatchd` periodically obtains the host status from the master LIM, and then verifies the status by polling each `sbatchd` at an interval defined by the parameters `MBD_SLEEP_TIME` and `LSB_MAX_PROBE_SBD`.

Managing your user's ability to move jobs in a queue

`JOB_POSITION_CONTROL_BY_ADMIN=Y` allows an LSF administrator to control whether users can use `btop` and `bbot` to move jobs to the top and bottom of queues. When set, only the LSF administrator (including any queue administrators) can use `bbot` and `btop` to move jobs within a queue. A user attempting to use `bbot` or `btop` receives the error "User permission denied."

REMEMBER: You must be an LSF administrator to set this parameter.

Managing the number of pending reasons

For efficient, scalable management of pending reasons, use `CONDENSE_PENDING_REASONS=Y` in `lsb.params` to condense all the host-based pending reasons into one generic pending reason.

If a job has no other main pending reason, `bjobs -p` or `bjobs -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.

Achieving efficient event switching

Periodic switching of the event file can weaken the performance of `mbatchd`, which automatically backs up and rewrites the events file after every 1000 batch job completions. The old `lsb.events` file is moved to `lsb.events.1`, and each old `lsb.events.n` file is moved to `lsb.events.n+1`.

Change the frequency of event switching with the following two parameters in `lsb.params`:

- ◆ `MAX_JOB_NUM` specifies the number of batch jobs to complete before `lsb.events` is backed up and moved to `lsb.events.1`. The default value is 1000
- ◆ `MIN_SWITCH_PERIOD` controls how frequently `mbatchd` checks the number of completed batch jobs

The two parameters work together. Specify the `MIN_SWITCH_PERIOD` value in seconds.

For example:

```
MAX_JOB_NUM=1000
```

```
MIN_SWITCH_PERIOD=7200
```

This instructs `mbatchd` 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, `mbatchd` 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, `mbatchd` does not switch the events file until till the end of the two-hour period

TIP: For large clusters, set the `MIN_SWITCH_PERIOD` to a value equal to or greater than 600. This causes `mbatchd` to fork a child process that handles event switching, thereby reducing the load on `mbatchd`. `mbatchd` 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.

Automatic load updating

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.

Managing the I/O performance of the info directory

In large clusters, there are large numbers of jobs submitted by its users. Since each job generally has a job file, this 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.

This delay is caused by a limit in the total size of files that can reside in a file server directory. This limit is dependent on the file system implementation. A high load on the file server delays the master batch daemon operations, and therefore slows down the overall cluster throughput.

You can prevent this delay by creating and using subdirectories under the parent directory. Each new subdirectory is subject to the file size limit, but the parent directory is not subject to the total file size of its subdirectories. Since the total file size of the `info` directory is divided among its subdirectories, your cluster can process more job operations before reaching the total size limit of the job files.

If your cluster has a lot of jobs resulting in a large `info` directory, you can tune your cluster by enabling LSF to create subdirectories in the `info` directory. Use `MAX_INFO_DIRS` in `lsb.params` to create the subdirectories and enable `mbatchd` to distribute the job files evenly throughout the subdirectories.

Syntax

```
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.

Duplicate event logging

Run `badmin reconfig` to create and use the subdirectories.

NOTE: If you enabled duplicate event logging, you must run `badmin mbdrestart` instead of `badming reconfig` to restart `mbatchd`.

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/logdir/info/0 to
LSB_SHAREDIR/cluster_name/logdir/info/9.
```

Processor binding for LSF job processes

See also [Processor Binding for Parallel Jobs](#) on page 576.

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 physical processors 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 only the number of physical processors (`EGO_DEFINE_NCPUS=procs`).

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 processors, 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.applications`, 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.

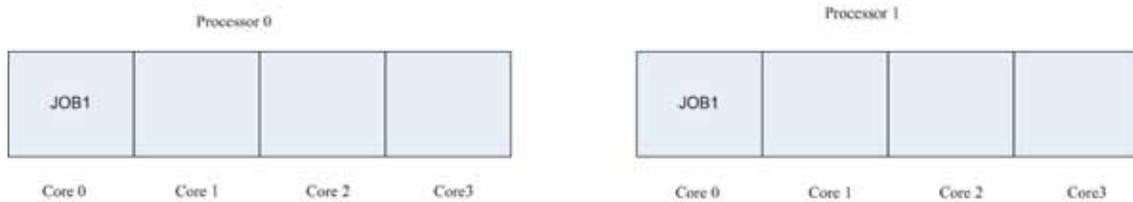
BIND_JOB=BALANCE

The `BIND_JOB=BALANCE` option instructs LSF to bind the job based on the load of the available processors/cores/threads. For each slot:

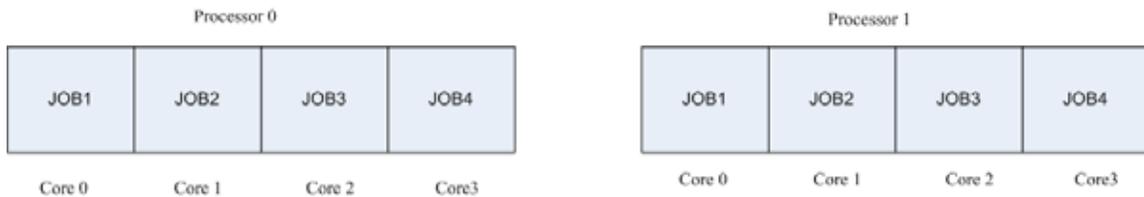
- ◆ If the PCT level is set to processor, the lowest loaded physical processor runs the job.
- ◆ If the PCT level is set to core, the lowest loaded core on the lowest loaded processor runs the job.
- ◆ If the PCT level is set to thread, the lowest loaded thread on the lowest loaded core on the lowest 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:

Tuning LSF for Large Clusters



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

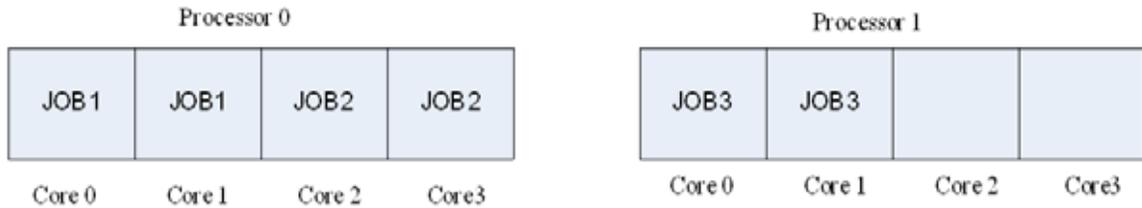
The `BIND_JOB=PACK` option instructs LSF to try to pack all the processes onto a single processor. If this cannot be done, LSF tries to use as few processors as possible. Email is sent to you after job dispatch and when job finishes. If no processors/cores/threads are free (when the `PCT` level is processor/core/thread 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.

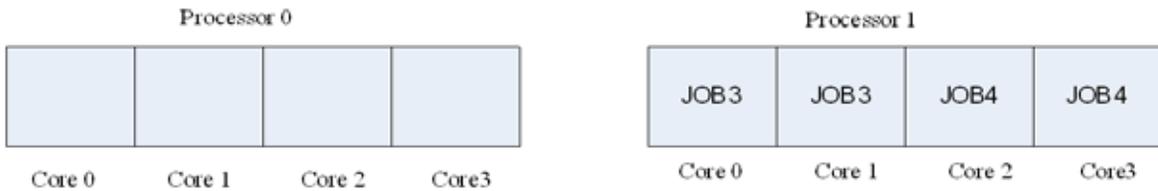
If `PCT` level is processor (default value after installation), there is no difference between `BALANCE` and `PACK`.

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_C PU_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.
- ◆ `badadmin hrestart`
`badadmin 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.
- ◆ `badadmin reconfig`

If the `BIND_JOB` parameter is modified in an application profile, `badmin reconfig` 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 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.applications`)

Define `BIND_JOB` in an application profile configuration in `lsb.applications` 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.applications`, 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.

Increasing the 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.

Monitoring Performance Metrics in Real Time

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, `bjobs` 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 `bjobs` and `bhist` increases to 10 characters.

Monitoring Performance Metrics in Real Time

Enable metric collection

Set `SCHED_METRIC_ENABLE=Y` in `lsb.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 `lsb.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 `lsb.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 `lsb.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 `lsb.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. (`bjobs`, `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

```
File Descriptor Metrics
```

