

**November 2007**



## **Performance of Large Journaling File Systems**

## Table of Contents

---

Preface.....	4
Level of Understanding.....	4
Objectives .....	4
Summary .....	4
Hardware equipment and software environment .....	7
Server hardware .....	7
Server software .....	8
Environment.....	9
Workload description.....	9
IOzone.....	9
Format utility .....	10
Filling up the file systems.....	10
dbench.....	11
System setup .....	11
Linux installation .....	11
Linux disk I/O features .....	12
Attaching FCP disks .....	12
I/O scheduler.....	12
Read ahead.....	12
Logical volumes.....	13
Multipath.....	13
File systems.....	15
Storage server disk usage.....	15
Instruction sequence to prepare the measurement .....	17
Understanding the CPU utilization .....	17
Results.....	18
Finding the best logical volume configuration .....	18
Using different multipath modes .....	18
Conclusion from the measurements.....	18
Scaling disks in failover multipath mode.....	19
Conclusion from the measurements.....	19
Finding the best stripe size for the logical volume .....	19
Conclusion from the measurements.....	19
Run times for the format utility .....	20
Run times to fill up the file system to 40% utilization .....	21
Measurements on a 40% filled file system .....	22
IOzone workload.....	23
Sequential I/O readers.....	23
dbench workload.....	27
Measurements on an empty file system .....	29
dbench workload.....	29
Appendix A: Other Sources of Information.....	31

## Preface

### *Level of Understanding*

Knowledge and understanding of Linux<sup>®</sup> and Linux file systems will help you understand the results detailed in this paper. For some parts, basic knowledge of the logical volume manager and device mapper is helpful.

## Objectives

Linux file system sizes have grown rapidly over the last few years with a size of several terabytes being common. This is especially true in the mainframe area for applications like database systems or large backup volumes for IBM Tivoli<sup>®</sup> Storage Manager (TSM). Such large file systems are related to having a large amount of metadata to manage the file data. The effort of finding and navigating inside a file system of 1 terabyte is considerably higher than for a file system size of 10 GB. This study was intended to show how current file system types, like EXT3, XFS, and ReiserFS v3 behave with a file system of 1.6 TB in a multi processor environment on Linux on IBM System z<sup>™</sup>.

Because all file systems are expected to perform equally on an empty file system, regardless of the file system size, the file system we used was filled up to 40% to emulate a file system in use.

The file system itself was created by using a striped logical volume including failover handling by using two paths to one disk. The best number of disks and the best stripe size was evaluated, before we analyzed the file systems. We wanted to see the impact of:

- *The number of disks*
- *The stripe size*
- *The multipath environment*
- *The various file systems*

## Summary

This section provides a short summary of our test results. Our test results and recommendations are specific to our environment. Parameters useful in our environment might be useful in other environments, but are dependent on application usage and system configuration. You will need to determine what works best for your environment. For our detailed test results information, see [Results](#).

All tests were done with FCP/SCSI disks.

**October 2007**

Our summary results regarding the correct configuration for the logical volumes follow:

- *The singlepath multipath mode can give the best throughput, but there is no fault tolerance. For high availability we recommend the failover multipath mode. The throughput was close to the results of a single path connection, which were very good. We used the failover multipath mode for the rest of our measurements.*
- *We varied the number of physical disks of a logical volume from 4 disks up to 32 disks. The best throughput and lowest CPU utilization was seen with 8 and 16 disks.*
- *The best stripe size for the logical volume depends on the access pattern. We found a good compromise at 64 KB, which is also the default. Sequential and random write workloads were best at 128 KB and larger. Random read had its peak at 16 KB and sequential read worked well with everything between 16 KB and 128 KB.*

Our summary results regarding our tests with the journaling file system follow:

- *The format times for the file systems were very short for all the file system types we tested. Creating a file system on a 1.6 TB volume takes between some seconds and one and a half minutes. Even the longest time might be negligible.*
- *When filling the 1.6 TB file system with 619 GB of data, we found that the fastest journaling file system was EXT3. XFS needed a little bit longer. Both consumed the same amount of CPU utilization for this work.*
- *Two different disk I/O workloads were tried on the non-empty file system. First we used IOzone, a file I/O workload with separate phases of sequential/random and read/write access on dedicated files. Second we tried dbench, which generated a file server I/O mix on a larger set of files with various sizes.*
- *The IOzone workload causes significant metadata changes only during the initial write of the files. This phase was not monitored during our tests. Overall XFS was the best file system regarding highest throughput and lowest CPU utilization. XFS was not the best in all disciplines, but showed the best average, being even better than the non-journaling file system, EXT2.*
- *dbench causes a lot of metadata changes when creating, modifying, or deleting larger numbers of files and directories. This workload is closer to a customer-like workload than IOzone. In this test, XFS was the best journaling file system.*

October 2007

- *When comparing the measurement results from the empty file system with those from the 40% filled file system we saw no big difference with EXT2, EXT3, and ReiserFS. XFS degraded a bit more in the 40% filled case, but was still much faster than the other journaling file systems. EXT2 and EXT3 both showed equal results with empty and filled file systems. This was not expected because they distribute the metadata all over the volume.*
- *Using the index option with EXT3 did not lead to better results in our measurements.*
- *Overall we could recommend XFS as the best file system for large multiprocessor systems because it is fast at low CPU utilization. EXT3 is the best second choice.*

### **Hardware equipment and software environment**

This section provides details on the hardware and software used in our testing. Topics include:

- *Server hardware used*
- *Server software used*
- *The test environment*
- *A description of the workload used*

#### **Server hardware**

Host

18-way IBM System z9™ Enterprise Class (z9 EC), model 2094-S18 with:

- *0.58ns (1.7 GHz)*
- *2 books with 8/10 CPUs*
- *2 \* 40 MB L2 cache*
- *128 GB memory*
- *FICON® Express 2 cards*

One LPAR was used for our measurements with:

- *8 shared CPUs*
- *256 MB memory (2048 MB memory for the "Measurements on a 40% filled file system" and "Measurements on an empty file system" test cases)*
- *8 FICON Channels*
- *8 FCP Channels*