	Free	Used	Total
MBD file descriptor usage	800	424	1024

```
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.

badadmin 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.

$$LastPeriod = \frac{Metric\ Counter\ Value\ of\ Last\ Period}{Sample\ Period\ Interval} \times Sample\ Period$$

- 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.

$$Avg = \frac{Total}{LastPeriodEndTime - SamplingStartTime} \times Sample\ Period$$

Reconfiguring your cluster with performance metric sampling enabled

badadmin mbdrestart If performance metric sampling is enabled dynamically with `badadmin perfmon start`. You must enable it again after running `badadmin mbdrestart`. If performance metric sampling is enabled by default, `StartTime` will be reset to the point `mbatchd` is restarted.

badadmin reconfig If `SCHED_METRIC_ENABLE` and `SCHED_METRIC_SAMPLE_PERIOD` parameters are changed, `badadmin reconfig` is the same as `badadmin mbdrestart`.

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

Only one submission request is counted. Element jobs are counted for jobs submitted, jobs dispatched, and jobs completed.

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 requeue

Requeued 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 requeued with `brequeue`, LSF counts two jobs completed, since requeuing a job first kills the job and later puts the job into pending list. If the job is automatically requeued, 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.

Monitoring Performance Metrics in Real Time

Reporting

Reporting is a feature of Platform LSF. It allows you to look at the overall statistics of your entire cluster. You can analyze the history of hosts, resources, and workload in your cluster to get an overall picture of your cluster's performance.

Contents

- ◆ [Introduction to Reporting](#) on page 691
- ◆ [Getting Started with Standard Reports](#) on page 692
- ◆ [Custom Reports](#) on page 694
- ◆ [System Description](#) on page 698
- ◆ [Reports Administration](#) on page 700
- ◆ [Test the Reporting Feature](#) on page 712
- ◆ [Disable the Reporting Feature](#) on page 713
- ◆ [Move to a Production Database](#) on page 714

Introduction to Reporting

An efficient cluster maximizes the usage of resources while minimizing the average wait time of workload. To ensure that your cluster is running efficiently at all times, you need to analyze the activity within your cluster to see if there are any areas for improvement.

The reporting feature uses the data loader controller service, the job data transformer service, and the data purger service to collect data from the cluster, and to maintain this data in a relational database system. The reporting feature collects the cluster data from a relational database system and displays it in reports either graphically or in tables. You can use these reports to analyze and improve the performance of your cluster, and to troubleshoot configuration problems.

You can access the reporting feature from the Console (Platform Management Console).

Standard and custom reports

Platform has provided a set of standard reports to allow you to immediately analyze your cluster without having to create any new reports. These standard reports provide the most common and useful data to analyze your cluster.

You may also create custom reports to perform advanced queries and reports beyond the data produced in the standard reports.

The database

The reporting feature optionally includes the Apache Derby database, a JDBC-based relational database system. The Derby database is a small-footprint, open source database, and is only appropriate for demo clusters. If you want to use the reporting feature to produce regular reports for a production cluster, you must use a supported commercial database.

The reporting feature supports Oracle 9i, Oracle 10g, and MySQL 5.x databases.

IMPORTANT: The Apache Derby database is not supported for any production clusters.

Getting Started with Standard Reports

For your convenience, Platform has provided several standard reports for you to use. These reports allow you to keep track of some useful statistics in your cluster.

Standard reports overview

Standard reports are based on raw data stored in the relational database, and do not perform any data aggregation or calculations.

The following is a list of the standard reports that are included with the reporting feature. For further details on a report, open its full description as described in [View the full description of a report](#) on page 693.

Table 3: Standard reports

Name	Description	Category
Cluster Availability - EGO	EGO host availability in a cluster.	EGO
Host Resource Usage	Resource usage trends for selected hosts.	EGO
Resource Allocation vs Resource Plan	Actual resource allocation compared to resource plan and unsatisfied resource demand for the selected consumer.	EGO
Active Job States Statistics by Queue	Number of active jobs in each active job state in a selected queue.	LSF
Cluster Availability - LSF	LSF host availability in an LSF cluster.	LSF
Cluster Job Hourly Throughput	Number of submitted, exited, and done jobs in a cluster.	LSF
Cluster Job Slot Utilization	Job slot utilization levels in your cluster.	LSF
Job Slot Usage by Application Tag	Job slots used by applications as indicated by the application tag.	LSF
Performance Metrics	Internal performance metrics trend for a cluster. You can only produce this report if you enabled performance metric collection in your cluster (<code>badmin perfmon start</code>)	LSF

Name	Description	Category
Service Level Agreement (SLA)	Job statistics by job state over time, compared with SLA goals.	LSF
Hourly Desktop Job Throughput	Number of downloaded and completed jobs for each MED host or the entire cluster. You can only produce this report if you use LSF Desktop.	LSF Desktop
Desktop Utilization	Desktop utilization at each MED host or the entire cluster. You can only produce this report if you use LSF Desktop.	LSF Desktop
License Usage	The license usage under License Scheduler. You can only produce this report if you use LSF License Scheduler.	LSF License Scheduler
Jobs Forwarded to Other Clusters	The number of jobs forwarded from your cluster to other clusters. You can only produce this report if you use LSF MultiCluster.	LSF MultiCluster
Jobs Received from Other Clusters	The number of jobs forwarded to your cluster from other clusters. You can only produce this report if you use LSF MultiCluster.	LSF MultiCluster

View the full description of a report

- 1 In the Console, navigate to **Reports**, then **Standard Reports**.
- 2 Click the name of your report to open it.
- 3 Click **Report properties**.

What can I do with standard reports?

Producing reports

The reports stored in the system do not include actual data. Instead, the reports define what data to extract from the system, and how to display it graphically.

Reports need to be produced before you can see the data. When you produce a report, you query the database and extract specific data. The amount of system overhead depends on how much data is in the report.

Standard reports have configurable parameters so you can modify the report and get exactly the data that you want.

Exporting reports

Data expires from the database periodically, so producing a report at a later date may return different data, or return no output at all. After you produce a report, you can keep your results by exporting the report data as comma-separated values in a CSV file. In this way you can preserve your data outside the system and integrate it with external programs, such as a spreadsheet. You can also keep your graphical results by using your browser to save the report results as an image.

Produce a standard report

- 1 In the Console, navigate to **Reports**, then **Standard Reports**.
- 2 Click the name of your report to open it.

- 3 Set the report parameters as desired. Default settings are shown, but you can modify them to suit your needs.
- 4 Click **Produce Report**.
After a short time, the resulting data is displayed graphically.

When you close the report window, you lose the contents of the report unless you export it first.

Export report data

Once you produce a report, exporting is the best way to save the data for future use. You cannot produce the same report at a later date if the data has expired from the database.

-
- 1 In the Console, produce and view your report.
 - 2 Click **Export Report Data**.
 - 3 In the browser dialog, specify the output path and name the exported file.
In the **Save as type** field, specify "CSV".
-

Custom Reports

You can create and use custom reports if the standard reports are insufficient for your needs.

What are custom reports?

While standard reports are provided for your use by Platform, custom reports are reports you create as needed to satisfy specific reporting needs at your site.

Custom reports let you define combinations of data that are not available in the standard reports. Custom report output is always displayed in tabular format.

What can I do with custom reports?

Creating reports

The easiest way to create a custom report is to copy an existing report, then customize the SQL query string as desired. To customize the SQL query string, you may need to refer to the data schema, which describes the organization of information in the relational database. The data schema for each standard report is available in the Console by opening the report and clicking **Help**.

Even if you cannot edit SQL, saving a report as a custom report lets you re-use the report data without having to re-input the parameters in the standard report.

- If the time period is fixed, you get the same data every time you produce the report, but the report will be empty when the data expires from the database.

- If the time period is relative, you can get data for a different time period each time you produce the report.

You can also define custom reports from a blank template and input the SQL query string directly.

When you create custom reports, you can enter a category and use it to group the reports any way you want.

Deleting reports

Unlike standard reports, custom reports can be deleted. You might prefer to rename old reports (by modifying them) instead of deleting them.

Using reports

You produce custom reports and export the data in the same way as standard reports.

Data expires from the database periodically, so producing a report at a later date may return different data, or return no output at all. After you produce a report, you can keep your results by exporting the report data as comma-separated values in a CSV file. In this way you can preserve your data outside the system and integrate it with external programs, such as a spreadsheet. You can also keep your graphical results by using your browser to save the report results as an image.

If you ever want to modify parameters of a custom report, you must edit the SQL query string directly.

Create a custom report from an existing report

This method is convenient because you can extend an existing report. Examine your current standard and custom reports and select one with similar data sources or output to the new report that you want to create.

- 1 In the Console, select the report that you want to copy, with all the parameters configured as you wish to copy them.
- 2 Click **Copy to New Custom Report**.
- 3 Edit the report properties and query string as desired.
 - a In the `Report properties` section, you should give the new report a unique name. You can also modify the report summary, description, and category.
 - b In the `Report query` section, you can modify the SQL query directly.

To edit the SQL query, you will need to know about the data schema of the database. For further information on the data schema, refer to **Platform LSF Reports Data Schema** in the Platform LSF Knowledge Center.
 - c To validate your SQL query string and ensure that your report delivers the appropriate results, click **Produce Report**.

This will actually produce the report, so you might want to limit your testing to a small set of data.

You can continue to edit your SQL query string and test the results of your report until you are ready to save it.
- 4 To finish, click **Create**.

To access your new custom report, navigate to **Reports** then **Custom Reports**.

Create a new custom report

Prerequisites: You must be able to construct valid query strings with Structured Query Language (SQL).

-
- 1 In the Console, navigate to **Reports** then **Custom Reports**.
 - 2 Select **Global Actions > Create Custom Report**.
 - 3 Define the report properties and query string as desired.
 - a In the `Report properties` section, specify the report name, summary, description, and category.
 - b In the `Report query` section, input your SQL query string.
For further information on the data schema, refer to **Platform LSF Reports Data Schema** in the Platform LSF Knowledge Center.
 - c To validate your SQL query string and ensure that your report delivers the appropriate results, click **Produce Report**.
This will actually produce the report, so you might want to limit your testing to a small set of data.
You can continue to edit your SQL query string and test the results of your report until you are ready to save it.
 - 4 To finish, click **Create**.

To access your new custom report, navigate to **Reports** then **Custom Reports**.

Modify a custom report

-
- 1 In the Console, navigate to **Reports** then **Custom Reports**.
 - 2 Click the name of your report.
 - 3 Modify the report properties and query string as desired.
 - a Edit the report properties and SQL query string.
For further information on the data schema, refer to **Platform LSF Reports Data Schema** in the Platform LSF Knowledge Center.
 - b To validate your SQL query string and ensure that your report delivers the appropriate results, click **Produce Report**.
This will actually produce the report, so you might want to limit your testing to a small set of data.
You can continue to edit your SQL query string and test the results of your report until you are ready to save it.
 - 4 To confirm your changes, click **Save**.
-

Produce a custom report

- 1 In the Console, navigate to **Reports** then **Custom Reports**.
- 2 Click the name of your report to open it.
- 3 Click **Produce Report**.

After a short time, the resulting data is displayed in tabular format.

When you close the report window, you will lose the contents of the report unless you export it first.

Export report data

Once you produce a report, exporting is the best way to save the data for future use. You cannot produce the same report at a later date if the data has expired from the database.

- 1 In the Console, produce and view your report.
- 2 Click **Export Report Data**.
- 3 In the browser dialog, specify the output path and name the exported file. In the **Save as type** field, specify "CSV".

Delete a custom report

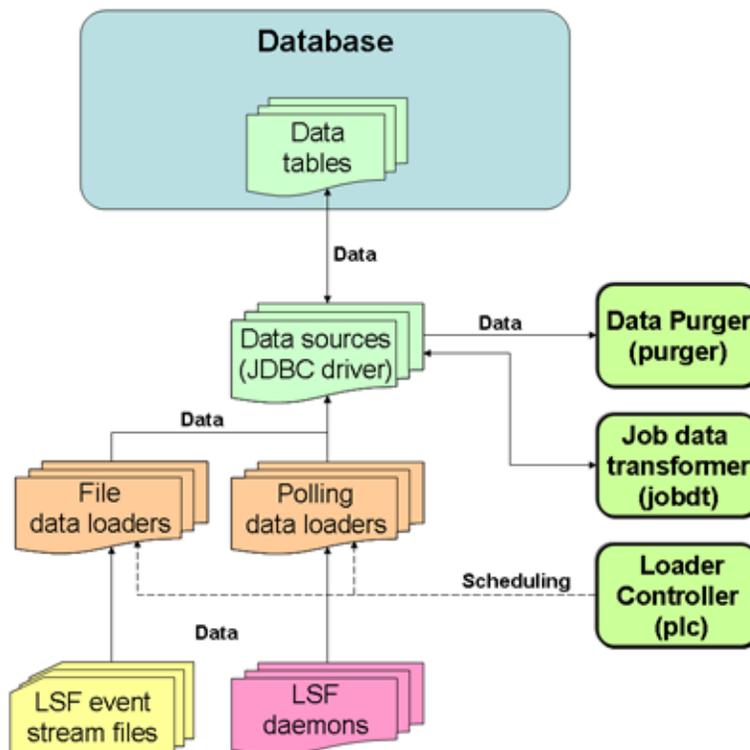
- 1 In the Console, navigate to **Reports** then **Custom Reports**.
- 2 Locate your report in the list.
- 3 Select **Actions > Delete Report**.

System Description

The reporting feature is built on top of the Platform Enterprise Reporting Framework (PERF) architecture. This architecture defines the communication between your EGO cluster, relational database, and data sources via the PERF Loader Controller (PLC). The loader controller is the module that controls multiple loaders for data collection.

PERF architecture

The following diagram illustrates the PERF architecture as it relates to your cluster, reporting services, relational database, and data loaders.



Data loaders

The reporting feature collects cluster operation data using data loaders to load data into tables in a relational database. The data loaders connect to the database using a JDBC driver. The data loaders handle daylight savings automatically by using GMT time when collecting data.

Default data loaders

The following are lists of the data loaders and default behavior:

Table 4: LSF data loaders

Data loader name	Data type	Data gathering interval	Data loads to	Loader type
License Scheduler (bldloader)	license usage	5 minutes	BLD_LICUSAGE	polling
Desktop job (desktopjobdataloader) - Linux hosts only	job completion log	1 day	ACTIVE_DESKTOP_JOBDATA	polling
Desktop client (desktopclientdataloader) - Linux hosts only	client status (data from the WSClientStatus file)	10 minutes	ACTIVE_DESKTOP_SED_CLIENT	polling
Desktop active event (desktopeventloader) - Linux hosts only	downloaded and reported jobs (data from the event.log file)	each time an event is logged in event.log.	ACTIVE_DESKTOP_ACEVENT for each event of type 2 (REPORT_JOB) and type 4 (COMPLETE_JOB)	polling
Host metrics (hostmetricsloader)	host-related metrics	5 minutes	RESOURCE_METRICS RESOURCES_RESOURCE_METRICS	polling
Host properties (hostpropertiesloader)	resource properties	1 hour	LSF_RESOURCE_PROPERTIES	polling
Bhosts (lsfbhostsloader)	host utilization and state-related	5 minutes	LSF_BHOSTS	polling
LSF events (lsfeventsloader)	events with a job ID, performance events, resource events	5 minutes	LSB_EVENTS LSB_EVENTS_EXECHOSTLIST LSF_PERFORMANCE_METRIC	file
Resource properties (lsfresproploader)	shared resource properties	1 hour	LSF_RESOURCE_PROPERTIES	polling
SLA (lsfslaloader)	SLA performance	5 minutes	LSF_SLA	polling
Shared resource usage (sharedresusageloader)	shared resource usage	5 minutes	SHARED_RESOURCE_USAGE SHARED_RESOURCE_USAGE_HOSTLIST	polling

Table 5: EGO data loaders

Data loader name	Data type	Data gathering interval	Data loads to	Loader type
Consumer resource (egoconsumerresloader)	resource allocation	5 minutes	CONSUMER_DEMAND CONSUMER_RESOURCE_ALLOCATION CONSUMER_RESOURCELIST	polling
Dynamic metric (egodynamicresloader)	host-related dynamic metric	5 minutes	RESOURCE_METRICS RESOURCES_RESOURCE_METRICS	polling
EGO allocation events (egoeventsloader)	resource allocation	5 minutes	ALLOCATION_EVENT	file
Static attribute (egostaticresloader)	host-related static attribute	1 hour	ATTRIBUTES_RESOURCE_METRICS RESOURCE_ATTRIBUTES	polling

System services

The reporting feature has system services, including the Derby service if you are running a demo database. If your cluster has PERF controlled by EGO, these services are run as EGO services. Each service uses one slot on a management host.

Loader controller The loader controller service (`plc`) controls the data loaders that collect data from the system and writes the data into the database.

Data purger The data purger service (`purger`) maintains the size of the database by purging old records from the database. By default, the data purger purges all data that is older than 14 days, and purges data every day at 12:30am.

Job data transformer The job data transformer service (`jobdt`) converts raw job data in the relational database into a format usable by the reporting feature. By default, the job data transformer converts job data every hour at thirty minutes past the hour (that is, at 12:30am, 1:30am, and so on throughout the day).

Derby database If you are running a demo database, the Derby database (`derbydb`) stores the cluster data. When using a supported commercial database, the Derby database service no longer runs as an EGO service.

Reports Administration

What do I need to know?

Reports directories The reporting feature resides in various `perf` subdirectories within the LSF directory structure. This document uses `LSF_TOP` to refer to the top-level LSF installation directory. The reporting feature directories include the following:

Table 6: LSF reporting directory environment variables in UNIX

Directory name	Directory description	Default file path
<code>\$PERF_TOP</code>	Reports framework directory	<code>LSF_TOP/perf</code>
<code>\$PERF_CONFDIR</code>	Configuration files	<code>LSF_TOP/conf/perf/cluster_name/conf</code>
<code>\$PERF_LOGDIR</code>	Log files	<code>LSF_TOP/log/perf</code>
<code>\$PERF_WORKDIR</code>	Working directory	<code>LSF_TOP/work/perf</code>
<code>\$PERF_DATADIR</code>	Data directory	<code>LSF_TOP/work/cluster_name/perf/data</code>

Table 7: LSF reporting directory environment variables in Windows

Directory name	Directory description	Default file path
<code>%PERF_TOP%</code>	Reports framework directory	<code>LSF_TOP\perf</code>
<code>%PERF_CONFDIR%</code>	Configuration files	<code>LSF_TOP\conf\perf\cluster_name\conf</code>
<code>%PERF_LOGDIR%</code>	Log files	<code>LSF_TOP\log\perf</code>
<code>%PERF_WORKDIR%</code>	Working directory	<code>LSF_TOP\work\perf</code>
<code>%PERF_DATADIR%</code>	Data directory	<code>LSF_TOP\work\cluster_name\perf\data</code>

Reporting services The reporting feature uses the following services.

- ◆ Job data transformer (`jobdt`)

- ◆ Loader controller (plc)
- ◆ Data purger (purger)

The Derby demo database uses the `derbydb` service.

If your cluster has PERF controlled by EGO, these services are run as EGO services. You need to stop and restart a service after editing its configuration files. If you are disabling the reporting feature, you need to disable automatic startup of these services, as described in [Disable automatic startup of the reporting services](#) on page 705.

Log files for these services are available in the `PERF_LOGDIR` directory. There are seven logging levels that determine the detail of messages recorded in the log files. In decreasing level of detail, these are ALL (all messages), TRACE, DEBUG, INFO, WARN, ERROR, FATAL, and OFF (no messages). By default, all service log files log messages of INFO level or higher (that is, all INFO, WARN, ERROR, and FATAL messages). You can change the logging level of the `plc` service using the loader controller client tool as described in [Dynamically change the log level of your loader controller log file](#) on page 705, or the logging level of the other services as described in [Change the log level of your log files](#) on page 706.

Job data transformer

The job data is logged in the relational database in a raw format. At regular intervals, the job data transformer converts this data to a format usable by the reporting feature. By default, the data transformer converts the job data every hour at thirty minutes past the hour (that is, at 12:30am, 1:30am, and so on throughout the day).

To reschedule the transformation of data from the relational database to the reporting feature, you can change the data transformer schedule as described in [Change the data transformer schedule](#) on page 709.

If your cluster has PERF controlled by EGO, you can edit the `jobdt.xml` configuration file, but you need to restart the `jobdt` service and EGO on the master host after editing the file. The `jobdt.xml` file is located in the EGO service directory:

- ◆ UNIX: `LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services`
- ◆ Windows: `LSF_CONFDIR\ego\cluster_name\eservice\esc\conf\services`

Loader controller

The loader controller manages the data loaders. By default, the loader controller manages the following data loaders:

- ◆ `bldloader` (License Scheduler data loader)
- ◆ `desktopjobdataloader` (Desktop job data loader)
- ◆ `desktopclientdataloader` (Desktop client data loader)
- ◆ `desktopeventloader` (Desktop active event data loader)
- ◆ `egoconsumerresloader` (consumer resource data loader)
- ◆ `egodynamicresloader` (dynamic metric data loader)
- ◆ `egoeventsloader` (EGO allocation events data loader)
- ◆ `egostaticresloader` (static attribute data loader)

- ◆ `lsfbhostsloader` (bhosts data loader)
- ◆ `lsfeventsloader` (LSF events data loader)
- ◆ `lsfslaloader` (SLA data loader)
- ◆ `lsfresproploader` (LSF resource properties data loader)
- ◆ `sharedresusageloader` (share resource usage data loader)

You can view the status of the loader controller service using the loader controller client tool as described in [View the status of the loader controller](#) on page 705.

Log files for the loader controller and data loaders are available in the `PERF_LOGDIR` directory. There are logging levels that determine the detail of messages recorded in the log files. In decreasing level of detail, these are ALL (all messages), TRACE, DEBUG, INFO, WARN, ERROR, FATAL, and OFF (no messages). By default, all service log files log messages of INFO level or higher (that is, all INFO, WARN, ERROR, and FATAL messages). You can change the logging level of the `plc` service using the loader controller client tool as described in [Dynamically change the log level of your loader controller log file](#) on page 705, or the logging level of the data loaders using the client tool as described in [Dynamically change the log level of your data loader log files](#) on page 706.

To balance data accuracy with computing power, you can change how often the data loaders collect data by changing the frequency of data collection per loader, as described in [Change the frequency of data collection](#) on page 711. To reduce the amount of unwanted data logged in the database, you can also disable individual data loaders from collecting data, as described in [Disable data collection for individual data loaders](#) on page 711.

If you edit any `plc` configuration files, you need to restart the `plc` service.

If your cluster has PERF controlled by EGO, you can edit the `plc_service.xml` service configuration file, but you must restart the `plc` service and EGO on the master host after editing the file. The `plc_service.xml` file is located in the EGO service directory:

- ◆ UNIX: `LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services`
- ◆ Windows: `LSF_CONFDIR\ego\cluster_name\eservice\esc\conf\services`

Data purger

The relational database needs to be kept to a reasonable size to maintain optimal efficiency. The data purger manages the database size by purging old data at regular intervals. By default, the data purger purges records older than 14 days at 12:30am every day.

To reschedule the purging of old data, you can change the purger schedule, as described in [Change the data purger schedule](#) on page 708. To reduce or increase the number of records in the database, you can change the duration of time that records are stored in the database, as described in [Change the default record expiry time](#) on page 710. If there are specific tables that are containing too much or too little data, you can also change the duration of time that records are stored in each individual table within the database, as described in [Change the record expiry time per table](#) on page 710.

If you edit any purger configuration files, you need to restart the purger service.

If your cluster has PERF controlled by EGO, you can edit the `purger_service.xml` service configuration file, but you must restart the `purger` service and EGO on the master host after editing the file. The `purger_service.xml` file is located in the EGO service directory:

- ◆ UNIX: `LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services`
- ◆ Windows: `LSF_CONFDIR\ego\cluster_name\eservice\esc\conf\services`

Derby database

The Derby database uses the `derbydb` service. You need to restart this service manually whenever you change the Derby database settings. The Derby database is only appropriate for demo clusters. To use the reporting feature to produce regular reports for a production cluster, you must move to a production database using a supported commercial database and disable automatic startup of the `derbydb` service.

Event data files

The events logger stores event data in event data files. The EGO allocation event data file (for EGO-enabled clusters only) is named `ego.stream` by default and has a default maximum size of 10MB. The LSF event data file is named `lsb.stream` by default and has a default maximum size of 100MB. When a data file exceeds this size, the events logger archives the file and creates a new data file.

The events logger only maintains one archive file and overwrites the old archive with the new archive. The default archive file name is `ego.stream.0` for EGO and `lsb.stream.0` for LSF. The two LSF files are located in `LSF_TOP/work/cluster_name/logdir/stream` by default, and the two EGO files are located in `LSF_TOP/work/cluster_name/ego/data` by default. The event data loaders read both the data files and the archive files.

If your system logs a large number of events, you should increase the maximum file size to see more archived event data. If your disk space is insufficient for storing the four files, you should decrease the maximum file size, or change the file path to a location with sufficient storage space. Change the disk usage of your LSF event data files as described in [Change the disk usage of LSF event data files](#) on page 707 or the file path as described in [Change the location of the LSF event data files](#) on page 707. Change the disk usage or file path of your EGO allocation event data files as described in [Change the disk usage of EGO allocation event data files](#) on page 707.

You can manage your event data files by editing the system configuration files. Edit `ego.conf` for the EGO allocation event data file configuration and `lsb.params` for the LSF event data file configuration.

Administering reports

Determine if your cluster is EGO-enabled and has PERF controlled by EGO

You need to determine whether your cluster is EGO-enabled and has PERF controlled by EGO in order to determine which command you use to manage the reporting services.

- 1 In the command console, run `egosh service list` to see the list of EGO services.
 - ❖ If you see a list of services showing that the reporting services are `STARTED`, your cluster is EGO-enabled and has PERF controlled by EGO. The reporting services are run as EGO services, and you use `egosh service` to manage the reporting services.
 - ❖ If you see a list of services showing that the reporting services are not `STARTED`, your cluster is EGO-enabled but does not have PERF controlled by EGO. You use `perfadmin` to manage the reporting services.
 - ❖ If you get an error running `egosh service list`, your cluster is not EGO-enabled and therefore does not have PERF controlled by EGO. You use `perfadmin` to manage the reporting services.

Stop or restart reporting services (PERF controlled by EGO)

Prerequisites: Your cluster must have PERF controlled by EGO.

Stop or restart the `derbydb` (if you are using the Derby demo database), `jobdt`, `plc`, and `purger` services. If your cluster has PERF controlled by EGO, the reporting services are run as EGO services, and you use the `egosh service` command to stop or restart these services.

- 1 In the command console, stop the service by running `egosh service stop`.
`egosh service stop service_name`
- 2 If you want to restart the service, run `egosh service start`.
`egosh service start service_name`

Stop or restart reporting services (PERF not controlled by EGO)

Prerequisites: Your cluster must have PERF not controlled by EGO.

Stop or restart the `derbydb` (if you are using the Derby demo database), `jobdt`, `plc`, and `purger` services. If your cluster does not have PERF controlled by EGO, you use the `perfadmin` command to stop or restart these services.

- 1 In the command console, stop the service by running `perfadmin stop`.
`perfadmin stop service_name`

- 2 If you want to restart the service, run `perfadmin start`.
`perfadmin start service_name`

Disable automatic startup of the reporting services

Prerequisites: Your cluster must be EGO-enabled.

When disabling the reporting feature, disable automatic startup of the `derbydb` (if you are using the Derby demo database), `jobdt`, `plc`, and `purger` services. When moving from the Derby demo database to a production database, disable automatic startup of the `derbydb` service.

Disable automatic startup of these services by editing their service configuration files (`jobdt.xml`, `plc_service.xml`, `purger_service.xml`, and `derby_service.xml` for the `jobdt`, `plc`, `purger`, and `derbydb` services, respectively).

- 1 In the command console, open the EGO service directory.
 - ◆ UNIX: `cd`
`LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services`
 - ◆ Windows: `cd` `LSF_CONFDIR\ego\cluster_name\eservice\esc\conf\services`
- 2 Edit the service configuration file and change the service type from automatic to manual.
In the `<sc:StartType>` tag, change the text from `AUTOMATIC` to `MANUAL`.
- 3 Stop the service.
- 4 In the command console, restart EGO on the master host to activate these changes.
`egosh ego restart master_host_name`

View the status of the loader controller

Use the loader controller client tool to view the status of the loader controller.

- 1 Launch the loader controller client tool with the `-s` option.
 - ◆ In UNIX, run `$PERF_TOP/version/bin/plcclient.sh -s`.
 - ◆ In Windows, run `%PERF_TOP%\version\bin\plcclient -s`.

Dynamically change the log level of your loader controller log file

Use the loader controller client tool to dynamically change the log level of your `plc` log file if it does not cover enough detail, or covers too much, to suit your needs.

If you restart the `plc` service, the log level of your `plc` log file will be set back to the default level. To retain your new log level, change the level of your `plc` log file as described in [Change the log level of your log files](#) on page 706.

- 1 Launch the loader controller client tool with the `-l` option.
 - ❖ In UNIX, run `$PERF_TOP/bin/plcclient.sh -l log_level`.
 - ❖ In Windows, run `$PERF_TOP\bin\plcclient -l log_level`.

In decreasing level of detail, the log levels are ALL (for all messages), TRACE, DEBUG, INFO, WARN, ERROR, FATAL, and OFF (for no messages).

Dynamically change the log level of your data loader log files

Use the loader controller client tool to dynamically change the log level of your individual data loader log files if they do not cover enough detail, or cover too much, to suit your needs.

If you restart the `plc` service, the log level of your data loader log files will be set back to the default level. To retain your new log level, change the level of your data loader log files as described in [Change the log level of your log files](#) on page 706.

- 1 If you are using the default configuration file, launch the loader controller client tool with the `-n` and `-l` options.
 - ❖ In UNIX, run `$PERF_TOP/version/bin/plcclient.sh -n data_loader_name -l log_level`.
 - ❖ In Windows, run `%PERF_TOP%\version\bin\plcclient -n data_loader_name -l log_level`.

Refer to [Loader controller](#) on page 701 for a list of the data loader names.

In decreasing level of detail, the log levels are ALL (for all messages), TRACE, DEBUG, INFO, WARN, ERROR, FATAL, and OFF (for no messages).

Change the log level of your log files

Change the log level of your log files if they do not cover enough detail, or cover too much, to suit your needs.

- 1 Edit the `log4j.properties` file, located in the reports configuration directory (`PERF_CONFDIR`).
- 2 Navigate to the section representing the service you want to change, or to the default loader configuration if you want to change the log level of the data loaders, and look for the `log4j.logger.com.platform.perf` variable.

For example, to change the log level of the data purger log files, navigate to the following section, which is set to the default `INFO` level:

```
# Data purger ("purger") configuration
log4j.logger.com.platform.perf.purger=INFO,
com.platform.perf.purger
```

- 3 Change the `log4j.logger.com.platform.perf` variable to the new logging level.
In decreasing level of detail, the valid values are ALL (for all messages), TRACE, DEBUG, INFO, WARN, ERROR, FATAL, and OFF (for no messages). The services or data loaders only log messages of the same or lower level of detail as specified by the `log4j.logger.com.platform.perf` variable. Therefore, if you change the log level to ERROR, the service or data loaders will only log ERROR and FATAL messages.
For example, to change the data purger log files to the ERROR log level:

```
# Data purger ("purger") configuration
log4j.logger.com.platform.perf.purger=ERROR,
com.platform.perf.purger
```
- 4 Restart the service that you changed (or the `plc` service if you changed the data loader log level).

Change the disk usage of LSF event data files

If your system logs a large number of events and you have sufficient disk space, increase the disk space allocated to the LSF event data files.

- 1 Edit `lsb.params` and specify or change the `MAX_EVENT_STREAM_SIZE` parameter.

```
MAX_EVENT_STREAM_SIZE = integer
```

If unspecified, this is 1024 by default. Change this to the new desired file size in MB.
The recommended size is 2000 MB.
- 2 In the command console, reconfigure the master host to activate this change.

```
badmin reconfig
```

Change the location of the LSF event data files

If your system logs a large number of events and your do not have enough disk space, move the LSF event data files to another location.

- 1 Edit `lsb.params` and specify or change the `EVENT_STREAM_FILE` parameter.

```
EVENT_STREAM_FILE = file_path
```

If unspecified, this is
`LSF_TOP/work/cluster_name/logdir/stream/lsb.stream` by default.
- 2 In the command console, reconfigure the master host to activate this change.

```
badmin reconfig
```
- 3 Restart the `plc` service on the master host to activate this change.

Change the disk usage of EGO allocation event data files

Prerequisites: Your cluster must be EGO-enabled.

If your system logs a large number of events, increase the disk space allocated to the EGO allocation event data files. If your disk space is insufficient, decrease the space allocated to the EGO allocation event data files or move these files to another location.

- 1 Edit `ego.conf`.
 - a To change the size of each EGO allocation event data file, specify or change the `EGO_DATA_MAXSIZE` parameter.
`EGO_DATA_MAXSIZE = integer`
 If unspecified, this is 10 by default. Change this to the new desired file size in MB.
 - b To move the files to another location, specify or change the `EGO_DATA_FILE` parameter.
`EGO_DATA_FILE = file_path`
 If unspecified, this is
`LSF_TOP/work/cluster_name/ego/data/ego.stream` by default.
- 2 In the command console, restart EGO on the master host to activate this change.
`egosh ego restart master_host_name`

Change the data purger schedule

Prerequisites: Your cluster must be EGO-enabled.

To reschedule the deletion of old data, change the time in which the data purger deletes the old data.

- 1 Edit `purger_service.xml` in the EGO service directory.
 - ◆ **UNIX:**
`LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services`
 - ◆ **Windows:** `LSF_CONFDIR\ego\cluster_name\eservice\esc\conf\services`
 - 2 Navigate to `<ego:Command>` with the `-t` parameter in the purger script.
 - ◆ In UNIX, this is `<ego:Command> ...purger.sh -t ...`
 - ◆ In Windows, this is `<ego:Command> ...purger.bat -t ...`

By default, the data purger is scheduled to delete old data at 12:30am every day.
 - 3 Change the `-t` parameter in the data purger script to the new time (`-t new_time`).
- You can change the data purger schedule to a specific daily time, or at regular time intervals, in minutes, from when the `purger` service first starts up.
- For example, to change the schedule of the data purger:

- ◆ To delete old data at 11:15pm every day:
`<ego:Command> ...purger... -t 23:15`
 - ◆ To delete old data every 12 hours from when the purger service first starts up:
`<ego:Command> ...purger... -t *[12]`
- 4 In the command console, restart EGO on the master host to activate these changes.
`egosh ego restart master_host_name`
 - 5 Restart the purger service.
-

Change the data transformer schedule

Prerequisites: Your cluster must be EGO-enabled.

To have reschedule the transformation of data from the relational database to the reporting feature, change the time in which the data transformer converts job data.

- 1 Edit `jobdt.xml` in the EGO service directory.
 - ◆ UNIX:
`LSF_CONFDIR/ego/cluster_name/eservice/esc/conf/services`
 - ◆ Windows: `LSF_CONFDIR\ego\cluster_name\eservice\esc\conf\services`
- 2 Navigate to `<ego:Command>` with the `-t` parameter in the purger script.
 - ◆ In UNIX, this is `<ego:Command> ...jobdt.sh -t ...`
 - ◆ In Windows, this is `<ego:Command> ...jobdt.bat -t ...`

By default, the data transformer converts the job data every hour at thirty minutes past the hour (that is, at 12:30am, 1:30am, and so on throughout the day).

- 3 Change the `-t` parameter in the data transformer script to the new time (`-t new_time`).

You can change the data transformer schedule to a specific daily time, a specific hourly time, or at regular time intervals, in minutes or hours, from when the `jobdt` service first starts up.

For example, to change the schedule of the data transformer:

- ◆ To convert job data at 10:20pm every day:
`<ego:Command> ...jobdt... -t 22:20`
- ◆ To convert job data at the 25th minute of every hour:
`<ego:Command> ...jobdt... -t *:25`
- ◆ To convert job data every fifteen minutes from when the `jobdt` service first starts up:
`<ego:Command> ...jobdt... -t *:[15]`

- ◆ To convert job data every two hours from when the `jobdt` service first starts up:

```
<ego:Command> ...jobdt... -t *[2]
```

- 4 In the command console, restart EGO on the master host to activate these changes.