October 2007

Storage server setup  
2107-922 (DS8300)

- 256 GB cache
- 8 GB NVS
- 256 \* 73 GB disks (15,000 RPM)
- Organized in units to 8 disks building one RAID5 array (called a rank)
- 8 FCP attachments
- 8 FICON attachments

For the operating system:

- 2 ECKD™ mod9 from one rank/LCU
- 8 FICON paths

For the file system disks:

- 32 SCSI disks - 100 GB spread over 16 ranks
- 8 FCP paths

### Server software

Table 1. Server software used

Product	Version/Level
SUSE Linux Enterprise Server (64-bit)	SLES10 GA <ul style="list-style-type: none"><li>• 2.6.16.21-0.8-default (SUSE)</li><li>• 2.6.16.21-0.8-normalstacksize (built ourselves)</li></ul>
dbench	2.1 compiled for 64-bit
IOzone	3.196 compiled for 64-bit

### File systems

For this evaluation we used several popular Linux file systems. EXT2 is a good reference file system because it has no journaling and has been used over many years without major problems.

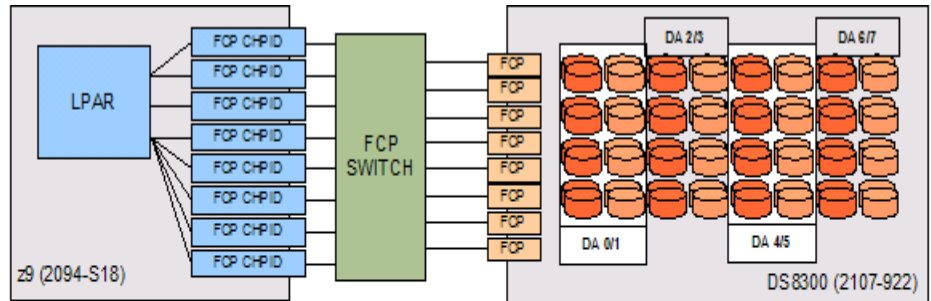
The file systems we used were:

1. EXT2 (1993) - the "old" Linux file system without any journaling.
2. EXT3 (1999) - the EXT2 file system with a journal. We used EXT3 with and without a `dir_index`.
3. ReiserFS v3 (2001) - the standard file system on SUSE Linux which was developed by Namesys (Hans Reiser).
4. XFS (1994) - the IRIX file system, which was released in 2000 as open source. This file system was developed by SGI and is one of the oldest journaling file systems.

October 2007

### Environment

Our environment consisted of an IBM System z and a DS8300 storage server. They were connected together over a switched fabric with eight 2 Gbps FCP links.



FCP CHPID = Fibre channel attachment on IBM System z  
FCP = Fibre channel attachment on the storage server  
DA = Device adapter pairs

Figure 1. Journaling test environment

On System z we used one LPAR (configured as mentioned above) which had access to the disks on the storage server. [Figure 1](#) shows the storage server on the right side. In this storage server we used up to 32 disks. Details on the storage servers are described in [Storage server disk usage](#).

### Workload description

#### IOzone

IOzone is a file system benchmark tool. This tool reads and writes to a file. We had sixteen instances running at the same time. IOzone can perform sequential and random operations to the file.

The first write operation was only used to allocate the space on the file system. For our measurements, only the rewrite and read results were used.

IOzone setup used for our measurements:

- Sequential I/O
  - Write, rewrite (cmdline option "-i 0"), and read (cmdline option "-i 1") of a 2000 MB (cmdline option "-s 2000m") file
  - 16 threads (cmdline option "-t 16") working on one file system
  - 64 KB record size (cmdline option "-r 64k")
  - Random I/O

October 2007

- Write, random write, and random read (cmdline option "-i 2") of a 2000 MB file (cmdline option "-s 2000m")
- 16 threads (cmdline option "-t 16") working on one file system
- 64 KB record size (cmdline option "-r 64k")
- The random I/O modes produce separate values for read and write throughput, but only one value for CPU utilization because the reads and writes are mixed

Other command line options we used for our measurements follow:

- "-C" - Show bytes transferred by each child in throughput testing
- "-e" - Include flush (fsync, fflush) in the timing calculations
- "-R" - IOzone will generate an Excel-compatible report to standard out
- "-w" - Do not unlink temporary files when finished using them

For a detailed description of these and all possible parameters see the documentation located at <http://www.iozone.org/>.

### Format utility

The format utility simply formats the file system. We used the tools provided with the file system tools. To compare the time it took to format the file systems we collected sysstat data (using sadc with a resolution of one second) and CPU times (using the time command) while the format utility was running.

### Filling up the file systems

To fill the file system up to 40% we needed a large amount of data. We decided to use a Linux vanilla kernel 2.6.19 archive (240690 Kbytes). This archive was an uncompressed tar, containing 1252 directories and 20936 files.

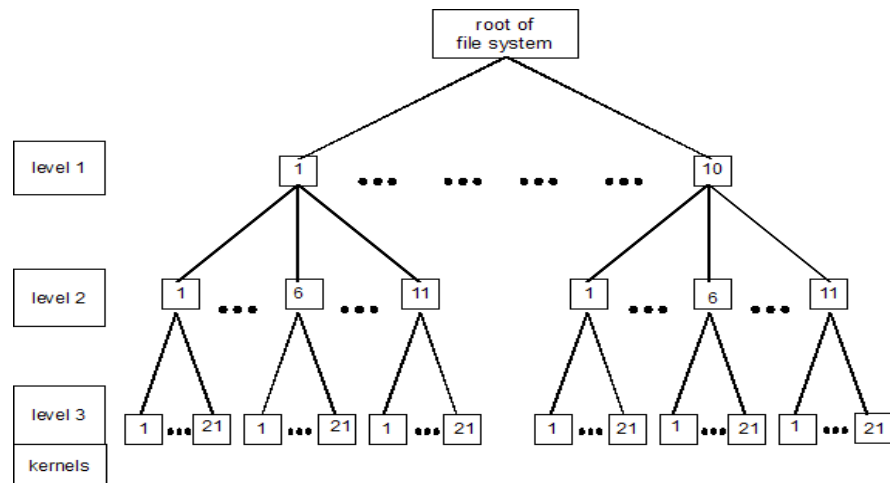


Figure 2. Schemata of extracted kernel directories

**October 2007**

The directory structure produced had three levels. Each first level directory, out of ten, contained eleven second level directories. Each level 2 directory contained 21 subdirectories, each had one extracted kernel.

The CPU times were collected by sysstat using sadc with a resolution of one second.

The tar archive was extracted 2310 times into the 21 level three directories of one level two tree in parallel. Because the tar file is not compressed this load consisted mostly of file system operations. The totally used file system space was 619 GB.

The execution time and the CPU utilization were measured.

### **[dbench](#)**

dbench ([samba.org/ftp/tridge/dbench](http://samba.org/ftp/tridge/dbench)) is an emulation of the file system load function of the Netbench benchmark. It does all the same I/O calls that the smbd server daemon in Samba would produce when it is driven by a Netbench workload. However, it does not perform network calls. This benchmark is a good simulation of a real server setup (such as Web-server, proxy-server, mail-server), in which a large amount of files with different sizes and directions have to be created, written, read, and deleted.

dbench takes only one parameter on the command line, which is the number of processes (clients) to start. Issuing the command "dbench 30" for example, creates 30 parallel processes of dbench. All processes are started at the same time and each of them runs the same workload. The workload for each dbench process is specified by a client.txt configuration file in the working (testing) directory. It consists of a mixture of file system operations executed by each dbench process. dbench runs with n parallel processes and delivers only one value as a result. The resulting value is an average throughput of the file system operations described in client.txt and measured in megabytes per second.

With a large amount of memory, dbench measurements can be used to detect the effects of memory scaling.

For the dbench workload used for our measurements the number of clients was scaled as shown in the following sequence: 1, 4, 8, 12, 16, 20, 32, 40, 46, 50, 54,

### **[System setup](#)**

In this section we detail changes we made to our system setup.

#### **[Linux installation](#)**

We used an SLES10 installation on System z with developer tools.

October 2007

### [Linux disk I/O features](#)

#### [Attaching FCP disks](#)

For attaching the FCP disks we followed the instructions described in Chapter 6, "SCSI-over-Fibre Channel device driver," section "Working with the zfcpc device driver" in *Linux on System z - Device Drivers, Features, and Commands*, SC33-8289. This book can be found at:

<http://www.ibm.com/developerworks/linux/linux390/>

Choose your stream (for example, October 2005 stream) and then click on the Documentation link.

#### [I/O scheduler](#)

For our measurements we used the deadline I/O Scheduler. This setting was made to the kernel option at boot time (option "elevator=deadline" in the zipl.conf file).

After booting with this scheduler (verify with 'grep scheduler /var/log/boot.msg') we modified the following parameters in /sys/block/<sd[a-z][a-z]>/queue/iosched for each disk device concerned:

front\_merges to 0 (default 1)

Avoids scanning of the scheduler queue for front mergers, which rarely occur. Using this setting saves CPU utilization and time.

write\_expire to 500 (default 5000)

read\_expire to 500 (default 500)

We set write to the same expiration value as read to force the same timeout for read and write requests.

writes\_starved to 1 (default 2)

Processes the next write request after each read request (per the default, two read requests are done before the next write request). This handles reads and writes with the same priority.

These settings are mostly suitable for servers such as databases and file servers. The intention of these settings is to handle writes in the same manner as reads.

#### [Read ahead](#)

The read ahead on any Linux data disk and logical volume is set to zero to measure exactly the requested I/O.

October 2007

### [Logical volumes](#)

The logical volume was striped over several disks depending on the test case. Never use the devices `/dev/sd*` for your logical volume. You will bypass the device mapper. Use the device nodes under `/dev/disk/by-id/` instead.

### [Multipath](#)

The multipath setup is quite simple, but you must check that the utilization of your fibre channel paths is balanced. The critical point here is the order of the device names. The Multipath / Device mapper uses the old device names such as `sda`, `sdb`, and so on. This means that the device `sdaa` comes before device `sdb` which may cause an unbalanced load on the fibre channel paths.

To make sure the fibre channel path utilization is optimal, we used the following sequence for adding disks:

1. *Connect the first path to the disks in ascending order over all eight fibre channel paths.*
2. *Connect the second path to the disks in descending order over all eight fibre channel paths.*

If you use two paths to every disk you should add the primary path to the first disk, the secondary path to the first disk, the primary path to the second disk, the secondary path to the second disk and so on. With this you will get a balanced fibre channel usage.

The two path setup is described in [Table 2](#).

October 2007

[Table 2. FCP disk setup with 16 disks](#)

Attached order	Disk ID	Fibre channel path	Device adapter pair	Internal server	Rank (RAID Array)
1	0x1413	1	0-1	0	9
2	0x1413	8	0-1	0	9
3	0x1613	1	0-1	0	12
4	0x1613	8	0-1	0	12
5	0x1513	2	0-1	1	11
6	0x1513	7	0-1	1	11
7	0x1713	2	0-1	1	14
8	0x1713	7	0-1	1	14
9	0x1013	3	2-3	0	1
10	0x1013	6	2-3	0	1
11	0x1213	3	2-3	0	4
12	0x1213	6	2-3	0	4
13	0x1113	4	2-3	1	3
14	0x1113	5	2-3	1	3
15	0x1313	4	2-3	1	6
16	0x1313	5	2-3	1	6
17	0x1c13	5	4-5	0	24
18	0x1c13	4	4-5	0	24
19	0x1e13	5	4-5	0	29
20	0x1e13	4	4-5	0	29
21	0x1d13	6	4-5	1	26
22	0x1d13	3	4-5	1	26
23	0x1f13	6	4-5	1	31
24	0x1f13	3	4-5	1	31
25	0x1813	7	6-7	0	16
26	0x1813	2	6-7	0	16
27	0x1a13	7	6-7	0	21
28	0x1a13	2	6-7	0	21
29	0x1913	8	6-7	1	18
30	0x1913	1	6-7	1	18
31	0x1b13	8	6-7	1	23
32	0x1b13	1	6-7	1	23

For your setup you will use either "failover" or "multibus" as the policy for the multipath daemon. Be sure that you set this using "multipath -p <mypolicy>" after starting the multipath daemon.