```
egosh ego restart master_host_name
```

- 5 Restart the `jobdt` service.
-

Change the default record expiry time

To reduce or increase the number of records stored in the database, change the duration of time that a record is stored in the database before it is purged. This applies to all tables in the database unless you also specify the record expiry time in a particular table.

- 1 Edit the purger configuration files for your data loaders.

- ◆ For EGO data loaders, edit `purger_ego_rawdata.xml`.
- ◆ For LSF data loaders, edit `purger_lsf_basic_rawdata.xml`.

The purger configuration files are located in the `purger` subdirectory of the reports configuration directory:

- ◆ UNIX: `$PERF_CONFDIR/purger`
- ◆ Windows: `%PERF_CONFDIR%\purger`

- 2 In the `<TableList>` tag, edit the `Duration` attribute to your desired time in days, up to a maximum of 31 days.

For example, to have the records purged after 7 days:

```
<TableList Duration="7">
```

By default, the records are purged after 14 days.

- 3 Restart the purger service.
-

Change the record expiry time per table

To reduce or increase the number of records stored in the database for a particular table, change the duration of time that a record is stored in the database per table before it is purged. The duration only applies to this particular table.

- 1 Edit the purger configuration files for your data loaders.

- ◆ For EGO data loaders, edit `purger_ego_rawdata.xml`.
- ◆ For LSF data loaders, edit `purger_lsf_basic_rawdata.xml`.

The purger configuration files are located in the `purger` subdirectory of the reports configuration directory:

- ◆ UNIX: `$PERF_CONFDIR/purger`
- ◆ Windows: `%PERF_CONFDIR%\purger`

- 2 Navigate to the specific `<Table>` tag with the `TableName` attribute matching the table that you want to change.

For example:

```
<Table TableName="RESOURCE_METRICS"
TimestampColumn="TIME_STAMP" ... />
```

- 3 Add or edit the `Duration` attribute with your desired time in days, up to a maximum of 31 days.

For example, to have the records in this table purged after 10 days:

```
<Table TableName="RESOURCE_METRICS"
TimestampColumn="TIME_STAMP" Duration="10" ... />
```

- 4 Restart the purger service.
-

Change the frequency of data collection

To change how often the data loaders collect data, change the frequency of data collection per loader.

- 1 Edit the `plc` configuration files for your data loaders.

- ❖ For EGO data loaders, edit `plc_ego_rawdata.xml`.
- ❖ For LSF data loaders, edit `plc_lsf_basic_rawdata.xml`.

The `plc` configuration files are located in the `plc` subdirectory of the reports configuration directory:

- ◆ UNIX: `$PERF_CONFDIR/plc`
- ◆ Windows: `%PERF_CONFDIR%\plc`

- 2 Navigate to the specific `<DataLoader>` tag with the `Name` attribute matching the data loader that you want to change.

For example:

```
<DataLoader Name="egodynamicresloader" Interval="300" ... />
```

- 3 Add or edit the `Interval` attribute with your desired time in seconds.

For example, to have this plug-in collect data every 200 seconds:

```
<DataLoader Name="egodynamicresloader" Interval="200" ... />
```

- 4 Restart the `plc` service.
-

Disable data collection for individual data loaders

To reduce unwanted data from being logged in the database, disable data collection for individual data loaders.

- 1 Edit the `plc` configuration files for your data loaders.

- ❖ For EGO data loaders, edit `plc_ego_rawdata.xml`.
- ❖ For LSF data loaders, edit `plc_lsf_basic_rawdata.xml`.

Test the Reporting Feature

The `plc` configuration files are located in the `plc` subdirectory of the reports configuration directory:

- ◆ UNIX: `$PERF_CONFDIR/plc`
 - ◆ Windows: `%PERF_CONFDIR%\plc`
- 2 Navigate to the specific `<DataLoader>` tag with the `Name` attribute matching the data loader that you want to disable.
For example:

```
<DataLoader Name="egodynamicresloader" ... Enable="true" .../>
```
 - 3 Edit the `Enable` attribute to `"false"`.
For example, to disable data collection for this plug-in:

```
<DataLoader Name="egodynamicresloader" ... Enable="false" ... />
```
 - 4 Restart the `plc` service.
-

Test the Reporting Feature

Verify that components of the reporting feature are functioning properly.

- 1 Check that the reporting services are running.
 - ❖ If your cluster has PERF controlled by EGO, run `egosh service list`.
 - ❖ If your cluster has PERF not controlled by EGO, run `perfadmin list`.
- 2 Check that there are no error messages in the reporting logs.
 - a View the loader controller log file.
 - ◆ UNIX: `$PERF_LOGDIR/plc.log.host_name`
 - ◆ Windows: `%PERF_LOGDIR%/plc.log.host_name.txt`
 - b Verify that there are no `ERROR` messages and that, in the `DataLoader Statistics` section, there are data loader statistics messages for the data loaders in the last hour.

You need to find statistics messages for the following data loaders:

- ◆ `bldloader`
- ◆ `desktopjobdataloader`
- ◆ `desktopclientdataloader`
- ◆ `desktopenventloader`
- ◆ `lsfbhostsloader`
- ◆ `lsfeventsloader`
- ◆ `lsfslloader`
- ◆ `lsfresproploader`
- ◆ `sharedresusageloader`
- ◆ EGO data loaders (for EGO-enabled clusters only):
 - ❖ `egoconsumerresloader`
 - ❖ `egodynamicresloader`

- ❖ `egoeventsloader`
 - ❖ `egostaticresloader`
- c** View the data purger and data loader log files and verify that there are no ERROR messages in these files.
- You need to view the following log files (*PERF_LOGDIR* is *LSF_LOGDIR/perf*):
- ◆ `PERF_LOGDIR/dataloader/bldloader.host_name.log`
 - ◆ `PERF_LOGDIR/dataloader/desktopjobdataloader.host_name.log`
 - ◆ `PERF_LOGDIR/dataloader/desktopclientdataloader.host_name.log`
 - ◆ `PERF_LOGDIR/dataloader/desktopeventloader.host_name.log`
 - ◆ `PERF_LOGDIR/jobdt.host_name.log`
 - ◆ `PERF_LOGDIR/dataloader/lsfbhostsloader.host_name.log`
 - ◆ `PERF_LOGDIR/dataloader/lsfeventsloader.host_name.log`
 - ◆ `PERF_LOGDIR/dataloader/lsfslaloder.host_name.log`
 - ◆ `PERF_LOGDIR/purger.host_name.log`
 - ◆ `PERF_LOGDIR/dataloader/lsfresproploader.host_name.log`
 - ◆ `PERF_LOGDIR/dataloader/sharedresusageloder.host_name.log`
 - ◆ **EGO data loader log files (EGO-enabled clusters only):**
 - ❖ `PERF_LOGDIR/dataloader/egoconsumerresloader.host_name.log`
 - ❖ `PERF_LOGDIR/dataloader/egodynamicresloader.host_name.log`
 - ❖ `PERF_LOGDIR/dataloader/egoeventsloader.host_name.log`
 - ❖ `PERF_LOGDIR/dataloader/egostaticresloader.host_name.log`
- 3** Check the report output.
- a Produce a standard report.
 - b Verify that the standard report produces a chart or table with data for your cluster.

Postrequisites: If you were not able to verify that these components are functioning properly, identify the cause of these problems and correct them.

Disable the Reporting Feature

Prerequisites: You must have `root` or `lsfadmin` access in the master host.

- 1** Disable the LSF events data logging.
 - a Define or edit the `ENABLE_EVENT_STREAM` parameter in the `lsb.params` file to disable event streaming.


```
ENABLE_EVENT_STREAM = N
```


You need to copy the Oracle JDBC driver to the following directories:

- ◆ `$PERF_TOP/version/lib`
- ◆ `LSF_TOP/gui/version/tomcat/common/lib`

- 5 Configure your database connection.
- 6 Restart the reporting services.
Restart the `jobdt`, `plc`, and `purger` services.
- 7 If your cluster is EGO-enabled, restart the Platform Management Console.

NOTE: The Platform Management Console will be unavailable during this step.

- a In the command console, stop the `WEBGUI` service.

```
egosh service stop WEBGUI
```

- b Restart the `WEBGUI` service.

```
egosh service start WEBGUI
```

The report data will now be loaded into the production database and the Console will use the data in this database.

Create an Oracle database schema

Prerequisites: The Oracle database is properly configured and running:

- ◆ You have a user name, password, and URL to access the database server.
- ◆ You installed the latest JDBC driver (`ojdbc14.jar` or newer) for the Oracle database. This driver is available from the following URL:
http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html

- 1 In the command console, open the EGO database schema directory.
 - ◆ UNIX: `cd $PERF_TOP/ego/version/DBschema/Oracle`
 - ◆ Windows: `cd %PERF_TOP%\ego\version\DBschema\Oracle`
- 2 Run the script to create the EGO database schema.

```
sqlplus user_name/password@connect_string @egodata.sql data_tablespace
index_tablespace
```

where

- ◆ *user_name* is the user name on the database server.
 - ◆ *password* is the password for this user name on the database server.
 - ◆ *connect_string* is the named SQLNet connection for this database.
 - ◆ *data_tablespace* is the name of the tablespace where you intend to store the table schema.
 - ◆ *index_tablespace* is the name of the tablespace where you intend to store the index.
- 3 In the command console, open the LSF database schema directory.
 - ◆ UNIX: `cd $PERF_TOP/lsf/version/DBschema/Oracle`

Move to a Production Database

- ◆ Windows: `cd %PERF_TOP%\lsf\version\DBschema\Oracle`

4 Run the script to create the LSF database schema.

```
sqlplus user_name/password@connect_string @lsfdata.sql data_tablespace
index_tablespace
```

where

- ◆ *user_name* is the user name on the database server.
- ◆ *password* is the password for this user name on the database server.
- ◆ *connect_string* is the named SQLNet connection for this database.
- ◆ *data_tablespace* is the name of the tablespace where you intend to store the table schema.
- ◆ *index_tablespace* is the name of the tablespace where you intend to store the index.

Create a MySQL database schema

Prerequisites: The MySQL database is properly configured and running:

- ◆ You have a user name, password, and URL to access the database server.
- ◆ You installed the latest JDBC driver (`mysql-connector-java-3.1.12-bin.jar` or newer) for the MySQL database. This driver is available from the following URL:

<http://dev.mysql.com/downloads/>

1 In the command console, open the EGO database schema directory.

- ◆ UNIX: `cd $PERF_TOP/ego/version/DBschema/MySQL`
- ◆ Windows: `cd %PERF_TOP%\ego\version\DBschema\MySQL`

2 Run the scripts to create the EGO database schema.

```
mysql --user=user_name --password=password --database=report_database < egodata.sql
```

where

- ◆ *user_name* is the user name on the database server.
- ◆ *password* is the password for this user name on the database server.
- ◆ *report_database* is the name of the database to store the report data.

3 In the command console, open the LSF database schema directory.

- ◆ UNIX: `cd $PERF_TOP/lsf/version/DBschema/MySQL`
- ◆ Windows: `cd %PERF_TOP%\lsf\version\DBschema\MySQL`

4 Run the scripts to create the LSF database schema.

```
mysql --user=user_name --password=password --database=report_database < lsfdata.sql
```

where

- ◆ *user_name* is the user name on the database server.
- ◆ *password* is the password for this user name on the database server.
- ◆ *report_database* is the name of the database to store the report data.

Configure the database connection

Prerequisites: You have a user name, password, and URL to access the database server.

Launch the database configuration tool to configure your database connection.

- 1 If you connected to the UNIX host via `telnet` and are running `xserver` on a local host, set your display environment.

Test your display by running `xclock` or another X-Windows application.

If the application displays, your display environment is already set correctly; otherwise, you need to set your display environment.

- ❖ For `csh` or `tcsh`:

```
setenv DISPLAY hostname:0.0
```

- ❖ For `sh`, `ksh`, or `bash`:

```
DISPLAY=hostname:0.0
```

```
export DISPLAY
```

where `hostname` is your local host.

- 2 Launch the database configuration tool.

- ❖ In UNIX, run `$PERF_TOP/version/bin/dbconfig.sh`.

- ❖ In Windows, run `%PERF_TOP%\version\bin\dbconfig`.

- 3 In the **User ID** and **Password** fields, specify the user account name and password with which to connect to the database and to create your database tablespaces.

This user account must have been defined in your database application, and must have read and write access to the database tables.

- 4 In the **JDBC driver** field, select the driver for your commercial database.

- 5 In the **JDBC URL** field, enter the URL for your database.

This should be similar to the format given in **Example URL format**.

- 6 In the **Maximum connections** field, specify the maximum allowed number of concurrent connections to the database server.

This is the maximum number of users who can produce reports at the same time.

Move to a Production Database

Event Generation

Contents

- ◆ [Event Generation](#) on page 719
- ◆ [Enabling event generation](#) on page 719
- ◆ [Events list](#) on page 720
- ◆ [Arguments passed to the LSF event program](#) on page 720

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.

Enabling event generation

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. See [Arguments passed to the LSF event program](#) on page 720 and [Events list](#) on page 720 for more information.
- 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 `lsadmin reconfig` and `badmin reconfig`.

Events list

The following daemon operations cause `mbatchd` 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 `sbatchd` goes down (detected by `mbatchd`).
- 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 `mbatchd` comes up and is ready to schedule jobs (detected by `mbatchd`).
- 8 `mbatchd` goes down (detected by `mbatchd`).
- 9 `mbatchd` receives a reconfiguration request and is being reconfigured (detected by `mbatchd`).
- 10 `LSB_SHAREDIR` becomes full (detected by `mbatchd`).

Arguments passed to the 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 `HostA` in `Cluster1`, the function returns:

```
xxx Cluster1 1 HostA
```

The custom LSF event program can interpret or ignore these arguments.

Arguments passed to the LSF event program

47

Tuning the Cluster

Contents

- ◆ [Tuning LIM on page 724](#)
- ◆ [Improving performance of mbatchd query requests on UNIX on page 731](#)

Tuning 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.

You can also change default LIM behavior and pre-select hosts to be elected master to improve performance.

In this section

- ◆ [Adjusting LIM Parameters](#) on page 724
- ◆ [Load Thresholds](#) on page 724
- ◆ [Changing Default LIM Behavior to Improve Performance](#) on page 726

Adjusting 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.

See [Load Thresholds](#) on page 609 for more details.

In this section

- ◆ [Load indices that affect LIM performance](#) on page 725
- ◆ [Comparing LIM load thresholds](#) on page 725
- ◆ [If LIM often reports a host as busy](#) on page 726
- ◆ [If interactive jobs slow down response](#) on page 726
- ◆ [Multiprocessor systems](#) on page 726

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

For more details on load indices see [Load Indices](#) on page 253.

Comparing LIM load thresholds

To tune LIM load thresholds, compare the output of `lsload` to the thresholds reported by `lshosts -l`.

The `lsload` and `lsmom` commands display an asterisk * next to each load index that exceeds its threshold.

Example

Consider the following output from `lshosts -l` and `lsload`:

```
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
...
```

Tuning LIM

LOAD_THRESHOLDS:

r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
-	3.5	-	-	15	-	-	-	-	2M	1M

lsload

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.

If LIM often 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.

If interactive jobs slow down response

If you find that interactive jobs slow down system response too much 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. See [Load Indices](#) on page 253 for the concept of effective and normalized run queue lengths.

Changing Default LIM Behavior to Improve Performance

You may want to change the default LIM behavior in the following cases:

- ◆ In very large sites. As the size of the cluster becomes large (500 hosts or more), reconfiguration of the cluster causes each LIM to re-read the configuration files. This can take quite some time.

- ◆ In sites where each host in the cluster cannot share a common configuration directory or exact replica.

In this section

- ◆ [Default LIM behavior](#) on page 727
- ◆ [Changing Default LIM Behavior to Improve Performance](#) on page 726
- ◆ [Reconfiguration and LSF_MASTER_LIST](#) on page 727
- ◆ [How LSF works with LSF_MASTER_LIST](#) on page 728
- ◆ [Considerations](#) on page 729

Default LIM behavior

By default, each LIM running in an LSF cluster must read the configuration files `lsf.shared` and `lsf.cluster.cluster_name` to obtain information about resource definitions, host types, host thresholds, etc. This includes master and slave LIMs.

This requires that each host in the cluster share a common configuration directory or an exact replica of the directory.

Change default LIM behavior

The parameter `LSF_MASTER_LIST` in `lsf.conf` allows you to identify for the LSF system which hosts can become masters. Hosts not listed in `LSF_MASTER_LIST` will be considered as slave-only hosts and will never be considered to become master.

Set LSF_MASTER_LIST (lsf.conf)

- 1 Edit `lsf.conf` and set the parameter `LSF_MASTER_LIST` to indicate hosts that are candidates to become the master host. For example:

```
LSF_MASTER_LIST="hostA hostB hostC"
```

The order in which you specify hosts in `LSF_MASTER_LIST` is the preferred order for selecting hosts to become the master LIM.
- 2 Save your changes.
- 3 Reconfigure the cluster

```
lsadmin reconfig
badmim mbdrestart.
```

Reconfiguration and LSF_MASTER_LIST

If you change LSF_MASTER_LIST

Whenever you change the parameter `LSF_MASTER_LIST`, reconfigure the cluster with `lsadmin reconfig` and `badmim mbdrestart`.

If you change `lsf.cluster.cluster_name` or `lsf.shared`

If you make changes that do not affect load report messages such as adding or removing slave-only hosts, you only need to restart the LIMs on all master candidates with the command `lsadmin limrestart` and the specific host names.

For example:

```
lsadmin limrestart hostA hostB hostC
```

If you make changes that affect load report messages such as load indices, you must restart all the LIMs in the cluster. Use the command `lsadmin reconfig`.

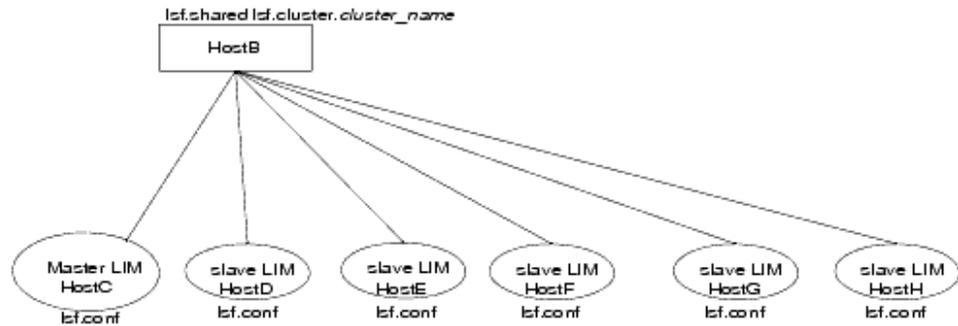
How LSF works with LSF_MASTER_LIST

LSF_MASTER_LIST undefined

In this example, `lsf.shared` and `lsf.cluster.cluster_name` are shared among all LIMs through an NFS file server. The preferred master host is the first available server host in the cluster list in `lsf.cluster.cluster_name`.

Any slave LIM can become the master LIM. Whenever you reconfigure, all LIMs read `lsf.shared` and `lsf.cluster.cluster_name` to get updated information.

For this example, slave LIMs read local `lsf.conf` files.



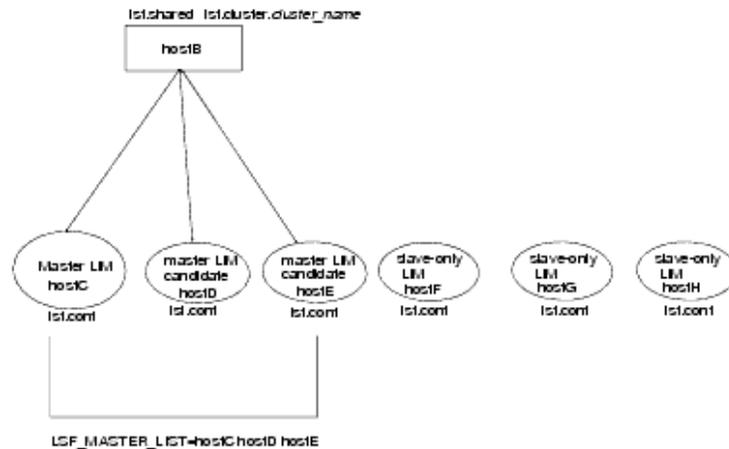
LSF_MASTER_LIST defined

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.



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 not defined

When `LSF_MASTER_LIST` is not defined, LSF rejects candidate master hosts from the cluster if their `lsf.cluster.cluster_name` and `lsf.shared` files are different from the files of the elected master. Even if only comment lines are different, hosts are rejected.

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 `lsf.cluster.cluster_name` and `lsf.shared` are identical for all hosts and restart all LIMs in the cluster with the command:

```
lsadmin limrestart all
```

LSF_MASTER_LIST defined

When `LSF_MASTER_LIST` is defined, 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
```

LSF_MASTER_LIST defined, and master host goes down

If `LSF_MASTER_LIST` is defined and 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 will have the same situation as described in [LSF_MASTER_LIST defined](#) on page 728. You will 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.

Improving 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 `bjobs` 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.

In this section

- ◆ [How mbatchd works without multithreading](#) on page 731
- ◆ [Configure mbatchd to use multithreading](#) on page 731
- ◆ [Set a query-dedicated port for mbatchd](#) on page 733
- ◆ [Specify an expiry time for child mbatchds \(optional\)](#) on page 733
- ◆ [Specify hard CPU affinity](#) on page 733
- ◆ [Configure mbatchd to push new job information to child mbatchd](#) on page 734

How mbatchd works 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.

Servicing 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*, or a job 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 bjobs 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 lsb.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 lsf.conf.
See [Set a query-dedicated port for mbatchd](#) on page 733.
 - 2 Optional: Set an interval of time to indicate when a new child mbatchd is to be forked by setting MBD_REFRESH_TIME in lsb.params. The default value of MBD_REFRESH_TIME is 5 seconds, and valid values are 0-300 seconds.
See [Specify an expiry time for child mbatchds \(optional\)](#) on page 733.
 - 3 Optional: Use NEWJOB_REFRESH=Y in lsb.params to enable a child mbatchd to get up to date new job information from the parent mbatchd.
See [Configure mbatchd to push new job information to child mbatchd](#) on page 734.
-

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 `lsf.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 `lsf.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 `lsf.conf` file.
- 5 Reconfigure the cluster:


```
badmin mbdrestart
```

Specify an expiry time for child mbatchds (optional)

Use `MBD_REFRESH_TIME` in `lsb.params` to define how often `mbatchd` forks a new child `mbatchd`.