### File systems

The file systems were created with their default values using the following commands:

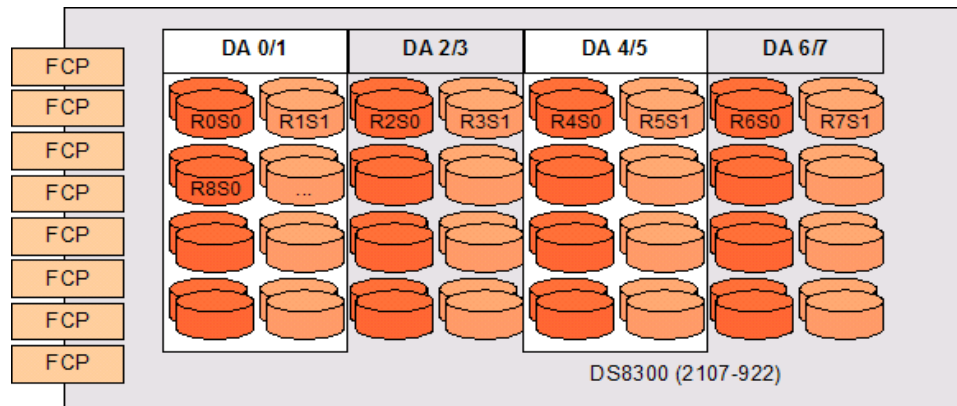
- `mkfs.ext2 -q`
- `mkfs.ext3 -q`
- `mkfs.ext3 -q -O dir_index`
- `mkfs.reiserfs -q`
- `mkfs.xfs -q -f`

The option "-q" means quiet execution. In most commands this shows no interactive output and no prompts. The option "-f" at `mkfs.xfs` tells the `mkfs` that it is OK to override an existing file system on the target device.

### Storage server disk usage

To get good results you should choose the disks from your storage server with care. For example, if you have a DS8300, choose the disks from different ranks (RAID arrays). See the storage server manual (*IBM System Storage DS8000 Series: Architecture and Implementation, SG24-6786*) for details.

The figure below shows the disk layout used for our tests.



DA = device adapter pairs  
RxSy = rank x server y, for example, R1S0 is rank 1 accessed from server 0  
FCP = Fiber channel attachment on DS8300

Figure 3. Storage server disk layout

October 2007

For performance reasons, we selected disks evenly over as many ranks as possible (which was 16) and over both servers within the storage server. The disks with the white background were controlled by the first internal server of the DS8300 and the other disks were controlled by the second internal server. The disks inside the storage server were organized into 16 ranks and connected via four device adapter pairs.

#### [Instruction sequence to prepare the measurement](#)

We performed the steps shown below to execute each run:

- *Boot linux*
- *SLES10 default zipl with one change: elevator=deadline*
- *Attach SCSI disks*
- *Start multipath daemon*
- *service multipathd start*
- *Configure multipath policy*
- *multipath -p multibus or multipath -p failover*
- *Prepare the disks to use with LVM (each disk has one big partition)*
- */sbin/pvcreate -f -y /dev/disk/by-id/scsi-3...1013-part1*
- *Create volume group*
- */sbin/vgcreate largefs /dev/disk/by-id/scsi-3...1413-part1*
- */dev/disk/by-id/scsi-3...1513-part1 ...*
- *Create logical volume*
- */sbin/lvcreate -i 16 -l 64 -L 1.56T -n lvol0 largefs*
- *Set read\_ahead for physical and logical volumes to 0*
- *lvchange -r 0 /dev/16disks/lvol0*
- *blockdev --setra /dev/disk/by-id/scsi...361013 /dev/disk/by-id/scsi-3...1113 ...*
- *Set parameters for the I/O scheduler (see [I/O scheduler](#))*
- *Perform run (contains the formatting of the file system)*

#### [Understanding the CPU utilization](#)

The CPU utilization reports show five different types of CPU load. These types are defined in [Table 3](#), which comes from the sar main page.

[Table 3. CPU utilization types](#)

%idle	Percentage of time that the CPU or CPUs were idle and the system did not have an outstanding disk I/O request.
%iowait	Percentage of time that the CPU or CPUs were idle during which the system had an outstanding disk I/O request.
%system	Percentage of CPU utilization that occurred while executing at the system level (kernel).
%nice	Percentage of CPU utilization that occurred while executing at the user level with nice priority.
%user	Percentage of CPU utilization that occurred while executing at the user level (application).

## **Results**

This section provides our detailed test results and conclusions.

### ***Finding the best logical volume configuration***

To find the best logical volume configuration for our testing environment, we performed the following tests:

- *Using different multipath modes*
- *Scaling disks in failover multipath mode*
- *Finding the best stripe size for the logical volume*

### ***Using different multipath modes***

Our goal in this test case was to determine the best multipath mode.

To exploit fault tolerance we used the following multipath modes:

- *failover (one active path and one standby path)*
- *multibus (two active paths)*
- *singlepath (no fault tolerance) - used for performance comparisons*

We used the IOzone workload for these tests. The workload operates on a striped logical volume (LVM2) with 4, 8, 16, 24, and 32 physical disks. The file system was EXT2. The stripe size used was 64 KB (the default).

### ***Conclusion from the measurements***

The failover multipath mode behaves similarly to the singlepath mode, except for random readers. The failover mode provides the fault tolerance with the lowest impact on performance. The multibus multipath mode always has the lower throughput. Based on these results we decided to use the failover mode for our remaining measurements.

### **Scaling disks in failover multipath mode**

Our goal with this test case was to find the best number of physical volumes within our striped logical volume. In this test case we continued using the IOzone workload. We used the same setup that we used in [Using different multipath modes](#). The file system was EXT2.

We used the failover multipath mode (as determined in our previous tests) and four different physical volume configurations:

- *4 physical disks*
- *8 physical disks*
- *16 physical disks*
- *24 physical disks*
- *32 physical disks*

The best configuration was expected to have the best throughput and least amount of CPU utilization (user and system CPU together). The stripe size was 64 KB (the default).

### ***Conclusion from the measurements***

The best throughput at low CPU utilization was seen with 8 and 16 disks. For our remaining measurements we used 16 disks.

### **Finding the best stripe size for the logical volume**

To determine the best stripe size for our logical volume we used the same IOzone workload as in the two previous test cases. For this test we scaled the stripe size of the logical volume from 8 KB to 512 KB.

We wanted to determine which stripe size is the best for a logical volume of 8 disks and which stripe size is the best for a logical volume of 16 disks.

### ***Conclusion from the measurements***

We have two optimal stripe sizes. 32 KB is the optimal stripe size for random workloads and 128 KB is optimal for sequential workloads. We assumed that most real-life workloads are somewhere in between sequential and randomized. Therefore, we decided to use a 64 KB stripe size as a good overall solution. A database server with many small randomized reads would get better results with a smaller stripe size.

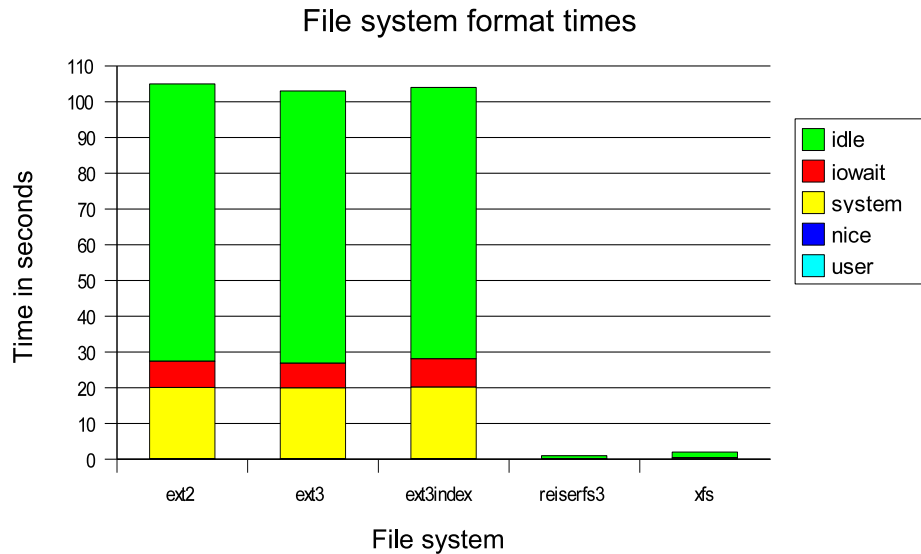
For all further measurements we used a striped (64 KB) logical volume (LVM2) with 16 physical disks and a Linux system with 2 GB of memory.

[Run times for the format utility](#)

Formatting a file system is a fast and easy task, but how long does the formatting of 1.6 TB file system actually take? Usually, there is no time limitation the first time this task is performed. However, the format time needed in a disaster recover case may be critical. This test case was used to find an average format time.

We used the UNIX<sup>®</sup> time command and sysstat to measure the format time.

[Figure 4](#) and [Figure 5](#) show the total run time to format the file system. The lower values were the better times. The bars in the charts are split to also show the CPU utilization. Non-appearing colors have a value of zero.

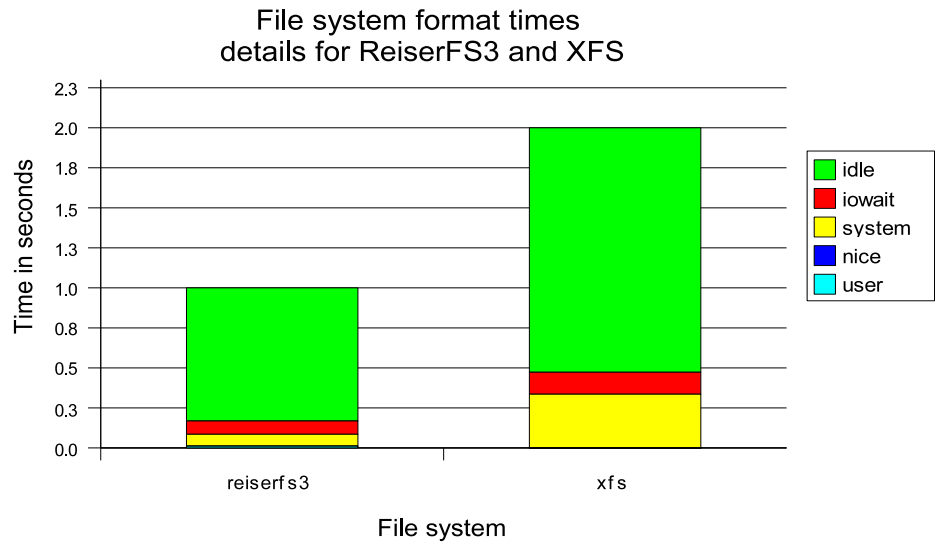


[Figure 4. Run times for the format utility - file system format times](#)

**Note:**

CPU utilization types are explained in [Understanding the CPU utilization](#).

[Figure 5](#) is the detailed view of the ReiserFS3 and XFS file systems. These file systems are much faster than the EXT2 and EXT3 file systems so the overall chart does not show the details of the CPU utilization.



[Figure 5. Run times for the format utility - file system format times, ReiserFS3 and XFS details](#)

**Note:**

CPU utilization types are explained in [Understanding the CPU utilization](#).