- 1 Log on to the host as the primary LSF administrator.
- 2 Edit `lsb.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 `lsb.params` file.
- 5 Reconfigure the cluster as follows:


```
badmin reconfig
```

Specify hard CPU affinity

You can specify the master host CPUs on which `mbatchd` child query processes can run (hard CPU affinity). This improves `mbatchd` scheduling and dispatch performance by binding query processes to specific CPUs so that higher priority `mbatchd` processes can run more efficiently.

When you define this parameter, LSF runs `mbatchd 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 `mbatchd 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 `mbatchd 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 `mbatchd 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 `bjobs`.
 - 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

Prerequisites: `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 `mbatchd` to get up to date new job information from the parent `mbatchd`.

When `NEWJOB_REFRESH=Y` the parent `mbatchd` pushes new job information to a child `mbatchd`. Job queries with `bjobs` display new jobs submitted after the child `mbatchd` was created.

-
- 1 Log on to the host as the primary LSF administrator.
 - 2 Edit `lsb.params`.
 - 3 Add `NEWJOB_REFRESH=Y`.
You should set `MBD_REFRESH_TIME` in `lsb.params` to a value greater than 10 seconds.
 - 4 Save the `lsb.params` file.
 - 5 Reconfigure the cluster as follows:
`badmin reconfig`
-

Improving performance of mbatchd query requests on UNIX

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.

Contents

- ◆ [Authentication options](#) on page 737
- ◆ [Authorization options](#) on page 739

Authentication options

During LSF installation, the authentication method is set to external authentication (`eauth`), which offers the highest level of security. To change the authentication method used by LSF, configure the parameter `LSF_AUTH` in `lsf.conf`.

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 the <i>Platform LSF Configuration Reference</i>.
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>.

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.

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.

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
<code>eauth</code>	◆ External authentication failed
<code>identd</code>	◆ The identification daemon is not available on the local or submitting host
<code>setuid</code>	◆ The LSF applications are not installed <code>setuid</code> ◆ The NFS directory is mounted with the <code>nosuid</code> option
<code>ruserok</code>	◆ 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

Authorization options

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.

Authorization options

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

Enabling 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 the *Platform LSF Configuration Reference*.

Specifying a user account

To change the user account for ...	Define the parameter ...	In the file ...
eauth	LSF_EAUTH_USER	lsf.sudoers
eexec	LSF_EEXEC_USER	
Pre- and post execution commands	LSB_PRE_POST_EXEC_USER	

Controlling user access to LSF resources and functionality

If you want to ...	Define ...	In the file ...	Section ...
Specify the user accounts with cluster administrator privileges	ADMINISTRATORS	lsf.cluster. <i>cluster_name</i>	ClusterAdmins
Allow the <code>root</code> user to run jobs on a remote host	LSF_ROOT_REX	lsf.conf	N/A
Allow specific user accounts to use <code>@</code> for host redirection with <code>lscsh</code>	LSF_SHELL_AT_USERS	lsf.conf	N/A
Allow user accounts other than <code>root</code> to start LSF daemons	LSF_STARTUP_USERS LSF_STARTUP_PATH	lsf.sudoers	N/A

TIP: For information about how to configure the LSF daemon startup control feature, see the *Platform LSF Configuration Reference*.

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.	The user should <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"> ◆ <code>resControl: operation permission denied, uid = <uid></code> 	<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 <code>root</code> user to make RES control requests, define <code>LSF_ROOT_REX</code> in <code>lsf.conf</code>.</p>
<ul style="list-style-type: none"> ◆ <code>do_restartReq: Failed to get hData of host <host_name>/<host_addr></code> 	<p><code>mbatchd</code> received a request from <code>sbatchd</code> on host <i>host_name</i>, but that host is not known to <code>mbatchd</code>. Either</p> <ul style="list-style-type: none"> ◆ The configuration file has been changed but <code>mbatchd</code> has not been reconfigured. ◆ <i>host_name</i> is a client host but <code>sbatchd</code> is running on that host. 	<p>To reconfigure <code>mbatchd</code>, run the command <code>badadmin reconfig</code></p> <p>To shut down <code>sbatchd</code> on <i>host_name</i>, run the command <code>badadmin hshutdown host_name</code></p>

Authorization options

Job Email and Job File Spooling

Contents

- ◆ [Mail Notification When a Job Starts](#) on page 745
- ◆ [File Spooling for Job Input, Output, and Command Files](#) on page 748

Mail Notification When a Job Starts

When a batch job completes or exits, LSF by default sends a job report by electronic mail 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.

Mail Notification When a Job Starts

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.

See [Remote File Access](#) on page 752 for an example of copying the output file back to the submission host if the job executes on a file system that is not shared between the submission and execution hosts.

Disabling job email

If you do not want job output to be sent by mail, 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, if you want no job output or email at all, 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.

Specifying 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/lsf_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/lsf_out/ myjob
```

The following command creates the directory `C:\lsf\work\lsf_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:\lsf\work\lsf_err\\ myjob
```

For more information

See the *Platform LSF Configuration Reference* for information about the `LSB_MAILSIZE` environment variable and the `LSB_MAILTO`, `LSB_MAILSIZE_LIMIT` parameters in `lsf.conf`, and `JOB_SPOOL_DIR` in `lsb.params`.

File Spooling for Job Input, Output, and Command Files

About job file spooling

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.

Specifying job input files

Use the `bsub -i input_file` and `bsub -is input_file` commands to get the standard input for the job from the file path name specified by *input_file*. The path can be an absolute path or a relative path to the current working directory. The input file can be any type of file, though it is typically a shell script text file.

LSF first checks the execution host to see if the input file exists. If the file exists on the execution host, LSF uses this file as the input file for the job.

If the file does not exist on the execution host, 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.

The `-is` option of `bsub` 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.

Use the `bsub -is` command if you need to change the original input file before the job completes. Removing or modifying the original input file does not affect the submitted job.

Unless you use `-is`, 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). The special characters `%J` and `%I` are not valid with the `-is` option.

Specifying a job command file (`bsub -Zs`)

Use the `bsub -Zs` command 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.

Use the `bmod -Zs` command if you need 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.

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.

About the job spooling directory (`JOB_SPOOL_DIR`)

If `JOB_SPOOL_DIR` is specified in `lsb.params`:

- ◆ The job input file for `bsub -is` is spooled to `JOB_SPOOL_DIR/lsf_indir`. If the `lsf_indir` 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/lsf_cmddir`. If the `lsf_cmddir` directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.

The `JOB_SPOOL_DIR` directory should be a shared directory accessible from the master host and the submission host. The directory must be readable and writable by the job submission users.

`JOB_SPOOL_DIR` can be any valid path up to a maximum length up to 4094 characters on UNIX and Linux or up to 255 characters for Windows.

`JOB_SPOOL_DIR` must be readable and writable by the job submission user, and it must be shared by the master host and the submission host. If the specified directory is not accessible or does not exist, `bsub -is` cannot write to the default directory `LSB_SHAREDIR/cluster_name/lsf_cmddir` and the job will fail.

Except for `bsub -is` and `bsub -Zs`, if `JOB_SPOOL_DIR` is not accessible or does not exist, output is spooled to the default job output directory `.lsbatch`.

For `bsub -is` 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 -is` and `bsub -Zs` cannot write to the default directory and the job will fail.

If `JOB_SPOOL_DIR` is not specified in `lsb.params`:

- ◆ The job input file for `bsub -is` is spooled to `LSB_SHAREDIR/cluster_name/lsf_indir`. If the `lsf_indir` directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.

Modifying the job input file

- ◆ The job command file for `bsub -Zs` is spooled to `LSB_SHAREDIR/cluster_name/lsf_cmddir`. If the `lsf_cmddir` 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, but do not specify `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 `lsf_indir` and `lsf_cmddir` directories under `LSB_SHAREDIR/cluster_name` that are readable and writable by all job submission users.

Modifying the job input file

Use the `-i` and `-is` options of `bmod` to specify a new job input file. The `-in` and `-isn` options cancel the last job input file modification made with either `-i` or `-is`.

Modifying the job command file

Use the `-Z` and `-Zs` options of `bmod` to modify the job command file specification. `-Z` modifies a command submitted without spooling, and `Zs` modifies a spooled command file. The `-Zsn` option of `bmod` cancels the last job command file modification made with `-Zs` and uses the original spooled command.

For more information

See the *Platform LSF Command Reference* for more information about the `bsub` and `bmod` commands.

See the Platform LSF Configuration Reference for more information about the `JOB_SPOOL_DIR` parameter in `lsb.params`, and the `LSF_TMPDIR` environment variable.

Non-Shared File Systems

Contents

- ◆ [About Directories and Files](#) on page 751
- ◆ [Using LSF with Non-Shared File Systems](#) on page 752
- ◆ [Remote File Access](#) on page 752
- ◆ [File Transfer Mechanism \(lsrctp\)](#) on page 754

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 `automount`.
- ◆ 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. See [Remote File Access](#) on page 752 for more information.

Some networks do not share files between hosts. LSF can still be used on these networks, with reduced fault tolerance. See [Using LSF with Non-Shared File Systems](#) on page 752 for information about using LSF in a network without a shared file system.

Using LSF with Non-Shared File Systems

LSF installation

To install LSF on a cluster without shared file systems, follow the complete installation procedure on every host to install all the binaries, man pages, and configuration files.

Configuration files

After you have installed LSF on every host, you must update the configuration files on all hosts so that they contain the complete cluster configuration. Configuration files must be the same on all hosts.

Master host

You must choose one host to act as the LSF master host. LSF configuration files and working directories must be installed on this host, and the master host must be listed first in `lsf.cluster.cluster_name`.

You can use the parameter `LSF_MASTER_LIST` in `lsf.conf` to define which hosts can be considered to be elected master hosts. In some cases, this may improve performance.

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.

Fault tolerance

Some 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

Using LSF 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, `sbatchd` 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. See [File Transfer Mechanism \(lsrcp\)](#) on page 754 for more information.

bsub -f

The `-f "[local_file operator [remote_file]]"` option to the `bsub` command copies a file between the submission host and the execution host. To specify multiple files, repeat the `-f` option.

local_file

File name on the submission host

remote_file

File name on the execution host

The files `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.

operator

Operation to perform on the file. The operator must be surrounded by white space. 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 ><

If the submission and execution hosts have different directory structures, you must ensure that the directory where `remote_file` and `local_file` will be placed exists. 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.

You should 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. Be careful not to overwrite an existing file in your home directory.

File Transfer Mechanism (lsrscp)

bsub -i

If the input file specified with `bsub -i` is not found on the execution host, the file is copied from the submission host using the LSF remote file access facility and is removed from the execution host after the job finishes.

bsub -o and bsub -e

The output files specified with the `-o` and `-e` arguments to `bsub` are created on the execution host, and are not copied back to the submission host by default. You can 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
```

Example

To submit `myjob` 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
```

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.

See [Remote File Access](#) on page 752 for more information about using `bsub -f`.

Limitations to lsrscp

Because LSF client hosts do not run RES, jobs that are submitted from client hosts should only specify `bsub -f` if `rscp` is allowed. You must set up the permissions for `rscp` 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 [Authorization options](#) on page 739 for more information.

Workarounds

In these situations, use the following workarounds:

rcp on UNIX

If `lsrnp` 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`.

See the `rcp(1)` and `rsh(1)` man pages for more information on using the `rcp` command.

Custom file transfer mechanism

You can replace `lsrnp` with your own file transfer mechanism as long as it supports the same syntax as `lsrnp`. This might be done to take advantage of a faster interconnection network, or to overcome limitations with the existing `lsrnp`. `sbatchd` looks for the `lsrnp` executable in the `LSF_BINDIR` directory as specified in the `lsf.conf` file.

File Transfer Mechanism (Isrctp)

Error and Event Logging

Contents

- ◆ [System Directories and Log Files](#) on page 757
- ◆ [Managing Error Logs](#) on page 758
- ◆ [System Event Log](#) on page 760
- ◆ [Duplicate Logging of Event Logs](#) on page 761
- ◆ [LSF Job Termination Reason Logging](#) on page 762
- ◆ [Understanding LSF job exit codes](#) on page 766

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.events.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`.

Log levels and descriptions

Number	Level	Description
0	<code>LOG_EMERG</code>	Log only those messages in which the system is unusable.
1	<code>LOG_ALERT</code>	Log only those messages for which action must be taken immediately.
2	<code>LOG_CRIT</code>	Log only those messages that are critical.
3	<code>LOG_ERR</code>	Log only those messages that indicate error conditions.
4	<code>LOG_WARNING</code>	Log only those messages that are warnings or more serious messages. This is the default level of debug information.
5	<code>LOG_NOTICE</code>	Log those messages that indicate normal but significant conditions or warnings and more serious messages.
6	<code>LOG_INFO</code>	Log all informational messages and more serious messages.
7	<code>LOG_DEBUG</code>	Log all debug-level messages.
8	<code>LOG_TRACE</code>	Log all available messages.

Support for 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.

Support for 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 *Platform 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.

Managing 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 7. 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

Prerequisites: 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 `lsfshutdown` 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 `lsfstartup` on the host.

View the number of file descriptors remaining

Prerequisites: The performance monitoring (perfmon) 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 `badmin perfmon 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.

Error logging

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`.

System Event Log

The LSF daemons keep an event log in the `lsb.events` file. The `mbatchd` daemon uses this information to recover from server failures, host reboots, and `mbatchd` restarts. The `lsb.events` file is also used by the `bhist` command to display detailed information about the execution history of batch jobs, and by the `badmin` command to display the operational history of hosts, queues, and daemons.

By default, `mbatchd` automatically backs up and rewrites the `lsb.events` file after every 1000 batch job completions. This value is controlled by the `MAX_JOB_NUM` parameter in the `lsb.params` file. The old `lsb.events` file is moved to

`lsb.events.1`, and each old `lsb.events.n` file is moved to `lsb.events.n+1`. LSF never deletes these files. If disk storage is a concern, the LSF administrator should arrange to archive or remove old `lsb.events.n` files periodically.

CAUTION: Do not remove or modify the current `lsb.events` file. Removing or modifying the `lsb.events` file could cause batch jobs to be lost.

Duplicate Logging of Event Logs

To recover from server failures, host reboots, or `mbatchd` 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) 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 `mbatchd`.

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 `mbatchd` 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`.

Setting an event update interval

If NFS traffic is too high and you want to reduce network traffic, 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.

Automatic archiving and duplicate logging

Event logs

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

To enable duplicate logging, set `LSB_LOCALDIR` in `lsf.conf` to a directory on the first master host (the first host configured in `lsf.cluster.cluster_name`) that will be used to store the primary copies of `lsb.events`. This directory should only exist on the first master host.

- 1 Edit `lsf.conf` and set `LSB_LOCALDIR` to a local directory that exists only on the first master host.
- 2 Use the commands `lsadmin reconfig` and `badmin mbdrestart` to make the changes take effect.

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 <srunk sleep 100000>
```

```
Thu Sep 16 15:22:09: Submitted from host <hostA>, CWD <${HOME}>;
```

```
Thu Sep 16 15:22:20: Dispatched to 4 Hosts/Processors <4*hostA>;
```

```
Thu Sep 16 15:22:20: slurm_id=21793;ncpus=4;slurm_alloc=n[13-14];
```

```
Thu Sep 16 15:23:21: 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

Termination reasons displayed by bacct

When LSF detects that a job is terminated, `bacct -l` displays 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 bkill -b	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_SLURM	Job terminated abnormally in SLURM (node failure)	22
TERM_SWAP	Job killed after reaching LSF swap usage limit	20
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 `lsb.acct` and termination reason keywords are mapped in `lsbatch.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 `lsb.acct` is set to `TERM_OWNER` or `TERM_ADMIN`.

Example output of `bacct` and `bhist`

Example termination cause	Termination reason in <code>bacct -l</code>	Example <code>bhist</code> output
<code>bkill -s KILL</code> <code>bkill job_ID</code>	Completed <exit>; <code>TERM_OWNER</code> or <code>TERM_ADMIN</code>	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;
<code>bkill -r</code>	Completed <exit>; <code>TERM_FORCE_ADMIN</code> or <code>TERM_FORCE_OWNER</code> when <code>sbatchd</code> is not reachable. Otherwise, <code>TERM_USER</code> or <code>TERM_ADMIN</code>	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;
<code>TERMINATE_WHEN</code>	Completed <exit>; <code>TERM_LOAD/</code> <code>TERM_WINDOWS/</code> <code>TERM_PREEMPT</code>	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>; <code>TERM_MEMLIMIT</code>	Thu Mar 13 19:31:13: Exited by signal 2. The CPU time used is 0.1 seconds;
Run limit reached	Completed <exit>; <code>TERM_RUNLIMIT</code>	Thu Mar 13 20:18:32: Exited by signal 2. The CPU time used is 0.1 seconds.
CPU limit	Completed <exit>; <code>TERM_CPULIMIT</code>	Thu Mar 13 18:47:13: Exited by signal 24. The CPU time used is 62.0 seconds;
Swap limit	Completed <exit>; <code>TERM_SWAPLIMIT</code>	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>; <code>TERM_ZOMBIE</code>	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;
<code>brequeue -r</code>	For each <code>requeue</code> , Completed <exit>; <code>TERM_REQUEUE_ADMIN</code> or <code>TERM_REQUEUE_OWNER</code>	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;
<code>bchkpnt -k</code>	On the first run: Completed <exit>; <code>TERM_CHKPNT</code>	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;

Example termination cause	Termination reason in <code>bacct -l</code>	Example <code>bhist</code> output
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;

Understanding 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.

TIP: 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
```

Example

The following example shows a job that exited with exit code 139, which means that the job was terminated with signal 11 (SIGSEGV on most systems, $139-128=11$). This means that the application had a core dump.

```
bjobs -l 2012
```

```
Job <2012>, User , Project , Status , Queue , Command
Fri Dec 27 22:47:28: Submitted from host , CWD <${HOME}>;
Fri Dec 27 22:47:37: Started on , Execution Home , Execution CWD ;
Fri Dec 27 22:48:02: Exited with exit code 139. 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	-		-								

Understanding LSF job exit codes

Troubleshooting and Error Messages

Contents

- ◆ [Shared File Access](#) on page 770
- ◆ [Common LSF Problems](#) on page 771
- ◆ [Error Messages](#) on page 778
- ◆ [Setting Daemon Message Log to Debug Level](#) on page 785
- ◆ [Setting Daemon Timing Levels](#) on page 788

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 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.
-

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.

Common LSF Problems

This section lists some other common problems with the LIM, RES, `mbatchd`, `sbatchd`, and interactive applications.

Most problems are due to incorrect installation or configuration. Check the error log files; 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 `lslload` 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 `lslload` 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      localhost      myhostname
```

The following example correctly sets the master LIM IP address:

```
127.0.0.1      localhost
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      localhost ipv6-localhost ipv6-loopback

fe00::0      ipv6-localnet

ff00::0      ipv6-mcastprefix
ff02::1      ipv6-allnodes
ff02::2      ipv6-allrouters
ff02::3      ipv6-allhosts
```

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.

```
badmin 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.

See [Host not used by LSF](#) on page 774.

sbatchd starts but mbatchd does not

- 1 Check whether LIM is running. You can test this by running the `lsid` 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. See [Registering Service Ports](#) on page 87 for information about registering LSF services.

Host not used by LSF

If you configure a list of server hosts in the `Host` section of the `lsb.hosts` file, `mbatchd` allows `sbatchd` to run only on the hosts listed. 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.
```

Both of these errors are most often caused by not running the following commands, in order, after adding a host to the configuration.

```
lsadmin reconfig
```

```
badmin reconfig
```

You must run both of these before starting the daemons on the new host.

UNKNOWN host type or model

Viewing 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    ( )
```

Fixing UNKNOWN matched host type or matched model

- 1 Start the host.
- 2 Run `lsadmin limstartup` to start LIM on the host.

For example:

```
lsadmin limstartup hostA
Starting up LIM on <hostA> .... done
```

or, if EGO is enabled in the LSF cluster, you can also run:

```
egosh ego start lim hostA
Starting 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.

DEFAULT host type or model

Viewing 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.

Fixing 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 HostType 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 `HostModel` 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 `lsadmin reconfig` 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 `setuid 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. See [Registering Service Ports](#) on page 87 for information about configuring LSF services.

```
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 `lsadmin` or `badmin` 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.

Error Messages

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 `FLEXlm` 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.

See [Host Naming](#) on page 89 for information about working with multi-addressed hosts.

```
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.

```
lsadmin 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.

```
lsadmin 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 <cname>@<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
getElogLock: Can't open existing lock file <logdir/file>: error
getElogLock: Error in opening lock file <logdir/file>: error
releaseElogLock: unlink(<logdir/lockfile>) failed: error
touchElogLock: Failed to open lock file <logdir/file>: error
touchElogLock: 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*.

```
badmin reconfig
```

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>bhosts</code> displays "LSF is down. Please wait..." <code>bjobs</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...
Send/receive handshake message to/from <code>mbatchd</code>	Internal library error	Cannot connect to LSF. Please wait...
	<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).

Setting 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 `lsadmin` and `badmin` 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 `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.

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 `lsadmin` and `badmin`, 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*.

```
badmin sbddebug -o
```

Turn off temporary debug settings for `sbatchd` 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*.

```
badmin mbddebug -l 1
```

Log messages for `mbatchd` running on the local host and set the log message level to `LOG_DEBUG1`. This command must be submitted from the host on which `mbatchd` is running because *host_name* cannot be specified with `mbddebug`.

```
badmin sbddebug -f hostB/myfolder/myfile hostA
```

Log messages for `sbatchd` running on *hostA*, to the directory `myfile` on the server *hostB*, with the file name `myfile.sbatchd.log.hostA`. The debug level is the default value, `LOG_DEBUG` level in parameter `LSF_LOG_MASK`.

```
badmin schddebug -l 2
```

Log messages for `mbatchd` running on the local host and set the log message level to `LOG_DEBUG2`. This command must be submitted from the host on which `mbatchd` is running because *host_name* cannot be specified with `schddebug`.