**Observations**

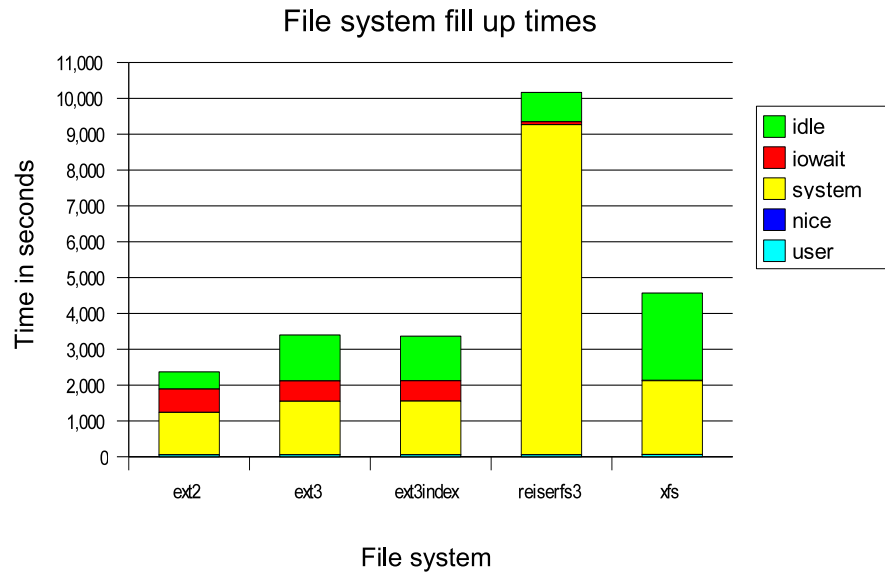
EXT2 and EXT3 are writing the metadata all over the file system on the disk, which takes a little extra time. However, both are still done in less than two minutes. All other file systems are writing only some information at the beginning, which takes between one and four seconds. There is almost no "nice" or "user" CPU consumption.

**Conclusion**

ReiserFS and XFS, are very fast when formatting the disk. The EXT2 and EXT3 are writing some data over the whole disk, which takes a little more time. However, the format times are also low enough that they might be negligible.

[\*Run times to fill up the file system to 40% utilization\*](#)

In this test case we measured the time to fill our 1.6 TB file system to 40% with a workload as described in [Filling up the file systems](#) to fill up our file system. We measured the fill up times and CPU utilization. We changed the kernel's stack size to normal because, with XFS and the small stack size (the default), we hit a kernel stack overflow. The Linux kernel was self-built with normal stack size.



[Figure 6. File system fill up times](#)

**Note:**

CPU utilization types are explained in [Understanding the CPU utilization](#).

[Figure 6](#) shows the total fill up time in seconds. The bars in the chart are split to show the CPU utilization. Colors which do not appear have a value of zero.

**Observations**

We have the following observations from these test runs:

- *EXT2 is the fastest file system running at about 40 minutes. EXT3, with and without an index, took about one hour, which is close to the EXT2 time.*
- *XFS needs about one hour and 20 minutes and has no I/O wait.*
- *ReiserFS Version 3 needs about two hours and 50 minutes and has high system CPU utilization.*
- *As in [Run times for the format utility](#) we have no "nice" and only very little "user" CPU consumption.*

**Conclusion**

The best journaling file system for the fill up is EXT3. XFS needs a little bit longer, but the CPU utilization is about the same.

[Measurements on a 40% filled file system](#)

**October 2007**

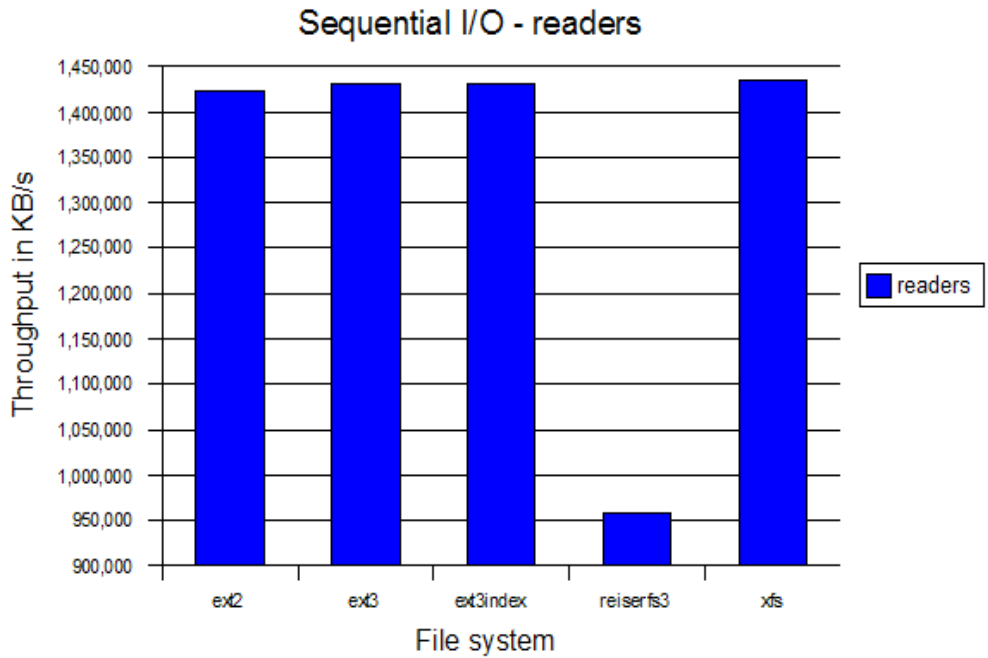
On the 40% filled file system we used the IOzone and dbench workloads to measure the behavior of the different file systems.

***[IOzone workload](#)***

The following charts show the results of the IOzone workload on the 40% utilized 1.6 TB file system. We varied the file systems in this test. Be aware that on the throughput charts the Y-axis does not start at zero!

***Sequential I/O readers***

[Figure 7](#) and [Figure 8](#) show the throughput and CPU utilization for the sequential read I/O.