```
badmin schddebug -l 1 -c "LC_PERFM"
```

```
badmin schdtime -l 2
```

Activate the LSF scheduling debug feature.

Log performance messages for `mbatchd` 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*.

For timing level examples, see [Setting Daemon Timing Levels](#) on page 788.

Setting 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 `badmin` 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 debug level examples, see [Setting Daemon Message Log to Debug Level](#) on page 785.

For a detailed description of `lsadmin` and `badmin`, see the *Platform LSF Command Reference*.

Setting Daemon Timing Levels

VIII

LSF Utilities

[Using lstcsh on page 793](#)

Using lstcsh

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.

Contents

- ◆ [About lstcsh](#) on page 793
- ◆ [Starting lstcsh](#) on page 796
- ◆ [Using lstcsh as Your Login Shell](#) on page 796
- ◆ [Host Redirection](#) on page 797
- ◆ [Task Control](#) on page 798
- ◆ [Built-in Commands](#) on page 798
- ◆ [Writing Shell Scripts in lstcsh](#) on page 800

About lstcsh

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.

`lstcsh` 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.

In this section

- ◆ [Task Lists](#) on page 794
- ◆ [Local and Remote Modes](#) on page 794
- ◆ [Automatic Remote Execution](#) on page 795

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 *Platform LSF Configuration Reference* for information about the `.lsftask` and `lsf.task` files.

Changing task list membership

You can use the LSF commands `lsltasks` and `lsrtasks` to inspect and change the memberships of the local and remote task lists.

Task lists and resource requirements

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.

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.

See [@ character](#) on page 795 for more details.

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 *Platform 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.

Starting lstcsh

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 `@`, as shown by examples in [Task Control](#) on page 798.

tcsh version

l`stcsh` is based on `tcsh 6.03` (7 bit mode). It does not support the new features of the latest `tcsh`.

Starting lstcsh

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`.
-

Using 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.

Set your login shell

Using csh

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 /usr/share/lsf/bin/lstcsh
```

The next time `user3` logs in, the login shell will be `lstcsh`.

Using 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 *Platform 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 `lsrtasks`.

Task Control

Task control in `lstcsh` is the same as in `tcsh` except for remote background tasks. `lstcsh` 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] + Running          sleep 30
[2]   Running          sleep 40
<hostB>
[1] + Running          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

`lstcsh` supports two built-in commands to control load sharing, `lsmode` and `connect`.

In this section

- ◆ [lsmode](#) on page 799
- ◆ [connect](#) on page 800

lsmode

Syntax

```
lsmode [on|off] [local|remote] [e|-e] [v|-v] [t|-t]
```

Description

The `lsmode` command reports that LSF is enabled if `lstcsh` was able to contact LIM when it started up. If LSF is disabled, no load-sharing features are available.

The `lsmode` command takes a number of arguments that control how `lstcsh` behaves.

With no arguments, `lsmode` 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. See [Local and Remote Modes](#) on page 794 for a description of local and remote modes.

[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.

`lstcsh` 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.

Writing Shell Scripts in lstcsh

You should write shell scripts in `/bin/sh` and use the `lstools` commands for load sharing. However, `lstcsh` can be used to write load-sharing shell scripts.

By default, an `lstcsh` script is executed as a normal `tosh` script with load-sharing disabled.

Run a script with load sharing enabled

The `lstcsh -L` option tells `lstcsh` 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 `lstcsh` script with load sharing enabled:

- ◆ Run `lstcsh -L script_name`, or
- ◆ Make the script executable and put the following as the first line of the script. By default, `lstcsh` is installed in `LSF_BINDIR`.

The following assumes you installed `lstcsh` in the `/usr/share/lsf/bin` directory):

```
#!/usr/share/lsf/bin/lstcsh -L
```

- 1 Start an interactive `lstcsh`.
 - 2 Enable load sharing, and set to remote mode:
`lsmode on remote`
 - 3 Use the `source` command to read the script in.
-

Index

Symbols

- !(NOT) operator
 - job dependencies 482
- %I substitution string in job arrays 528
- %J substitution string in job arrays 528
- %USRCMD string in job starters 628
- && (AND) operator
 - job dependencies 482
- .cshrc file and lscsh 796
- .lsbatch directory 37
- .rhosts file
 - disadvantages 739
 - file transfer with lsrcp 755
 - host authentication 739
 - troubleshooting 772
- /etc/hosts file
 - example host entries 92
 - host naming 89
 - name lookup 90
 - troubleshooting 772
- /etc/hosts.equiv file
 - host authentication 738
 - troubleshooting 772
 - using rcp 755
- /etc/services file
 - adding LSF entries to 87
- /etc/shells file, and lscsh 796
- /etc/syslog.conf file 206, 760
- /usr/include/sys/syslog.h file 759
- @ (at sign) in lscsh 797
- || (OR) operator
 - job dependencies 482
- ~ (tilde)
 - not operator
 - host partition fairshare 342
 - host-based resources 274

A

- abnormal job termination 122
- ABS_RUNLIMIT parameter in lsb.params 605
- absolute job priority scheduling
 - admin value 496
 - description 493
 - LSF feature interactions 500

- modifying calculated APS value 496
- priority factors 493
- resizable jobs 502

- absolute run time limit 605
- access permissions for interactive tasks 667
- accounting information for advance reservations 471
- adaptive dispatch. *See* chunk jobs
- admin APS value 496
- administrator comments
 - logging in lsb.events
 - for host open and close 71
 - for mbatchd restart 54
 - for queue events 111

ADMINISTRATORS

- lsb.queues file 496
- administrators
 - cluster administrator description 51
 - primary LSF administrator 51

ADMINISTRATORS parameter in

- lsf.cluster.cluster_name 51

advance reservation

- accounting information 471
- adding and removing 458
- commands 458
- configuring user policies 456
- description 454, 456
- preemption 474
- reservation ID 471
- schmod_advrsv plugin 456
- submitting jobs 473
- user policies 456
- viewing 467
- viewing accounting information 471
- weekly planner (brsvs -p) 468

advance reservations

- compute units 476
- resizable jobs 475

advanced dependency conditions 485

advanced reservation

- compute units 103

AFS (Andrew File System)

- overview 751
- tokens for esub and eexec 632

Index

- aliases
 - for resource names 310
 - host name ranges 89
 - using as host names 89
- allocation limits. *See* resource allocation limits
- AND operator (&&)
 - job dependencies 482
- Andrew File System. *See* AFS
- application profiles
 - adding and removing 402
 - checkpoint directory 514
 - configuring
 - for chunk jobs 520
 - controlling jobs 405
 - default application profile 403
 - description 402
 - job success exit values 403
 - modifying jobs (bmod -app) 405
 - pre- and post-execution commands
 - configuring 620
 - submitting jobs (bsub -app) 405
 - viewing
 - detailed information (bapp -l) 407
 - jobs (bjobs -app) 408
 - summary information (bacct -app) 408
 - summary information (bapp) 407
- APS. *See* absolute job priority scheduling
- APS_PRIORITY parameter in lsb.queues 495
- architecture
 - EGO 193
- architecture, viewing for hosts 66
- arguments
 - passed to the LSF event program 720
 - passing to job arrays 528
- arrays
 - chunking 532
- at sign (@) in lscsh 797
- augmentstarter job starter 629
- authentication
 - security 738
- automatic
 - duplicate event logging 762
 - event log archiving 762
 - job requeue 505
 - job rerun
 - description 511
 - resizable jobs 511
 - priority escalation 492
 - queue selection 33
 - remote execution in lscsh 795
 - time-based configuration 287
- automatic job priority escalation
 - resizable jobs 492
- automount command
 - NFS (Network File System) 751, 770
- automount option
 - /net 752
- autoresizable jobs
 - checkpoint and restart 515
- available
 - meaning 254
- B**
- bacct -app 408
- bacct command
 - CPU time display 608
 - SLA scheduling 390
- bacct -U
 - advance reservations 471
- backfill scheduling
 - default run limit 600
 - description 565
 - interruptible backfill 569
 - resizable jobs 566
 - resource allocation limits 422
- background jobs, bringing to foreground 798
- badadmin command
 - hopen 70
 - hrestart 52
 - hshutdown 52
 - hstartup 52
 - logging administrator comments
 - for host open and close 71
 - for mbatchd restart 54
 - for queue events 111
 - LSF event logs 760
 - mbdrestart 52, 58
 - qact 111
 - qclose 111
 - qinact 111
 - qopen 111
- balance keyword
 - cu string 548
- banded licensing 157
- bapp 407
- batch jobs
 - accessing files 751, 752
 - allocating processors 540
 - email about jobs
 - disabling 746
 - options 745
 - file access 631, 752
 - input and output 745
 - killing 131
 - pre- and post-execution commands 616
 - requeue 504
 - rerunning and restarting 511
 - signalling 131

- batch log files. *See* log files
- batch queues. *See* queues
- bbot command 127
 - user-assigned job priority 490
- B-Class LSF license type 157
- benchmarks for setting CPU factors 104
- Berkeley Internet Name Domain (BIND)
 - host naming 89
- bgadd command
 - job group limits 138
- bgdel command 143
- bgmod command
 - job group limits 143
- bhist command
 - job exit codes 766
 - LSF event logs 760
- bhist -l 142
 - administrator changes to absolute job priority scheduling 497
- bhosts command
 - checking time-based configuration 289
- bhosts -l 60
- bhosts -x
 - viewing host exception status 69
- BIND (Berkeley Internet Name Domain)
 - host naming 89
- binding processors
 - resizable jobs 576
- bjgroup command 139
 - SLA scheduling 396
- bjobs -app 408
- bjobs -aps
 - changes to absolute job priority scheduling 497
 - order of absolute job priority scheduling 499
- bjobs command
 - reservation ID for advance reservation 471
 - SLA scheduling 390
- bjobs -g 140
- bjobs -l
 - modified absolute job priority scheduling values 497
- bjobs -x
 - viewing job exception status 124
- bkill -app 405
- bkill -g 142
- black hole hosts 105, 145
- bladmin chkconfig command
 - checking time-based configuration 290
- blimits -c command
 - checking time-based configuration 290
- blimits command 434
- blinfo command
 - checking time-based configuration 290
- blstat command
 - checking time-based configuration 290
- bmod
 - absolute job priority scheduling 496
- bmod -app 405
- bmod -g 141
- bmod -is 750
- bmod -Zs 750
- Boolean resources 248
- bparams command
 - checking time-based configuration 290
 - viewing configuration parameters 44
- bqueues command
 - checking time-based configuration 290
 - cross-queue fairshare information 344
- bqueues -l
 - absolute job priority scheduling 499
- bresize cancel command 579
- bresize release command 579
- bresources command
 - checking time-based configuration 290
- brstart command
 - resizable jobs 515
- brresume -app 405
- brresume -g 141
- brsvadd -b
 - specifying begin times 459
- brsvadd command 458
- brsvadd -e
 - specifying end times 459
- brsvadd -m
 - specifying a host list 458
- brsvadd -R
 - specifying a resource requirement string 458
- brsvadd -t
 - specifying recurring reservations 460
- brsvdel command 467
- brsvmod command 462
- brsvs command 467
- brun command
 - advance reservation 475
 - forcing a job to run 129
 - job preemption 295
- bsla command 388
- bstop -app 405
- bstop -g 141
- bsub -app 405
- bsub command
 - email job notification 745
 - input and output 745
 - remote file access 753
 - submitting a job
 - associated to a job group 139
 - associated to a service class 385
- bsub -f 753

Index

- bsub -is 748
- bsub -sla 385
- bsub -Zs 749
- bswitch command
 - resizable jobs 579
- btop command
 - user-assigned job priority 490
- built-in load indices
 - overriding 276
- built-in resources 248
- busers command
 - checking time-based configuration 290
- busy host status
 - lsload command 61
 - status load index 254
- busy thresholds, tuning 724
- C**
- calculating license key check sums (lmcksum) 174
- calculation of required licenses 161
- candidate master hosts, specifying 727
- ceiling resource usage limit 599
- ceiling run limit 599
- chargeback fairshare 365
- check_license script, for counted software licenses 278
- checking
 - license server status (lmstat) 172
 - LSF floating client 185
- checkout of licenses 161
- checkpoint and restart
 - executables 516
 - resizable jobs 515
- checkpointable jobs
 - chunk jobs 523
- checksum
 - calculating for license key (lmcksum) 174
- chsh and lscsh 796
- chunk jobs
 - absolute job priority scheduling 501
 - checkpointing 523
 - CHUNK_JOB_DURATION parameter in lsb.params 520
 - configuring application profile for 520
 - configuring queue for 520
 - description 519
 - fairshare scheduling 523
 - job controls 522
 - killing 522
 - limitations on queues 521
 - migrating 523
 - modifying 523
 - rerunnable 523
 - resizable jobs 579
 - resource usage limits 597
 - resuming 523
 - submitting and controlling 522
 - WAIT status and pending reason 522
- CHUNK_JOB_DURATION
 - parameter in lsb.params 520
- chunking
 - job array 532
- CLEAN_PERIOD parameter in lsb.params 131
- client host
 - floating licenses 181
- closed host status 60
 - bhosts command 60
 - bhosts -l 60, 65
- closed_Adm condition, output of bhosts -l 60
- closed_Busy condition, output of bhosts -l 60
- closed_cu_Excl condition, output of bhosts -l 60
- closed_EGO condition, output of bhosts -l 60
- closed_Excl condition, output of bhosts -l 60
- closed_Full condition, output of bhosts -l 60
- closed_LIM condition, output of bhosts -l 60
- closed_Lock condition, output of bhosts -l 60
- closed_Wind condition, output of bhosts -l 60
- cluster administrators
 - description 51
 - viewing 44
- cluster names
 - viewing 44
- clusters
 - configuration file quick reference 57
 - protecting with strict checking 81
 - reconfiguring
 - commands 57
 - how reconfiguration affects licenses 58
 - viewing
 - information about 44
 - viewing information 44
- command file spooling
 - See also job file spooling
 - default directory 749
 - description 748
 - JOB_SPOOL_DIR parameter in lsb.params 748
- commands
 - built-in 799
 - checking time-based configuration 289
 - FLEXlm utilities 172
 - job starters 625
 - lmcksum (FLEXlm) 174
 - lmdown (FLEXlm) 174
 - lmhostid (FLEXlm) 174
 - lmremove (FLEXlm) 174
 - lmreread (FLEXlm) 174
 - lmstat (FLEXlm)
 - description 174

- displaying license server status 172
 - example 173
 - using 172
- lsmver (FLEXlm) 174
- lshosts -l 177
- post-execution. *See* post-execution commands
- pre-execution. *See* pre-execution commands
- running under user ID 621
- using in job control actions 646
- components
 - EGO 193
- compound resource requirements
 - multi-level 307
 - overview 298
 - syntax 305
- Comprehensive System Accounting (IRIX CSA)
 - configuring 758
- compute unit resource allocation 304
- compute units
 - advanced reservation 103
 - configuring external compute units 103
 - cu string 331
 - syntax 548
 - exclusive 551
 - external 103
 - host level job allocation 552
 - reservation 552
 - resource requirements 331
- concurrent threads 605
- CONDENSE keyword in lsb.hosts 98, 102
- CONDENSE_PENDING_REASONS parameter in lsb.params 678
- condensed host groups
 - defining 98, 102
 - viewing 64
- condensed notation
 - host names 95
- configuration
 - adding and removing
 - application profiles 402
 - queues 114
 - application profiles
 - job success exit values 403
 - commands for checking 289
 - default application profile 403
 - preselecting master hosts 726
 - removing
 - hosts 75
 - tuning
 - busy thresholds 724
 - LIM policies 724
 - load indices 725
 - load thresholds 725
 - mbatchd on UNIX 731
 - run windows 724
 - viewing
 - errors 58
- configuration files
 - location 151
 - non-shared file systems 752
 - reconfiguration quick reference 57
- configuration parameters. *See* individual parameter names
- CONSUMABLE
 - lsf.shared file 270
- consumers
 - about 194
- CONTROL_ACTION parameter in lsb.serviceclasses 395
- core file size limit 601
- CORELIMIT parameter in lsb.queues 601
- counted software licenses
 - configuring 278
 - description 277
- CPU
 - factors
 - static resource 258
 - time normalization 607
 - tuning in lsf.shared 103
 - limits
 - per job 602
 - per process 602
 - normalization 607
 - run queue length, description 656
 - time
 - cumulative and decayed 339
 - in dynamic user priority calculation 339
 - time limit
 - job-level resource limit 601
 - tuning CPU factors in lsf.shared 103
 - utilization, ut load index 254
 - viewing run queue length 104
- CPU factor
 - non-normalized run time limit 605
- CPU factor (cpuf) static resource 257
- CPU time
 - idle job exceptions 116, 145
- CPU time normalization 602
- CPU_TIME_FACTOR parameter in lsb.params
 - fairshare dynamic user priority 339
- cpuf static resource 258
- CPULIMIT parameter in lsb.queues 601
- Cray
 - UNICOS accounting log files 758
- cross-queue fairshare 343
- CSA (IRIX Comprehensive System Accounting)
 - configuring 758
- .cshrc file and lscsh 796

Index

- cu resource requirement string
 - resizable jobs 332
- cu string 331
 - keyword balance 548
 - keyword excl 548
 - keyword maxcus 548
 - keyword pref 548
 - keyword type 548
 - syntax 548
- cumulative CPU time 339
- custom event handlers 720
- custom file transfer
 - configuring 755
- custom resources
 - adding 270
 - configuring 270
 - description 268
 - resource types 248
- D**
- DAEMON line
 - license.dat file 159
- DAEMON line, editing 164
- daemons
 - commands 52
 - debug commands 785
 - error logs 206, 759
 - restarting
 - mbatchd 54
 - sbatchd 53
 - shutting down
 - mbatchd 55
 - sbatchd 53
 - TCP service ports 87
 - ypbind 90
- data loader plug-ins 701
 - LSF 698
- data loaders
 - EGO 699
- data purger 700, 702
 - record expiry time 710
 - schedule 702, 708
- data segment size limit 602
- DATALIMIT parameter in lsb.queues 602
- date of expiry (demo) 156
- DCE/DFS (Distributed File System)
 - credentials, esub and eexec 632
 - overview 751
- deadline constraint scheduling
 - description 291
 - parallel jobs 575
 - resizable jobs 292
- deadlock, avoiding signal and job action 647
- debug information
 - logging classes 208
 - logging levels 208
- debug level
 - commands for daemons 785
 - setting temporarily 785
- debug log classes
 - description 208
- debug log levels
 - description 208
- decayed
 - CPU time 339
 - run time 361
- dedicated resources. *See* exclusive resources
- DEFAULT
 - model or type with lshosts command 776
- default
 - input file spooling 749
 - job control actions 643
 - JOB_SPOOL_DIR 749
 - LSF log file location 206, 757
 - LSF_LOGDIR 206, 760
 - output file spooling 749
 - queue
 - viewing 33
 - resource usage limits 599
 - run limit
 - backfill scheduling 600
 - UNIX directory structure 49
 - Windows directory structure 50
- default normalization host 602
- DEFAULT_APPLICATION parameter in lsb.params 403
- DEFAULT_HOST_SPEC parameter
 - in lsb.params 602
 - in lsb.queues 607
- DEFAULT_JOBGROUP parameter in lsb.params 135
- defined keyword 312
- definitions
 - EGO 192
- delayed SLA scheduling goals
 - control action 395
- deletion
 - automatic job group cleanup 144
- demand
 - defining in SDK 193
- demo license 156
- dependency conditions
 - job arrays
 - operators 529
 - relational operators 484
- dependency conditions. *See* job dependency conditions
- dependency expressions
 - multiple conditions 482

- DFS (Distributed File System). *See* DCE/DFS
- directories
 - default UNIX directory structure 49
 - default Windows directory structure 50
 - log
 - permissions and ownership 207, 758
 - .lsbatch 37
 - LSF_SERVERDIR
 - esub and eexec 632
 - remote access 631, 752
 - shared 37
 - user accounts 37
- directory for license (demo) 163
- directory for license (permanent) 166
- disks
 - I/O rate 255
- dispatch order, fairshare 340
- dispatch turn
 - description 34
- dispatch windows
 - description 478
 - hosts 70
 - queues 112
 - tuning for LIM 724
- dispatch, adaptive. *See* chunk jobs
- DISPATCH_WINDOW
 - queues 113
- displaying
 - FLEXlm version information (lmver) 174
 - hardware host ID (lmhostid) 174
 - license server status (lmstat) 172, 174
 - licensed products 177
- distributed license server hosts 175
- distribution (partial licensing) 177
- distribution of licenses 161
- Domain Name Service (DNS)
 - host naming 89
- done job dependency condition 484
- DONE job state
 - description 120
 - post-execution commands 617
- done jobs, viewing 120
- dual-stack hosts
 - defining official host name 92
 - dns configuration 93
- duplicate event logging 762
 - after network partitioning 762
 - automatic 762
 - description 761
 - mbatchd restart with MAX_INFO_DIRS 680
- dynamic
 - hosts, protecting with strict checking 81
 - master host 38
 - resources 248
 - user priority
 - description 338
 - formula 339
- dynamic priority
 - fairshare adjustment 340
 - memory based 340
- E**
- eadmin script
 - default exception actions 146
 - host and job exception handling 145
- EADMIN_TRIGGER_DURATION parameter in
 - lsb.params 117
- E-Class LSF license type 157
- eexec (external executable) script
 - description 631
 - passing data to execution environments 641
 - running as non-root 641
- effective run queue length
 - built-in resources 254
 - description 656
 - tuning LIM 726
- EGO
 - components 193
 - grace period
 - resources 213
 - how it works 193
 - what it is 192
- ego.conf file
 - EGO_LOG_MASK parameter 759
 - managing error logs 759
- EGO_LOG_MASK parameter in ego.conf 759
- electronic mail. *See* email
- eligible hosts
 - viewing 36
- eligible hosts, viewing 36
- ELIM (external LIM)
 - counted software licenses 278
- email
 - disabling batch job notification 746
 - job options 745
 - limiting the size of job email 746
- embedded submission options for interactive
 - jobs 660
- ENABLE_EXIT_RATE_PER_SLOT parameter in
 - lsb.params 148
- ENABLE_ONE_UG_LIMITS
 - limits and user groups 424
- encryption
 - esub 635
 - X-Window 635
- ended job dependency condition 484
- environment
 - setting 205, 216