[Figure 7. Disk I/O measurements - sequential I/O readers throughput](#)

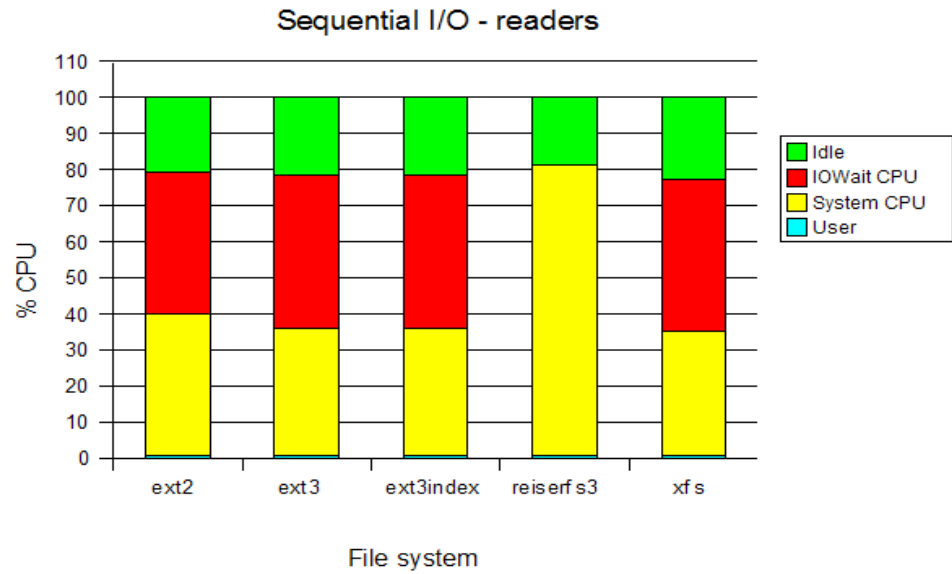


Figure 8. Disk I/O measurements - sequential I/O readers CPU utilization

**Note:**

CPU utilization types are explained in [Understanding the CPU utilization.](#)

**Observations**

XFS has the best read performance and ReiserFS has the worst. The performance of EXT3, with and without an index, is very close to XFS and slightly better than EXT2. XFS also has the lowest CPU utilization, but EXT3 and EXT2 are very close.

**Sequential I/O writers**

Figure 9 and Figure 10 show the throughput and CPU utilization for the sequential write I/O.

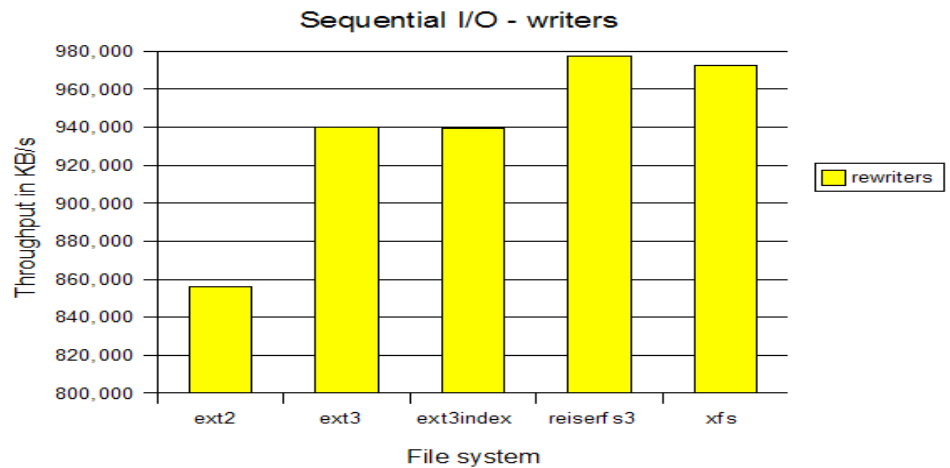


Figure 9. IOzone workload sequential I/O writers throughput

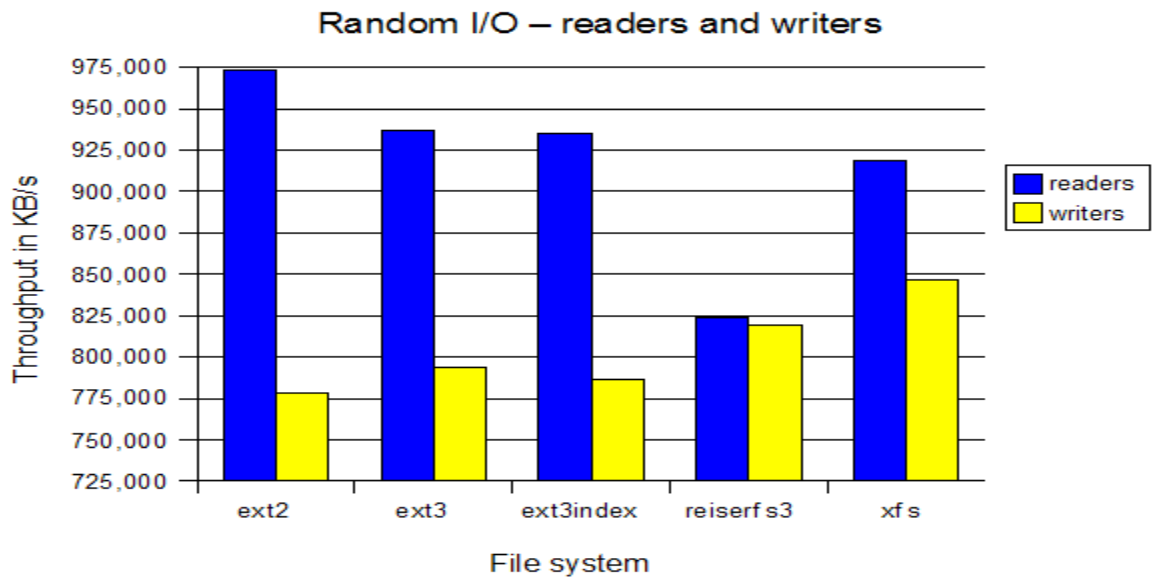
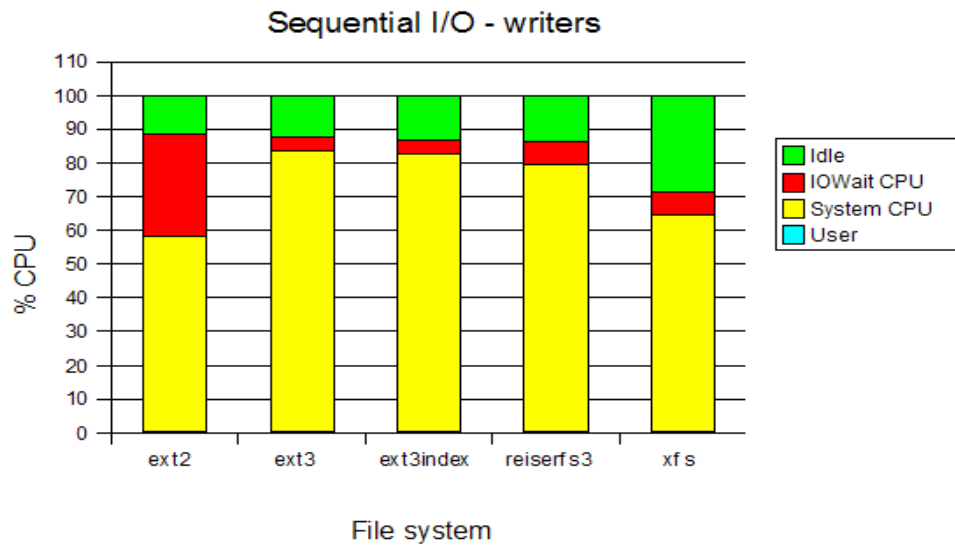