Index

- environment of a job 37
- environment variables. *See* individual environment variable names
- equal share fairshare 365
- error logs
 - EGO_LOG_MASK parameter 759
 - log directory
 - LSF_LOGDIR 206, 760
 - log files 206, 759
 - LSF daemons 206, 759
 - LSF_LOG_MASK parameter 759
 - managing log files 758
 - on UNIX and Windows 206, 760
- error messages
 - service 217
- errors
 - viewing in reconfiguration 58
- esub (external submission) executable
 - description 632
 - environment variables 632
 - job submission parameters 637
 - mandatory method (LSB_ESUB_METHOD) 639
 - pass data to execution environments 641
- esub method (LSB_ESUB_METHOD) 639
- /etc/hosts file
 - example host entries 92
 - host naming 89
 - name lookup 90
 - troubleshooting 772
- /etc/hosts.equiv file
 - host authentication 738
 - troubleshooting 772
 - using rcp 755
- /etc/services file
 - adding LSF entries to 87
- /etc/syslog.conf file 206, 760
- evaluation license 156
- event generation 719
- event log archiving
 - automatic 762
- event log replication. *See* duplicate event logging
- event logging
 - mbatchd restart with MAX_INFO_DIRS 680
- event logs 703
 - automatic archiving 703, 762
 - configuring duplicate logging 762
 - duplicate logging 762
 - ego.stream file 703
 - logging administrator comments
 - for host open and close 71
 - for mbatchd restart 54
 - for queue events 111
 - lsb.events file 38
 - lsb.stream file 703
 - LSF Batch log file in lsb.events file 760
 - update interval 762
- Event Viewer, Windows 719
- EVENT_UPDATE_INTERVAL in lsb.params 762
- events
 - custom programs to handle 720
 - generated by LSF 720
 - logging 38
- example
 - demo license 159
 - partial licensing 178
 - permanent license 160
- example.services file 87
- examples
 - /etc/hosts file entries 92
 - lstat command 173
- exception handling
 - configuring host exceptions 105
 - configuring in queues 116
 - description 39
- exception status
 - for hosts
 - viewing with bhosts 68
 - for jobs
 - viewing with bjobs 124
 - viewing with bqueues 110
- excl keyword
 - cu string 548
- exclusive jobs
 - requeue 509
- EXCLUSIVE parameter
 - in lsb.queues file 100
- exclusive resources host-based resources
 - exclusive resources 313
- exclusive scheduling
 - resizable jobs 292
- execution
 - environment 37
 - forcing for jobs 129
 - priority 257
- execution host
 - mandatory for parallel jobs 545
- exit codes
 - job success exit values 403
 - returned by jobs 766
- exit dependency condition
 - relational operators 484
- exit job dependency condition 484
- EXIT job state
 - abnormal job termination 122
 - pre- and post-execution commands 617
- exit rate for jobs 105, 145
- EXIT_RATE
 - bhosts -l 68

- EXIT_RATE parameter in lsb.hosts 105
- expiry date (demo) 156
- expiry time for mbatchd 732
- external
 - job dependency condition 485
 - job submission executable (esub) 632
 - LIM. *See* ELIM (external LIM)
- external resources 248
- F**
- factor grace period
 - absolute job priority scheduling 494
- factor limit
 - absolute job priority scheduling 493
- factor weight
 - absolute job priority scheduling 493
- fairshare adjustment plugin 340
- FAIRSHARE parameter in lsb.queues 345
- fairshare scheduling
 - absolute job priority scheduling 500
 - across queues 343
 - chargeback 365
 - chunk jobs 523
 - defining policies that apply to several queues 343
 - description 336
 - dynamic user priority
 - description 338
 - formula 339
 - equal share 365
 - global 364
 - hierarchical share tree 349
 - overview 335
 - parallel jobs 574
 - policies 336
 - priority user 366
 - resizable jobs 367
 - resource usage measurement 338
 - static priority 367
 - user share assignment 337
 - viewing cross-queue fairshare information 344
- FAIRSHARE_QUEUES parameter
 - in bqueues 344
 - in lsb.queues 345
 - OBSOLETE 497
- fast job dispatching 675
- fault tolerance
 - description 38
 - non-shared file systems 752
- FCFS (first-come, first-served) scheduling 34
- FEATURE line
 - license.dat file (demo) 158
 - license.dat file (permanent) 159
- features (LSF) 168
- file (for demo license) 158
- file (for permanent license) 159
- file access, interactive tasks 667
- file preparation, job arrays 527
- file sharing 37
- file size usage limit 603
- file spooling. *See* command file spooling, job file spooling
- file systems
 - AFS (Andrew File System) 751
 - DCE/DFS (Distributed File System) 751
 - NFS (Network File System) 751
 - supported by LSF 751
- file transfer
 - lsrcp command 754
- file, updating 170
- FILELIMIT parameter in lsb.queues 603
- files
 - /etc/hosts
 - example host entries 92
 - host naming 89
 - name lookup 90
 - /etc/services
 - adding LSF entries to 87
 - automatic time-based configuration 287
 - copying across hosts 667
 - enabling utmp registration 661
 - hosts
 - configuring 91
 - if-else constructs 287
 - license.dat
 - location 167
 - lsb.params
 - CHUNK_JOB_DURATION parameter 520
 - JOB_ACCEPT_INTERVAL parameter 34
 - lsf.cluster.cluster_name
 - FLOAT_CLIENTS_ADDR_RANGE parameter 184
 - lsf.conf
 - configuring TCP service ports 87
 - daemon service ports 87
 - lsf.shared
 - adding a custom host types and models 86
 - redirecting 653
 - redirecting stdout and stderr 667
 - resolv.conf 90
 - spooling command and job files 659
- finger command in lscsh 795
- first execution host
 - parallel jobs 545
 - resizable jobs 545
- first-come, first-served (FCFS) scheduling 34
- FLEXlm
 - calculating license key check sums

Index

- (lmcksum) 174
 - displaying
 - hardware host ID (lmhostid) 174
 - license server status (lmstat) 174
 - version information (lmver) 174
 - removing a licensed feature (lmremove) 174
 - rereading license file (lmreread) 174
 - shutting down license server (lmdown) 174
 - utility commands 172
 - FLEXlm log file 173
 - FLEXlm-based license 156
 - floating client
 - description 181
 - enabling 182
 - resetting 181
 - specifying host or model type 182
 - verifying 185
 - floating client licenses
 - FLOAT_CLIENTS parameter in
 - lsf.cluster.cluster_name 183
 - FLOAT_CLIENTS_ADDR_RANGE parameter in
 - lsf.cluster.cluster_name 184
 - floating LSF client
 - description 181
 - enabling 182
 - resetting 181
 - specifying host or model type 182
 - verifying 185
 - floating LSF client licenses
 - FLOAT_CLIENTS parameter in
 - lsf.cluster.cluster_name 183
 - FLOAT_CLIENTS_ADDR_RANGE parameter in
 - lsf.cluster.cluster_name 184
 - floating software licenses
 - configuring dedicated queue for 280
 - managing with LSF 278
 - forcing job execution 129
 - formula
 - fairshare dynamic user priority calculation 339
 - free memory 255
 - FS absolute job priority scheduling factor 495
 - G**
 - gethostbyname function (host naming) 90
 - global fairshare 364
 - GLOBAL_EXIT_RATE parameter in lsb.params 148
 - GLOBEtrrotter Software 160
 - goal-oriented scheduling. *See* SLA scheduling
 - goals
 - SLA scheduling 384
 - grace period
 - absolute job priority scheduling factor 494
 - EGO resources 213
 - licenses 161
 - groups
 - external host 99, 103
 - external user 153
 - hosts 96
 - users 151
 - groups, specifying 364
 - H**
 - hard resource limits
 - description 595
 - hard resource usage limits
 - example 599
 - hardware host ID
 - displaying (lmhostid) 174
 - hardware host name 165
 - hierarchical fairshare 347
 - hierarchical share tree 349
 - HIST_HOURS parameter in lsb.params
 - fairshare dynamic user priority 340
 - historical run time 361
 - history
 - job arrays 530, 534
 - HJOB_LIMIT parameter in lsb.queues 427
 - hname static resource 257
 - home directories
 - remote file access 753
 - shared 37
 - \$HOME/.lsbatch directory 37
 - \$HOME/.rhosts file
 - disadvantages 739
 - file transfer with lsrcp command 755
 - host authentication 739
 - hopen badadmin command 70
 - host affinity
 - same string 329
 - host dispatch windows 478
 - host entries
 - examples 92
 - host exception handling
 - configuring 105
 - example 106
 - job exit rate exception 105, 145
 - host failure 39
 - host groups 64
 - CONDENSE keyword 98, 102
 - condensed
 - viewing 64
 - configuring external host groups 99
 - defining 151
 - defining condensed 98, 102
 - external 99
 - overview 151
- host ID 165
 - displaying (lmhostid) 174

- host identifier 165
- host load levels 36
- host model static resource 257
- host models
 - adding custom names in `lsf.shared` 86
 - DEFAULT 776
 - select string 310
 - tuning CPU factors 104
- host name static resource 257
- host names
 - `/etc/hosts` file 89
 - aliases 89
 - matching with Internet addresses 89
 - ranges 95
 - ranges as aliases 89
 - `resolv.conf` file 90
 - resolver function 90
 - using DNS 90
 - wildcards and special characters 97, 101
- host partition fairshare 341
- host redirection 797
- host reservation. *See* advance reservation
- host selection 304
- host state. *See* host status
- host status
 - busy
 - load index 254
 - `lsload` 61
 - closed 65
 - bhosts 60
 - description 60
 - index 253
 - lockU and lockW 61, 254
 - ok 61, 254
 - ok
 - bhosts 60
 - load index 254
 - `lsload` 61
 - unavail 60
 - load index 254
 - `lsload` 61
 - unlicensed 60, 61, 254
 - unreach 60
- host thresholds 36
- host type static resource 257
- host types
 - adding custom names in `lsf.shared` 86
 - DEFAULT 776
 - resource requirements 298
 - select string 310
- host-based resources
 - description 248
- hostcache
 - modifying 84
- host-level
 - fairshare scheduling 341
 - resource information 450
- host-locked software licenses 277
- HOSTRESORDER environment variable 773
- hosts
 - adding with `lsinstall` 73
 - associating resources with 273
 - available 254
 - closing 70
 - connecting to remote 800
 - controlling 70
 - copying files across 667
 - dispatch windows 70
 - displaying 64
 - file 90
 - finding resource 668
 - for advance reservations 458
 - logging on the least loaded 668
 - master candidates 727
 - multiple network interfaces 91
 - official name 89
 - opening 70
 - preselecting masters for LIM 726
 - redirecting 797
 - removing 75
 - resource groups overview 194
 - restricting use by queues 115
 - selecting for task 664
 - setting up 74, 75
 - spanning with parallel jobs 554
 - specifying master candidates 727
 - specifying master host 762
 - viewing
 - architecture information 66
 - detailed information 65
 - eligible 36
 - exceptions 68
 - history 67
 - job exit rate and load 68
 - load by host 66, 251
 - load by resource 247
 - model and type information 67
 - resource allocation limits (blimits) 434
 - shared resources 250
 - status of closed servers 65
 - suspending conditions 613
- hosts file (`/etc/hosts`)
 - example host entries 92
 - host naming 89
 - name lookup 90
 - troubleshooting 772
- hosts file (LSF)
 - configuring 91

Index

- HOSTS parameter
 - in `lsb.hosts` 96
 - in `lsb.queues` file 96, 100
- `hosts.equiv` file
 - host authentication 738
 - using `rcp` 755
- `hostsetup` script 74, 75
- `hrestart` `badadmin` command 52
- `hshutdown` `badadmin` command 52
- `hstartup` `badadmin` command 52
- I**
- `%l` substitution string in job arrays 528
- idle job exceptions
 - configuring 116
 - description 116, 145
 - viewing with `bjobs` 124
 - viewing with `bqueues` 110
- idle resources
 - releasing for resizable jobs 578
- idle time
 - built-in load index 255
 - description 655
 - suspending conditions 612
- if-else constructs
 - creating 289
 - files 287
- INCREMENT lines 170
- index list for job arrays 526
- input and output files
 - and interactive jobs 653
 - job arrays 527
 - splitting `stdout` and `stderr` 654
 - spooling directory 749
- installation
 - on non-shared file systems 752
- installation (demo) 162
- installation (permanent) 164
- installation directories
 - default UNIX structure 49
 - Windows default structure 50
- interactive jobs
 - competing for software licenses 281
 - configuring queues to accept 652
 - redirecting scripts to standard input 660
 - resource reservation 324
 - running X applications 658
 - scheduling policies 651
 - specifying embedded submission options 660
 - specifying job options in a file 659
 - specifying shell 660
 - splitting `stdout` and `stderr` 654
 - spooling job command files 659
 - submitting 652
 - submitting and redirecting streams to files 653
 - submitting with pseudo-terminals 653
 - viewing queues for 652
 - writing job file one line at a time 659
 - writing job scripts 659
- interactive sessions
 - starting 668
- interactive tasks
 - file access 667
- interfaces, network 91
- Internet addresses
 - matching with host names 89
- Internet Domain Name Service (DNS)
 - host naming 89
- inter-queue priority 610
- interruptible backfill 569
 - resizable jobs 570
- io load index 255
- IPv6
 - configure hosts 94
 - supported platforms 93
 - using IPv6 addresses 93
- IRIX
 - Comprehensive System Accounting (CSA)
 - configuring 758
 - `utmp` file registration 661
- it load index
 - automatic job suspension 611
 - description 255, 655
 - suspending conditions 612
- J**
- `%J` substitution string in job arrays 528
- JL/P parameter in `lsb.users` 427
- JL/U parameter in `lsb.hosts` 427
- job array dependency conditions
 - operators 529
- job arrays
 - `%l` substitution string 528
 - `%J` substitution string 528
 - argument passing 528
 - controlling 532
 - creating 525
 - dependency condition operators 529
 - dependency conditions 529
 - file preparation 527
 - format 526
 - history 530, 534
 - index list 526
 - input and output files 527
 - maximum size 527
 - monitoring 530, 534
 - overview 525
 - passing arguments 528

- redirection of input and output 527
- requeueing 534
- specifying job slot limit 535
- standard input and output 528
- status 530, 534
- submitting 525
- syntax 526
- job checkpoint and restart
 - application profiles 514
 - checkpoint directory 514
 - chkpnt 516
 - restart 516
 - job restart description 516
 - queue level 514
 - resizable jobs 515
- job chunking. *See* chunk jobs
- job control actions
 - CHKPNT 646
 - configuring 645
 - default actions 643
 - LS_EXEC_T 643
 - on Windows 645
 - overriding terminate interval 644
 - RESUME 644
 - SUSPEND 643
 - TERMINATE 644
 - terminating 647
 - using commands in 646
 - with lscsh 798
- job data transformer 700, 701
 - schedule 709
- job dependencies
 - logical operators 482
- job dependency conditions
 - advanced 485
 - description 484
 - done 484
 - ended 484
 - examples 486
 - exit 484
 - external 485
 - job arrays 529
 - job name 485
 - post_done 485, 617
 - post_err 485, 617
 - post-processing 617
 - scheduling 482
 - specifying 482
 - specifying job ID 485
 - started 485
- job dispatch
 - fast 675
 - maximum per session 675
- job dispatch order, fairshare 340
- job email
 - bsub options 745
 - disabling batch job notification 746
 - limiting size with LSB_MAILSIZE_LIMIT 746
- job exception handling
 - configuring 116
 - default eadmin action 146
 - exception types 116, 145
 - viewing exception status with bjobs 124
 - viewing exceptions with bqueues 110
- job execution environment 37
- job exit rate exceptions
 - configuring 105
 - description 105, 145
 - viewing with bhosts 68
- job file spooling
 - See also* command file spooling
 - default directory 749
 - description 748
 - JOB_SPOOL_DIR parameter in lsb.params 748
- job files 33
- job groups
 - add limits 138
 - automatic deletion 144
 - controlling jobs 141
 - default job group 135
 - description 134
 - displaying SLA service classes 396
 - example hierarchy 137
 - job limits 136
 - modify limits 143
 - viewing 139
- job idle factor
 - viewing with bjobs 124
- job ladders. *See* batch jobs, pre-execution commands
- job limit 605
- job limits 422
 - job groups 136
- job migration
 - absolute job priority scheduling 501
 - configuration
 - application profile level 518
 - host level 518
 - queue level 517
 - description 517
 - manual 517
- job overrun exceptions
 - configuring 116
 - description 116, 145
 - viewing with bjobs 124
 - viewing with bqueues 110
- job preemption
 - absolute job priority scheduling 501
 - description 295

Index

- SLA scheduling 395
- job priority
 - automatic escalation 492
 - user assigned 490
- job requeue
 - absolute job priority scheduling 501
 - automatic 505
 - exclusive 509
 - resizable jobs 579
 - reverse requeue 508
 - user-specified 510
- job rerun
 - absolute job priority scheduling 501
 - disabling post-execution commands 512, 619
 - resizable jobs 511
- job restart
 - resizable jobs 515
- job scripts
 - writing for interactive jobs 659
- job slot limits 421
 - for job arrays 535
 - for parallel jobs 543
 - viewing resource allocation limits (blimits) 434
- job spanning 304, 327
- job starters
 - augmentstarter 629
 - command-level 625
 - lic_starter script to manage software licenses 281
 - preservestarter 629
 - queue-level
 - configuring 628
 - description 626
 - specifying command or script 627, 628
 - user commands 628
- job states
 - description 120
 - DONE
 - description 120
 - post-execution commands 617
 - EXIT
 - abnormal job termination 122
 - pre- and post-execution commands 617
 - PEND 120
 - POST_DONE 122, 617
 - POST_ERR 122, 617
 - post-execution 617
 - PSUSP 120
 - RUN 120
 - SSUSP 120
 - USUSP 120
 - WAIT for chunk jobs 522
- job submission 32
- job success exit values
 - application profile configuration 403
 - JOB_ACCEPT_INTERVAL parameter in Isb.params 34
 - JOB_CONTROLS parameter in Isb.queues 645
 - JOB_EXIT_RATE_DURATION parameter in Isb.params 105
 - JOB_GROUP_CLEAN parameter in Isb.params 144
 - JOB_IDLE parameter in Isb.queues 116
 - JOB_INCLUDE_POSTPROC parameter in Isb.applications 622
 - JOB_OVERRUN parameter in Isb.queues 116
 - JOB_POSITION_CONTROL_BY_ADMIN parameter in Isb.params 678
 - JOB_POSTPROC_TIMEOUT parameter in Isb.applications 623
 - JOB_PRIORITY_OVER_TIME parameter in Isb.params
 - automatic job priority escalation 492
 - JOB_SCHEDULING_INTERVAL parameter in Isb.params 34, 677
 - JOB_SPOOL_DIR parameter in Isb.params 748
 - JOB_STARTER
 - Isb.queues file 628
 - JOB_STARTER parameter in Isb.queues 628
 - JOB_TERMINATE_INTERVAL parameter in Isb.params 603, 644
 - JOB_UNDERRUN parameter in Isb.queues 116
- job-level
 - post-execution commands
 - description 617
 - pre-execution commands
 - configuring 618
 - description 617
 - resource requirements 302
 - resource reservation 438
 - run limits 604
- job-level suspending conditions
 - viewing 613
- jobs
 - changing execution order 127
 - checkpointing
 - chunk jobs 523
 - CHKPNT 646
 - controlling
 - in an application profile 405
 - dispatch order 35
 - email notification
 - disabling 746
 - options 745
 - enabling rerun 511
 - enforcing memory usage limits 603
 - exit codes
 - description 766
 - job success exit values 403
 - forcing execution 129
 - interactive. *See* interactive jobs

- killing 131
 - in an application profile 405
- limiting processors for parallel 559
- modifying
 - in an application profile 405
- optimum number running in SLA 384
- pending 120
- preemption 610
- preemptive and preemptable 296
- requeueing 534
- requeuing
 - description 510
- rerunning 511
- rerunning automatically 511
- restarting
 - automatically 511
- resuming 130, 614
 - in an application profile 405
- sending specific signals to 133
- short running 519
- specifying options for interactive 659
- specifying shell for interactive 660
- spooling command and job files 659
- spooling input, output, and command files 748
- stopping
 - in an application profile 405
- submitting
 - to a job group 139
 - to a service class 385
 - to an application profile 405
- suspended 614
- suspending 130, 610
- suspending at queue level 613
- switching queues 128
- viewing
 - by user 123
 - configuration parameters in lsb.params 45
 - order in queue 35
 - viewing resource allocation limits (blimits) 434
- jobs command in lscsh 798
- jobs requeue, description 504
- JRIORITY absolute job priority scheduling factor 495
- JSDL
 - configuration 273
 - required resources 271
- JSDL (Job Submission Description Language)
 - benefits 581
 - elim.jsdl 590
 - how to submit a job 590
 - LSF extension elements 587
 - schema files 581
 - supported elements 581
 - unsupported elements 589
 - using with LSF 581
- L**
 - libfairshareadjust 340
 - lic_starter script, to manage software licenses 281
 - license file (demo) 158
 - license file (permanent) 159
 - license host 160
 - license key
 - calculating checksum (lmcksum) 174
 - license management commands 172
 - license server
 - calculating license key check sums (lmcksum) 174
 - checking status (lmstat) 172
 - displaying
 - hardware host ID (lmhostid) 174
 - status (lmstat) 172
 - version information (lmver) 174
 - specifying TCP port 189
 - utility commands 172
 - license server daemon (lmgrd) 160
 - license server host 168
 - description 160
 - license.dat (demo) 158
 - license.dat (permanent) 159
 - license.dat file
 - location 167
 - license.dat, updating 170
 - licenses
 - cluster reconfiguration 58
 - displaying
 - FLEXlm version information (lmver) 174
 - hardware host ID (lmhostid) 174
 - LSF products 177
 - server status (lmstat) 174
 - floating LSF client
 - description 181
 - enabling 182
 - specifying host or model type 182
 - removing feature (lmremove) 174
 - rereading license file (lmreread) 174
 - shutting down FLEXlm server (lmdown) 174
 - software
 - counted 277
 - dedicated queue for 280
 - floating 278
 - host locked 277
 - interactive jobs and 281
- LIM (Load Information Manager)
 - preselecting master hosts 726
 - tuning 726
 - load indices 725
 - load thresholds 725
 - policies 724
 - run windows 724

Index

- LIM, master 160
- limdebug command 785
- limitations
 - lsrnp command 754
 - on chunk job queues 521
- limits
 - job 422
 - job group 136
 - job slot 421
 - See resource allocation limits or resource usage limits
- limtime command 788
- Linux and Windows hosts
 - multicore processor support 156
- lmcksum command 174
- lmdown command 174
- lmgrd daemon 160
- lmhostid 165
- lmhostid command 174
- lmremove command 174
- lmreread command 174
- lmstat command
 - description 174
 - example 173
 - using 172
- lmver command 174
- load average 254
- load indices
 - See also resources
 - built-in
 - overriding 276
 - summary 253
 - io 255
 - it 255
 - ls 255
 - mem 255
 - pg 255
 - r15m 254
 - r15s 254
 - r1m 254
 - swp 255
 - tmp 255
 - tuning for LIM 725
 - types 655
 - ut 254
 - ut load index
 - select resource requirement string 310
 - viewing 44, 256
- load levels
 - viewing by resource 247
 - viewing for cluster 44
 - viewing for hosts 66
- load sharing
 - displaying current setting 799
 - with lsrcsh 800
- load thresholds
 - configuring 612
 - description 300
 - paging rate, tuning 726
 - queue level 612
 - resizable jobs 612
 - tuning 725
 - tuning for LIM 724, 725
- loader controller 698, 700, 701
 - data collection
 - disabling 711
 - frequency 702, 711
- local event logging
 - mbatchd restart with MAX_INFO_DIRS 680
- local mode in lsrcsh 794
- LOCAL_MAX_PREEXEC_RETRY parameter
 - in lsb.applications, lsb.params, lsb.queues 623
- locality
 - parallel jobs 327, 547, 554
- location of license (demo) 163
- location of license (permanent) 166
- lockU and lockW host status
 - lslod command 61
 - status load index 254
- log file (FLEXlm) 173
- log files
 - about 207
 - change ownership 759
 - default location 206, 757
 - directory permissions and ownership 207, 758
 - ESC 207
 - logging events on Cray UNICOS 758
 - maintaining 206, 759
 - managing 206, 759
 - mbatchd.log.host_name 206, 759
 - mbschd.log.host_name 206, 759
 - named 207
 - PEM 207
 - res.log.host_name 206, 759
 - sbatchd.log.host_name 206, 759
 - troubleshooting 210
 - VEMKD 207
 - WSG 207
 - WSM 207
- LOG_DAEMON facility, LSF error logging 206, 760
- logging classes
 - description 208
- logging levels 758
 - description 208
- logical operators
 - in time expressions 286
 - job dependencies 482
- login sessions 255

- login shell, using lscsh as 796
- logs
 - classes 208
 - entry formats 207
 - levels 208
- lost_and_found queue 114
- ls load index 255
- ls_connect API call 632
- LS_EXEC_T environment variable 643
- LS_JOBPID environment variable 641
- ls_postevent() arguments 720
- lsadmin command
 - limlock 60
 - limunlock 60
- lsb.acct file
 - job exit information 763
 - job termination reason logging 762
 - killing jobs in a batch 131
- lsb.applications file
 - adding an application profile 402
 - JOB_INCLUDE_POSTPROC parameter 622
 - JOB_POSTPROC_TIMEOUT parameter 623
 - LOCAL_MAX_PREEEXEC_RETRY parameter 623
 - MAX_PREEEXEC_RETRY parameter 623
 - NAME parameter 402
 - POST_EXEC parameter 620
 - PRE_EXEC parameter 620
 - REMOTE_MAX_PREEEXEC_RETRY parameter 623
 - REQUEUE_EXIT_VALUES parameter 505
 - SUCCESS_EXIT_VALUES parameter 403
- lsb.events file
 - event logging 38
 - logging administrator comments
 - for host open and close 71
 - for mbatchd restart 54
 - for queue events 111
 - managing event log 760
- lsb.hosts file
 - CONDENSE keyword 98, 102
 - host exception handling 105
 - if-else constructs 287
 - time-based configuration 287
 - user groups 152
 - USER_SHARES parameter 152
 - using host groups 96
 - using user groups 152
- lsb.modules file
 - advance reservation 456
 - schmod_adrvsv plugin 456
- lsb.params file
 - absolute run time limit 605
 - CHUNK_JOB_DURATION parameter 520
 - CLEAN_PERIOD parameter 131
 - CONDENSE_PENDING_REASONS parameter 678
 - controlling lsb.events file rewrites 760
 - CPU time normalization 602
 - default application profile 403
 - default normalization host 602
 - DEFAULT_JOBGROUP parameter 135
 - EADMIN_TRIGGER_DURATION threshold for exception handling 117
 - ENABLE_EXIT_RATE_PER_SLOT parameter 148
 - GLOBAL_EXIT_RATE parameter 148
 - if-else constructs 287
 - job termination signal interval 603
 - JOB_ACCEPT_INTERVAL parameter 34
 - JOB_EXIT_RATE_DURATION for exception handling 105
 - JOB_GROUP_CLEAN parameter 144
 - JOB_POSITION_CONTROL_BY_ADMIN parameter 678
 - JOB_PRIORITY_OVER_TIME parameter 492
 - JOB_SCHEDULING_INTERVAL parameter 34, 677
 - JOB_SPOOL_DIR parameter 748
 - LOCAL_MAX_PREEEXEC_RETRY parameter 623
 - MAX_CONCURRENT_JOB_QUERY parameter 677
 - MAX_INFO_DIRS parameter 679
 - MAX_PEND_JOBS parameter 121
 - MAX_PREEEXEC_RETRY parameter 623
 - MAX_SBD_CONNS parameter 676
 - MAX_USER_PRIORITY parameter 490
 - MBD_QUERY_CPUS parameter 734
 - MBD_REFRESH_TIME parameter 731
 - MIN_REFRESH_TIME parameter 732
 - MIN_SWITCH_PERIOD parameter 678
 - NEWJOB_REFRESH parameter 734
 - non-normalized run time limit 605
 - PARALLEL_SCHED_BY_SLOT parameter 544
 - REMOTE_MAX_PREEEXEC_RETRY parameter 623
 - specifying job input files 748
 - SUB_TRY_INTERVAL parameter 121
 - time-based configuration 287
- lsb.queues file
 - adding a queue 114
 - EXCLUSIVE parameter 100
 - HOSTS parameter 96, 100
 - if-else constructs 287
 - job exception handling 116
 - JOB_IDLE parameter 116
 - JOB_OVERRUN parameter 116
 - JOB_UNDERRUN parameter 116
 - LOCAL_MAX_PREEEXEC_RETRY parameter 623
 - MAX_PREEEXEC_RETRY parameter 623
 - normalization host 607
 - POST_EXEC parameter 619
 - PRE_EXEC parameter 619

Index

- QUEUE_NAME parameter 114
- REMOTE_MAX_PREEXEC_RETRY parameter 623
- REQUEUE_EXIT_VALUES parameter 505
- resource usage limits 599
- restricting host use by queues 115
- time-based configuration 287
- user groups 152
- USERS parameter 152
- using compute units 100
- using host groups 96
- using user groups 152
- lsb.queues files
 - DEFAULT_HOST_SPEC parameter 607
- lsb.resources file
 - advance reservation policies 456
 - if-else constructs 287
 - parameters 425
 - ReservationUsage section 238
 - time-based configuration 287
 - viewing limit configuration (blimits) 434
- lsb.serviceclasses file
 - configuring SLA scheduling 386
 - CONTROL_ACTION 395
- lsb.users file
 - if-else constructs 287
 - MAX_PEND_JOBS parameter 121
 - time-based configuration 287
 - user groups 152
 - USER_NAME parameter 152
 - using user groups 152
- LSB_CHUNK_RUSAGE parameter in lsf.conf 597
- LSB_CONFDIR parameter in lsf.conf
 - default UNIX directory 49
- LSB_DEFAULT_JOBGROUP environment variable 135
- LSB_DEFAULTQUEUE environment variable 33
- LSB_DISABLE_RERUN_POST_EXEC parameter in lsf.conf 512, 619
- LSB_ESUB_METHOD in lsf.conf 639
- LSB_HOSTS environment variable 538
- LSB_JOB_CPULIMIT parameter in lsf.conf 602
- LSB_JOBEXIT_STAT environment variable 619
- LSB_JOBINDEX environment variable 528
- LSB_JOBPEND environment variable 619
- LSB_JOBPGIDS environment variable 646
- LSB_JOBPIIDS environment variable 646
- LSB_LOCALDIR parameter in lsf.conf file 762
- LSB_MAILSIZE environment variable 746
- LSB_MAILSIZE_LIMIT parameter in lsf.conf 746
- LSB_MAILTO parameter in lsf.conf 745
- LSB_MAX_JOB_DISPATCH_PER_SESSION parameter in lsf.conf 675
- LSB_MBD_PORT parameter in lsf.conf 87
- LSB_NTRIES environment variable 121
- LSB_PRE_POST_EXEC_USER parameter
 - in lsf.sudoers 621
- LSB_QUERY_PORT parameter in lsf.conf 677, 733
- LSB_REQUEUE_TO_BOTTOM parameter in lsf.conf 505, 508
- LSB_SBD_PORT parameter in lsf.conf 87
- LSB_SHAREDIR parameter in lsf.conf
 - default UNIX directory 49
 - duplicate event logging 761
- LSB_SHAREDIR/cluster_name/logdir
 - LSF log files 206, 757
- LSB_SIGSTOP parameter in lsf.conf 130, 647
- LSB_SUB_ABORT_VALUE environment variable 636
- LSB_SUB_COMMANDNAME
 - lsf.conf file 241
- LSB_SUB_COMMANDNAME parameter in lsf.conf 633
- LSB_SUB_PARM_FILE environment variable 632
- LSB_SUSP_REASON environment variable 646
- LSB_SUSP_SUBREASONS environment variable 646
- LSB_UTMP parameter in lsf.conf 661
- lsbapplications file
 - using compute units 100
- .lsbatch directory 37
- LSF Daemon Error Log 206, 759
- LSF events
 - generated by LSF 720
 - generation of 719
 - program arguments 720
- LSF features 168
- LSF license grace period 161
- LSF licenses
 - license file location 167
- LSF licensing
 - banded license types 157
 - lsf_mv_grid_filter feature 157
 - multicore processor Linux and Windows hosts 156
- LSF master LIM 160
- LSF parameters. *See* individual parameter names
- LSF products 168
 - displaying enabled license 177
- LSF vendor license daemon (lsf_ld) 160
- LSF version
 - viewing 44
- lsf.cluster.cluster_name file
 - ADMINISTRATORS parameter 51
 - configuring cluster administrators 51
 - exclusive resources 313
 - floating LSF client licenses 182
 - license checkout 161
- lsf.conf file
 - comprehensive system account 758
 - configuring duplicate logging 762

- configuring TCP service ports 87
- custom file transfer 755
- daemon service ports 87
- default UNIX directory 49
- duplicate event logging 761
- dynamic host startup time 77
- limiting the size of job email 746
- LSB_CHUNK_RUSAGE parameter 597
- LSB_DISABLE_RERUN_POST_EXEC parameter 512, 619
- LSB_JOB_CPULIMIT parameter 602
- LSB_JOB_MEMLIMIT 603
- LSB_MAILSIZE_LIMIT parameter 746
- LSB_MAILTO parameter 745
- LSB_MAX_JOB_DISPATCH_PER_SESSION parameter 675
- LSB_MEMLIMIT_ENFORCE 603
- LSB_QUERY_PORT parameter 677, 733
- LSB_SIGSTOP parameter 130
- LSB_SUB_COMMANDNAME parameter 241, 633
- LSF_BINDIR parameter 49, 755
- LSF_DYNAMIC_HOST_WAIT_TIME parameter 77
- LSF_ENABLE_CSA parameter 758
- LSF_LOG_MASK parameter 759
- LSF_LOGDIR parameter 206, 760
- LSF_MANDIR parameter 49
- LSF_MASTER_LIST parameter 77, 727
- LSF_MISC parameter 49
- lsf_mv_grid_filter license feature 157
- LSF_NT2UNIX_CLTRB environment variable 648
- LSF_NT2UNIX_CLTRC environment variable 648
- LSF_RES_PORT parameter in lsf.conf 87
- LSF_RSH parameter in lsf.conf
 - controlling daemons 52
- LSF_SERVERDIR 169
- LSF_SERVERDIR parameter in lsf.conf 49
- LSF_STRICT_CHECKING parameter in lsf.conf 81
- LSF_STRICT_RESREQ parameter in lsf.conf 313
- LSF_SUB_COMMANDLINE environment variable 241
- LSF_TOP directory
 - default UNIX directory structure 49
- lsfinstall
 - adding a host 73
- lsfinstall command
 - using 237
- lsfinstall program 162
- lsfshutdown command
 - shutting down daemons on all hosts 52
- lsfstartup command
 - starting daemons on all hosts 52
- lshosts
 - viewing dynamic host information 69
- lshosts command
 - DEFAULT host model or type 776
- lshosts -l
 - viewing licensed LSF products 177
- lsrcp command
 - description 753
 - executable file location 755
 - file transfer 754
 - restrictions 754
- lstcsh
 - about 793
 - difference from other shells 795
 - exiting 796
 - limitations 795
 - local mode 794
 - remote mode 794
 - resource requirements 794
 - starting 796
 - task lists 794
 - using as login shell 796
- lsf.conf parameter LSF_LICENSE_FILE 166
- lsf.licensescheduler file
 - if-else constructs 287
 - time-based configuration 287
- lsf.shared file
 - adding a custom host type and model 86
 - tuning CPU factors 103
- lsf.sudoers file
 - LSB_PRE_POST_EXEC_USER parameter 621
- LSF_BINDIR parameter in lsf.conf 49, 755
- LSF_CONFDIR parameter in lsf.conf 49
- LSF_DYNAMIC_HOST_WAIT_TIME parameter in lsf.conf 77
- LSF_ENABLE_CSA parameter in lsf.conf 758
- LSF_INCLUDEDIR parameter in lsf.conf

Index

writing shell scripts in 800

M

mail

disabling batch job notification 746

job options 745

limiting the size of job email 746

mandatory esub method (LSB_ESUB_METHOD) 639

mandatory first execution host

parallel jobs 545

resizable jobs 545

master esub (mesub)

configuring 640

description 638

master host candidates 218

with LSF_MASTER_LIST 727

master host failover

about 218

master hosts 218

in non-shared file systems 752

preselecting 726

specifying 762

viewing current 44

MAX_CONCURRENT_JOB_QUERY parameter in
lsb.params 677

MAX_INFO_DIRS parameter in lsb.params 679

MAX_JOB_NUM parameter in lsb.params 760

MAX_JOBS parameter in lsb.users 427

MAX_PEND_JOBS parameter in lsb.params or
lsb.users 121

MAX_PREEEXEC_RETRY parameter

in lsb.applications, lsb.params, lsb.queues 623

MAX_SBD_CONNS parameter in lsb.params 676

MAX_USER_PRIORITY parameter in lsb.params

automatic job priority escalation 492

user-assigned job priority 490

maxcus keyword

cu string 548

maximum

number of processors for parallel jobs 544

resource usage limit 599

run limit 599

maxmem static resource 257

maxswp static resource 257

maxtmp static resource 257

mbatchd (master batch daemon)

expiry time 732

push new job information to a child

mbatchd 732, 734

refresh time 732

restarting 54

shutting down 55

specifying query-dedicated port 732

specifying time interval for forking child 732

tuning on UNIX 731

mbatchd.log.host_name file 206, 759

MBD. See mbatchd

MBD_QUERY_CPUS parameter in lsb.params 734

MBD_REFRESH_TIME parameter in lsb.params 731

mbddebug command 785

mbdrestart badmin command 52

mbdtime command 788

mbschd.log.host_name file 206, 759

MEM absolute job priority scheduling factor 495

mem load index

description 255

MEMLIMIT parameter in lsb.queues 603

memory

available 255

usage limit 603

viewing resource allocation limits (blimits) 434

mesub (master esub)

configuring 640

description 638

migrated jobs

absolute job priority scheduling 501

MIN_REFRESH_TIME parameter in lsb.params 732

MIN_SWITCH_PERIOD parameter in lsb.params 678

minimum processors for parallel jobs 544

missed SLA scheduling goals

control action 395

model static resource 257

modify

LSF_MASTER_LIST 76

multicore processor support 156

multi-homed hosts 91

multiple condensed host groups 98

multiple conditions

dependency expressions 482

multiple esub 638

multiple license server hosts 175

multiple queues

absolute job priority scheduling 497

multi-processor hosts 161

multiprocessor hosts

configuring queue-level load thresholds 613

tuning LIM 726

multithreading, configuring mbatchd for 731

MXJ parameter in lsb.hosts 427

N

name lookup using /etc/hosts file 90

NAME parameter in lsb.applications 402

native language system, and lscsh 795

ncores static resource 257

ncpus static resource

dynamically changing processors 264

reported by LIM 257

- ndisks static resource 257
- network
 - failure 38
 - interfaces 91
 - partitioning
 - and duplicate event logging 762
 - port numbers
 - configuring for NIS or NIS+ databases 88
- Network File System. *See* NFS
- Network Information Service. *See* NIS
- NEWJOB_REFRESH parameter in lsb.params 734
- NFS (Network File System)
 - automount command 751, 770
 - nosuid option 739
 - overview 751
- NIS (Network Information Service)
 - configuring port numbers 88
 - host name lookup in LSF 89
 - yycat hosts.byname 90
- non-normalized run time limit 605
- non-shared file systems
 - installing LSF 752
- normalization
 - CPU time limit 607
 - host 607
 - run time limit 607
- normalization host 602
- normalized run queue length
 - description 254
 - tuning LIM 726
- nosuid option, NFS mounting 739
- NOT operator (!)
 - job dependencies 482
- not operator (~)
 - host partition fairshare 342
 - host-based resources 274
- nprocs static resource 257
- NQS (Network Queueing System)
 - logging events on Cray UNICOS 758
- nqsacct file 758
- nthreads static resource 257
- number of processors for parallel jobs 544
- numdone dependency condition 529
- numended dependency condition 529
- numerical resources 248
- numexit dependency condition 529
- numhold dependency condition 529
- numpend dependency condition 529
- numrun dependency condition 529
- numstart dependency condition 529
- O**
- obsolete parameters
 - FAIRSHARE_QUEUES 497
 - USER_ADVANCE_RESERVATION in lsb.params 457
- official host name 89
- ok host status
 - lsload command 61
 - status load index 254
- ok host status
 - bhosts command 60
 - lsload command 61
 - status load index 254
- one-time advance reservation 459
- operators
 - job array dependency conditions 529
 - logical in job dependencies 482
 - logical in time expressions 286
 - not (~)
 - host partition fairshare 342
 - host-based resources 274
 - relational
 - exit dependency condition 484
 - remote file access 753
 - resource requirements 311
 - selection strings 311
- OR operator (||)
 - job dependencies 482
- order of job execution, changing 127
- order resource requirement string
 - resizable jobs 319
- order string 318
- OS memory limit 604
- output and input files, for job arrays 528
- output file spooling
 - default directory 749
- overflow job exceptions
 - configuring 116
 - description 116, 145
 - viewing with bjobs 124
 - viewing with bqueues 110
- ownership of log directory 207, 758
- P**
- paging rate
 - automatic job suspension 611
 - checking 612
 - description 255, 655
 - load index 255
 - suspending conditions 612
- parallel fairshare 574
- parallel jobs
 - allocating processors 540
 - backfill scheduling 565
 - deadline constraint scheduling 575
 - fairshare 574
 - interruptible backfill scheduling 569

Index

- job slot limits 543
- limiting processors 559
- locality 327, 547, 554
- mandatory first execution host 545
- number of processors 544
- overview 537
- processor reservation 562
- PROCLIMIT
 - resizable jobs 560
- selecting hosts with same string 329
- spanning hosts 554
- submitting 540
- parallel programming
 - packages 539
- parallel tasks
 - running with lsgroup 665
 - starting 541
- PARALLEL_SCHED_BY_SLOT parameter in lsb.params 544
- partial licensing 177
- partitioned networks 38
- PATH environment variable
 - and lscsh 796
 - shared user directories 37
- paths
 - /etc/hosts file
 - example host entries 92
 - host naming 89
 - name lookup 90
 - /etc/hosts.equiv file 739
 - host authentication 738
 - using rcp 755
 - /etc/services file
 - adding LSF entries to 87
 - /net 752
 - /usr/bin/ 37
- PEND
 - job state 120
- pending jobs
 - absolute job priority scheduling 493
 - order of absolute job priority scheduling 499
- pending reasons
 - queue-level resource reservation 438
 - viewing 121
- per-CPU licensing 161
- PERF 698
- performance tuning
 - busy thresholds 724
 - LIM policies 724
 - load indices 725
 - load thresholds 725
 - mbatchd on UNIX 731
 - preselecting master hosts 726
 - run windows for LIM 724
- periodic tasks 206, 759
- per-job CPU limit 602
- permanent license 156
- permanent LSF license
 - displaying server status (lmstat) 172
- permissions
 - log directory 207, 758
- per-process limits
 - CPU limit 602
 - data segment size 602
 - file size 603
 - memory limit 603
 - stack segment size 606
- per-resource reservation
 - configuring 439
 - viewing with bresources 451
- pg load index
 - suspending conditions 612
- PIM (Process Information Manager)
 - resource use 251
- PJOB_LIMIT parameter in lsb.queues 427
- plclient 702, 705, 706
- PluginModule section in lsb.modules
 - advance reservation 456
- policies
 - fairshare 336
 - tuning for LIM 724
- port
 - notation
 - license daemon 189
- port (in LSF_LICENSE_FILE) 166
- port numbers
 - configuring for NIS or NIS+ databases 88
- ports
 - registering daemon services 87
 - specifying dedicated 732
- post_done job dependency condition 485, 617
- POST_DONE post-execution job state 122, 617
- post_err job dependency condition 485, 617
- POST_ERR post-execution job state 122, 617
- POST_EXEC parameter
 - in lsb.applications 620
 - in lsb.queues 619
- post-execution
 - job dependency conditions 617
 - job states 617
- post-execution commands
 - configuring 618
 - disabling for rerunnable jobs 512, 619
 - including in job processing 621
 - job-level 617
 - overview 616
 - queue-level 617
 - running under user ID 621

- setting a post-processing timeout [623](#)
- PRE_EXEC parameter
 - in lsb.applications [620](#)
 - in lsb.queues [619](#)
- PREEMPT_FOR parameter in lsb.params [576](#)
- preemptable
 - jobs [296](#)
 - queues [295](#)
- preemption
 - absolute job priority scheduling [501](#)
- preemption. *See* preemptive scheduling
- preemptive
 - jobs [296](#)
 - queues [295](#)
 - scheduling
 - description [295](#)
- preemptive scheduling
 - advance reservation [474](#)
 - SLA scheduling [395](#)
- pre-execution commands
 - configuring [618](#)
 - job-level [617](#)
 - overview [616](#)
 - queue-level [617](#)
 - running under user ID [621](#)
 - setting a retry limit [623](#)
- pref keyword
 - cu string [548](#)
- preservestarter job starter [629](#)
- priority
 - automatic escalation [492](#)
 - user assigned [490](#)
- PRIORITY parameter in lsb.queues [346](#), [352](#)
- priority user fairshare [366](#)
- priority. *See* dynamic user priority
- PROC absolute job priority scheduling factor [495](#)
- process allocation for parallel jobs [304](#), [329](#)
- PROCESLIMIT parameter in lsb.queues [604](#)
- processor binding
 - resizable jobs [576](#)
- processor reservation
 - configuring [562](#)
- processors
 - limiting for parallel jobs [559](#)
 - number for parallel jobs [544](#)
 - reservation [562](#)
- products [168](#)
- PRODUCTS line, editing [168](#)
- programs
 - handling LSF events [720](#)
- project names
 - viewing resource allocation limits (blimits) [434](#)
- pseudo-terminal
 - submitting interactive jobs with [653](#)
 - using to run a task [665](#)
- PSUSP job state
 - description [130](#)
 - overview [120](#)
- Q**
- qact badmin command [111](#)
- qclose badmin command [111](#)
- qinact badmin command [111](#)
- QJOB_LIMIT parameter in lsb.queues [427](#)
- qopen badmin command [111](#)
- QRIORITY absolute job priority scheduling factor [495](#)
- queue dispatch windows [478](#)
- queue groups
 - absolute job priority scheduling [497](#)
- queue priority [32](#)
- queue thresholds
 - viewing [36](#)
- QUEUE_GROUP parameter in lsb.queues [497](#)
- QUEUE_NAME parameter in lsb.queues [114](#)
- queue-based fairshare
 - resource usage measurement [339](#)
- queue-level
 - fairshare across queues [343](#)
 - fairshare scheduling [343](#)
 - job starter [628](#)
 - pre- and post-execution commands
 - configuring [619](#)
 - description [617](#)
 - resource limits [599](#)
 - resource requirements [300](#)
 - resource reservation [438](#)
 - run limits [600](#)
- queue-level resource information
 - viewing [450](#)
- queue-level resource limits, defaults [599](#)
- queues
 - adding and removing [114](#)
 - automatic selection [33](#)
 - backfill queue [566](#)
 - changing job order within [127](#)
 - chunk job limitations [521](#)
 - configuring
 - for chunk jobs [520](#)
 - job control actions [645](#)
 - suspending conditions [613](#)
 - default [33](#)
 - dispatch windows [112](#)
 - fairshare across queues [343](#)
 - interactive [652](#)
 - interruptible backfill [570](#)
 - lost_and_found [114](#)
 - overview [32](#)

Index

- preemptive and preemptable 295
 - QUEUE_EXIT_VALUES parameter 619
 - restricting host use 115
 - run windows 113
 - setting rerun level 511
 - specifying suspending conditions 613
 - user-assigned job priority 490
 - viewing
 - available 108
 - default 33
 - detailed queue information 108
 - for interactive jobs 652
 - history 109
 - job exception status 110
 - resource allocation limits (blimits) 434
 - status 108
 - viewing absolute job priority scheduling information 499
 - viewing administrator of 44
- R**
- R res_req command argument 304
 - r15m load index
 - built-in resources 254
 - description 656
 - suspending conditions 612
 - r15s load index
 - built-in resources 254
 - description 656
 - suspending conditions 612
 - r1m load index
 - built-in resources 254
 - description 656
 - suspending conditions 612
 - ranges
 - host name aliases 89
 - rcp command 753
 - recurring advance reservation 460
 - redundant license server hosts 176
 - relational operators
 - exit dependency condition 484
 - remote execution
 - with lscsh 795
 - remote file access
 - operators 753
 - remote jobs
 - bringing background jobs to foreground 798
 - execution priority 257
 - remote mode in lscsh 794
 - REMOTE_MAX_PREEEXEC_RETRY parameter
 - in lsb.applications, lsb.params, lsb.queues 623
 - remove
 - master host 76
 - removing
 - a licensed feature (lremove) 174
 - reports (reporting feature) 713
 - reports 691
 - architecture 698
 - creating 694, 695, 696
 - custom 692, 694
 - creating 694, 695, 696
 - deleting 695, 697
 - producing 697
 - data loader plug-ins 701
 - log files 702, 706
 - LSF 698
 - data purger 700, 702
 - record expiry time 710
 - schedule 702, 708
 - database 692, 701, 703
 - Derby 692, 703
 - moving 703, 714
 - MySQL 716
 - Oracle 234, 235, 715
 - schema 234, 715, 716
 - deleting 695, 697
 - disabling 713
 - event data files 703
 - EGO 708
 - LSF 707
 - exporting 693, 694, 695, 697
 - job data transformer 700, 701
 - schedule 709
 - loader controller 698, 700, 701
 - data collection 702, 711
 - log files 706
 - plclient 702, 705, 706
 - status 705
 - log files 701, 702, 706
 - data loader plug-ins 706
 - loader controller 706
 - PERF 698
 - producing
 - custom 697
 - standard 693
 - services 700
 - data purger 700, 702
 - database 701, 703
 - disabling automatic startup 705
 - job data transformer 700, 701
 - loader controller 700, 701
 - log files 701, 706
 - restarting 701, 704
 - standard 692
 - exporting 693, 695
 - producing 693
 - QUEUE_EXIT_VALUES parameter in lsb.applications 505

- QUEUE_EXIT_VALUES parameter in
 - lsb.queues 505, 508
- queued jobs
 - absolute job priority scheduling 501
 - automatic 505
 - description 504
 - exclusive 509
 - resizable jobs 579
 - reverse 508
 - user-specified 510
- rerun jobs
 - absolute job priority scheduling 501
- rerunnable jobs 511
 - chunk jobs 523
 - disabling post-execution 512, 619
- RERUNNABLE parameter in lsb.queues 511
- res.log.host_name file 206, 759
- RES_REQ parameter
 - in lsb.applications 100
 - in lsb.queues 100
- resdebug command 785
- reservation
 - advance 454, 456
- reservation ID
 - advance reservation 471
- ReservationUsage section
 - lsb.resources file 238
- reserved memory
 - for pending jobs 451
- resetting LSF floating client 181
- resizable jobs
 - absolute job priority scheduling 502
 - advance reservations 475
 - automatic job priority escalation 492
 - backfill scheduling 566
 - bresize cancel command 579
 - bresize release command 579
 - checkpoint and restart 515
 - chunk jobs 579
 - compute units 476
 - cu resource requirement string 332
 - deadline constraint scheduling 292
 - description 578
 - exclusive scheduling 292
 - fairshare scheduling 367
 - first execution host 545
 - idle resources 578
 - interruptible backfill 570
 - job rerun 511
 - JOB_ACCEPT_INTERVAL parameter 578
 - limiting processors for parallel jobs 560
 - load thresholds 612
 - minimum and maximum processors for parallel jobs 544
 - order resource requirement string 319
 - processor binding 576
 - queued jobs 579
 - resource allocation limits 424
 - resource requirements 299
 - usage resource requirement string 324
 - same resource requirement string 330
 - select resource requirement string 317
 - SLA scheduling 396
 - slot reservation 437
 - span resource requirement string 328
 - switched jobs 579
 - time-based slot reservation 447
- resolv.conf file 90
- resolver function 90
- resource allocation limits
 - configuring 425
 - description 420
 - enforcement 421
 - job limits 422
 - job slot limits 421
 - resolving conflicts 428
 - resource requirements 420
 - resource reservation and backfill 422
 - switched jobs 422
 - viewing (blimits) 434
- resource configurations
 - viewing with blimits 434
- resource consumers 420
- resource distribution tree
 - about 194
- resource granularity 439
- resource groups
 - about 194
- resource names
 - aliases 310
 - description 270
- resource plan
 - overview 194
- resource reclaim
 - grace period 213
- resource requirement string
 - cu section
 - syntax 548
- resource requirements
 - and task lists in lscsh 794
 - compound
 - multi-level 307
 - syntax 305
 - compute units 304
 - description 298
 - exclusive resources 313
 - for advance reservations 458
 - host type 298

Index

- operators 311
- ordering hosts 304, 318
- parallel job locality 304, 327
- parallel job processes 304, 329
- parallel jobs 331
 - selecting hosts 329
- resizable jobs 299
- resource reservation 320
- resource usage 304, 320
- select string 310
- selecting hosts 304, 310, 329
- simple
 - multi-level 306
 - syntax 304
- topology 331
- resource reservation
 - absolute job priority scheduling 501
 - description 437
 - resizable jobs 437
 - resource allocation limits 422
 - static shared resources 275
- resource types
 - external resources 248
- resource usage
 - fairshare scheduling 338
 - resource requirements 304, 320
 - viewing 251
- resource usage limits
 - ceiling 599
 - chunk job enforcement 597
 - configuring 599
 - conflicting 596
 - default 599
 - for deadline constraints 291
 - hard 599
 - maximum 599
 - priority 596
 - soft 599
 - specifying 599
- ResourceMap section in `lsf.cluster.cluster_name` 273
- ResourceReservation section in `lsb.resources` 456
- resources
 - See also* load indices
 - adding custom 270
 - advance reservations 454
 - associating with hosts 273
 - Boolean 248
 - built-in 253
 - configuring custom 270
 - configuring limits 425
 - custom 268
 - host-level 450
 - per-resource configuration 451
 - queue-level 450
 - releasing for resizable jobs 578
 - shared 249, 250
 - types 248
 - viewing
 - available 44, 246
 - host load 247
 - shared 44
- restime command 788
- restrictions
 - chunk job queues 521
 - lsrsh command 754
 - lstcsh 795
- RESUME job control action 644
- resume thresholds
 - viewing 614
- RESUME_COND parameter in `lsb.queues` 644
- reverse requeue 508
- rxpri static resource 257
- .rhosts file
 - troubleshooting 772
 - disadvantages 739
 - file transfer with lsrsh 755
 - host authentication 739
- rlogin command
 - interactive terminals 655
- rsh command
 - lsfstart 52
- RUN job state
 - overview 120
- run limits
 - ceiling 599
 - configuring 596, 604
 - default 600
 - maximum 599
 - specifying 607
- run queue
 - effective 254
 - normalized 254
 - suspending conditions 612
- run time
 - decayed 361
 - historical 361
 - normalization 607
- run time limit
 - non-normalized (absolute) 605
- run windows
 - description 477
 - queues 113
 - tuning for LIM 724
- RUN_JOB_FACTOR parameter in `lsb.params`
 - fairshare dynamic user priority 340
- RUN_TIME_FACTOR parameter in `lsb.params`
 - fairshare dynamic user priority 340
- RUN_WINDOW

- queues 113
- RUNLIMIT parameter in lsb.queues 604
- running jobs
 - viewing 120
- rusage
 - resource requirements section 304
 - resource reservation 438
 - usage string syntax 320
- rusage resource requirement string
 - resizable jobs 324
- S**
- same resource requirement string
 - resizable jobs 330
- same string 329
- sample /etc/hosts file entries 92
- sbatchd (slave batch daemon)
 - remote file access 752
 - restarting 53
 - shutting down 53
- sbatchd.log.host_name file 206, 759
- sbddebug command 785
- sbdtime command 788
- schddebug command 785
- schddtime command 788
- scheduling
 - exclusive 292
 - fairshare 336
 - hierarchical fairshare 347
 - preemptive
 - description 295
 - service level agreement (SLA) 34, 383
 - threshold
 - host selection 36
 - queue-level resource requirements 300
- scheduling policies
 - absolute job priority scheduling 493
 - automatic job priority escalation 492
 - user-assigned job priority 490
- scheduling priority factors
 - absolute job priority scheduling 493
- schmod_advrsv plugin for advance reservation 456
- S-Class LSF license type 157
- scripts
 - check_license for counted software licenses 278
 - lic_starter to manage software licenses 281
 - redirecting to standard input for interactive jobs 660
 - writing for interactive jobs 659
 - writing in lscsh 800
- SDK
 - defining demand 193
- security
 - LSF authentication 738
- select resource requirement string
 - resizable jobs 317
 - ut load index 310
- selection strings
 - defined keyword 312
 - description 310
 - operators 311
- server hosts, viewing detailed information 65
- SERVER line
 - license.dat file 159
- server static resource 257, 258
- server status closed 65
- servers
 - displaying license status (lmstat) 174
- service class
 - configuring 386
 - examples 387
 - goal-oriented scheduling 384
- service classes
 - bacct command 388, 390
 - bjgroup command 396
 - bjobs command 390
 - description 384
 - submitting jobs 385
- service database examples 87
- service error messages 217
- service level agreement. *See* SLA scheduling
- service level goals
 - job preemption 395
 - optimum number of running jobs 384
 - service classes 384
- service ports (TCP and UDP)
 - registering 87
- services
 - about 194
 - cluster
 - service director 194
 - web service gateway 194
 - WEBGUI 194
- setuid permissions 772
- setup (demo) 162
- setup (permanent) 164
- SGI IRIX. *See* IRIX
- share assignments 337
- share tree 349
- shared file systems
 - using LSF without 752
- shared files 770
- shared resources
 - defined keyword 312
 - description 249
 - exclusive resourceselection strings
 - exclusive resources 313
 - static

Index

- reserving 275
- viewing 250
- shared user directories 37
- shares
 - fairshare assignment 337
 - viewing user share information 150
- shell mode, enabling 667
- shell scripts. *See* scripts
- shell variables and lscsh 795
- shells
 - default shell for interactive jobs 661
 - lscsh 795
 - specifying for interactive jobs 660
- short-running jobs, as chunk jobs 519
- shutting down
 - FLEXlm server (lmdown) 174
- SIGCONT signal
 - default RESUME action 644
 - job control actions 133
- SIGINT signal
 - conversion to Windows 648
 - default TERMINATE action 644
 - job control actions 133
- SIGKILL signal
 - default TERMINATE action 644
 - job control actions 133
 - sending a signal to a job 133
- signals
 - avoiding job action deadlock 647
 - configuring SIGSTOP 130, 644, 647
 - converting 648
 - customizing conversion 648
 - job exit codes 766
 - sending to a job 133
 - SIGINT 133
 - SIGTERM 133
- SIGQUIT signal
 - conversion to Windows 648
- SIGSTOP signal
 - bstop 130
 - configuring 130, 644, 647
 - default SUSPEND action 644
 - job control actions 133
- SIGTERM signal
 - default TERMINATE action 644
 - job control actions 133
- SIGTSTP signal
 - bstop 130
 - default SUSPEND action 644
- simple resource requirements
 - multi-level 306
 - syntax 304
- sitched jobs
 - resource allocation limits 422
- site-defined resources
 - resource types 248
- SLA scheduling
 - bacct command 390
 - bjgroup command 396
 - bjobs command 390
 - bsla command 388
 - configuring 386
 - deadline goals 384
 - delayed goals 395
 - description 383
 - job preemption 395
 - missed goals 395
 - optimum number of running jobs 384
 - resizable jobs 396
 - service classes
 - description 384
 - examples 387
 - service level goals 384
 - submitting jobs 385
 - throughput goals 384
 - velocity goals 384
 - violation period 395
- slot limits 421
- slot reservation
 - resizable jobs 437
- SLOT_POOL parameter
 - in lsb.queues 352
- SLOT_SHARE parameter in lsb.queues 352
- slots
 - viewing resource allocation limits (blimits) 434
- soft resource limits
 - data segment size 602
 - description 595
 - example 599
 - file size 603
 - memory usage 603
 - stack segment size 606
- software licenses
 - counted 277
 - floating
 - dedicated queue for 280
 - description 278
 - host locked 277
 - interactive jobs competing with batch jobs 281
 - managing 281
- span resource requirement string
 - resizable jobs 328
- span string 327
- special characters
 - defining host names 97, 101
- specifying
 - license server TCP port 189
- spooling. *See* command file spooling, job file spooling

- SSH 635
- SSUSP job state
 - description 130
 - overview 120
- stack segment size limit 606
- STACKLIMIT parameter in `lsb.queues` 605
- standard input and error
 - splitting for interactive jobs 654
- standard input and output
 - job arrays 528
 - passing data between `esub` and `eexec` 641
- standard output and error
 - redirecting to a file 667
- started job dependency condition 485
- static job priority
 - absolute job priority scheduling 496
- static priority fairshare 367
- static resources
 - See also* individual resource names
 - description 257
 - shared
 - reserving 275
- STATUS
 - bhosts 60
- status
 - closed in bhosts 65
 - displaying license server (`lmstat`) 174
 - job arrays 530, 534
 - load index 253
 - viewing
 - queues 108
 - WAIT for chunk jobs 522
- stderr and stdout
 - redirecting to a file 667
 - splitting for interactive jobs 654
- stdin and stdout
 - passing data between `esub` and `eexec` 641
- STOP_COND parameter in `lsb.queues` 644
- stopping
 - FLEXlm server (`lmdown`) 174
- string resources 248
- SUB_TRY_INTERVAL parameter in `lsb.params` 121
- subfactors
 - absolute job priority scheduling 496
- submission executable (`esub`) 632
- submission options
 - embedding for interactive jobs 660
- success exit values
 - application profile configuration 403
- SUCCESS_EXIT_VALUES parameter in
 - `lsb.applications` 403
- Sun Network Information Service/Yellow Pages. *See* NIS
- supported file systems 751
- SUSPEND job control action
 - default 643
- suspended jobs
 - resuming 614
 - states 121
 - viewing resource allocation limits (`blimits`) 434
- suspending conditions
 - configuring 613
 - viewing 613
- suspending reason
 - viewing 121, 614
- suspending thresholds 614
- swap space
 - load index 255
 - suspending conditions 612
 - viewing resource allocation limits (`blimits`) 434
- SWAPLIMIT parameter in `lsb.queues` 606
- switched jobs
 - resizable jobs 579
- SWP absolute job priority scheduling factor 495
- swp load index
 - description 255
 - suspending conditions 612
 - viewing resource allocation limits (`blimits`) 434
- syslog.h file 759
- system overview 193
- T**
- task control
 - with `lstcsh` 798
- task lists
 - and `lstcsh` 794
 - changing memberships 794
- tasks
 - file access 667
 - running same on many hosts in sequence 665
 - selecting host to run on 664
 - starting parallel 541
- TCP service port numbers
 - configuring for NIS or NIS+ databases 88
 - registering for LSF 87
- tcsh
 - version and `lstcsh` 795
- temp space
 - suspending conditions 612
 - viewing resource allocation limits (`blimits`) 434
- temporary license 156
- TERMINATE job control action 644
- TERMINATE_WHEN parameter in `lsb.queues`
 - changing default SUSPEND action 647
 - TERMINATE job control action 644
- TerminateProcess() system call (Windows)
 - job control actions 645
- THREADLIMIT parameter in `lsb.queues` 605

Index

- threads
 - job limit 605
- thresholds
 - exited job exceptions 105
 - host and queue 36
 - idle job exceptions 116
 - job exit rate for hosts 105, 145
 - job overrun exceptions 116
 - job underrun exceptions 116
 - scheduling and suspending 614
 - tuning for LIM 725
- tilde (~)
 - not operator
 - host partition fairshare 342
 - host-based resources 274
- time expressions
 - creating for automatic configuration 286
 - logical operators 286
- time normalization
 - CPU factors 607
- time values
 - specifying 285
- time windows
 - syntax 285
- time-based configuration
 - automatic 287
 - commands for checking 289
- time-based resource limits 291
- time-based slot reservation
 - resizable jobs 447
- timing level
 - commands for daemons 788
- /tmp directory
 - default LSF_LOGDIR 206, 760
- tmp load index
 - description 255
 - suspending conditions 612
 - viewing resource allocation limits (blimits) 434
- /tmp_mnt directory 770
- troubleshoot
 - service error states 217
 - using multiple log files 210
- troubleshooting 187
- type keyword
 - cu string 548
- type static resource 66, 257
- U**
- UDP service port numbers
 - registering for LSF 87
- UJOB_LIMIT parameter in lsb.queues 427
- unavail host status
 - bhosts command 60
 - lsload command 61
- status load index
 - status load index 254
- uncondensed host groups
 - viewing 64
- underrun job exceptions
 - configuring 116
 - description 116, 145
 - viewing with bjobs 124
 - viewing with bqueues 110
- UNICOS accounting 758
- UNIX directory structure
 - example 49
- UNIX/Windows user account mapping
 - scope 222
- unlicensed cluster 161
- unlicensed host status
 - bhosts command 60
 - lsload command 61
 - status load index 254
- unreach host status
 - bhosts command 60
- update interval 762
 - duplicate event logging 762
- updating a license 170
- usage limits. *See* resource usage limits
- usage string 320
- user accounts
 - about 194
- user authentication
 - security 738
- user groups
 - configuring external user groups 153
 - external 153
 - overview 151
 - specifying 364
 - viewing information about 149
- user groups and limits 424
- user home directories
 - shared 37
- user priority
 - description 338
 - formula 339
- user share assignments 337
- USER_ADVANCE_RESERVATION parameter in lsb.params
 - obsolete parameter 457
- USER_NAME parameter in lsb.users 152
- USER_NAME parameter in lsb.users file 152
- USER_SHARES parameter in lsb.hosts 152
- USER_SHARES parameter in lsb.hosts file 152
- user-assigned job priority 490
- user-based host partition fairshare
 - resource usage measurement 339
- user-based queue-level fairshare

- resource usage measurement 339
- users
 - viewing information about 149
 - viewing jobs submitted by 123
 - viewing resource allocation limits (blimits) 434
 - viewing shares 150
- USERS parameter in lsb.queues 152
- USERS parameter in lsb.queues file 152
- user-specified job requeue 510
- /usr/include/sys/syslog.h file 759
- %USRCMD string in job starters 628
- USUSP job state
 - description 130
 - overview 120
 - suspending and resuming jobs 130
- ut load index
 - built-in resource 254
 - select resource requirement string 310
- utmp file registration on IRIX
 - enabling 661

V

- variables. *See* individual environment variable names
- vendor daemon (lsf_ld) 160
- verifying LSF floating client 185
- version information, displaying in FLEXIm (lmver) 174
- viewing
 - configuration errors 58
- viewing condensed and uncondensed 64
- violation period
 - SLA scheduling 395
- virtual memory
 - load index 255
 - suspending conditions 612
- virtual memory limit 606
- vmstat 255

W

- WAIT status of chunk jobs
 - description 522
 - viewing 124
- wall-clock run time limit 605
- weekly planner for advance reservation (brsvs-p) 468
- wildcards
 - defining host names 97, 101
- Windows
 - default directory structure 50
 - job control actions 645
 - TerminateProcess() system call
 - job control actions 645
- windows
 - dispatch 478
 - run 477
 - time 285

- Windows Event Viewer 719
- workarounds to lsrcp limitations 754

X

- X applications
 - running with bsub 658
- xterm
 - starting in LSF Base 668

Y

- ypbind daemon 90
- ypcat hosts.byname 90
- ypmake command 89