Figure 10. IOzone workload sequential I/O writers CPU utilization

**Note:**

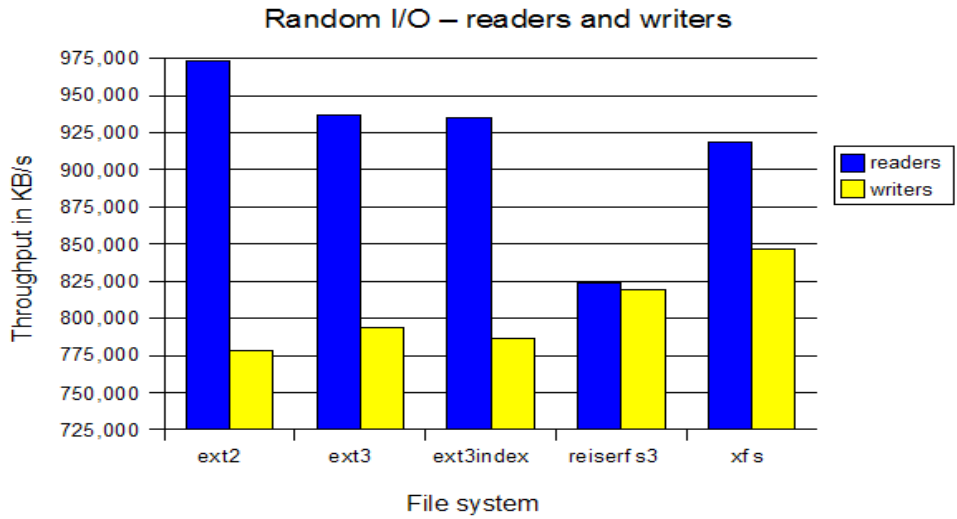
CPU utilization types are explained in [Understanding the CPU utilization](#).

**Observations**

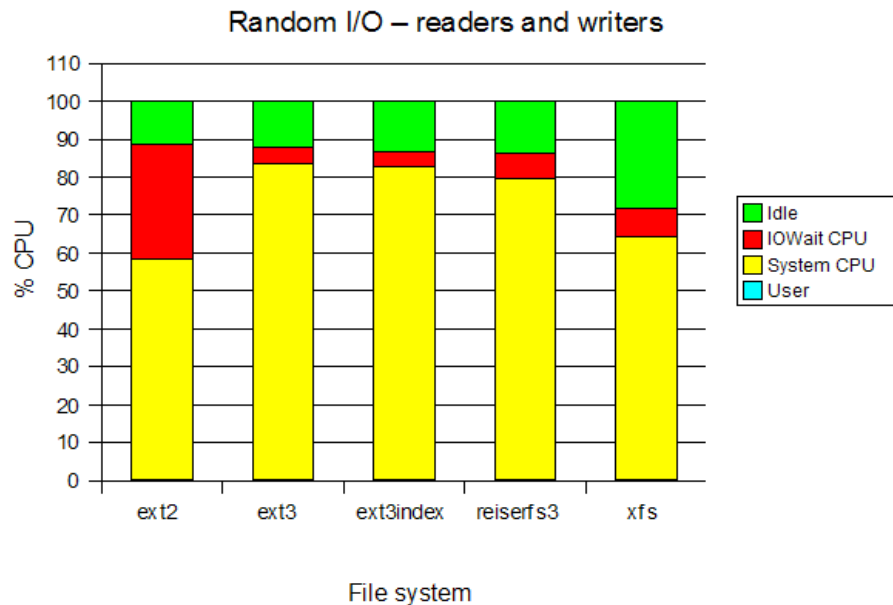
ReiserFS has the best throughput. XFS is only 1% below ReiserFS and EXT3 is 4% lower than ReiserFS (regardless of whether an index is used). The lowest CPU utilization was with EXT2. From the journaling file systems, XFS has the lowest CPU utilization, which is also very close to the EXT2 CPU utilization, but the throughput for XFS is higher.

### Random I/O

[Figure 11](#) and [Figure 12](#) show the throughput and CPU utilization for the random write and read I/O.



[Figure 11. IOzone workload random I/O readers and writers throughput](#)



[Figure 12. IOzone workload random I/O readers and writers CPU utilization](#)

### Note:

CPU utilization types are explained in [Understanding the CPU utilization](#).

**Observations**

EXT2 is the best file system for reading random data from the storage server, but it is also the worst for writing random data. For EXT3, with and without an index, we see similar results, but a bit worse for reading and a bit better for writing. ReiserFS has the lowest throughput for reading random data, but has good performance for writing random data. The best throughput for writing random data is XFS. The lowest CPU utilization is with EXT2, EXT3, and XFS, while XFS has the lowest I/O wait.

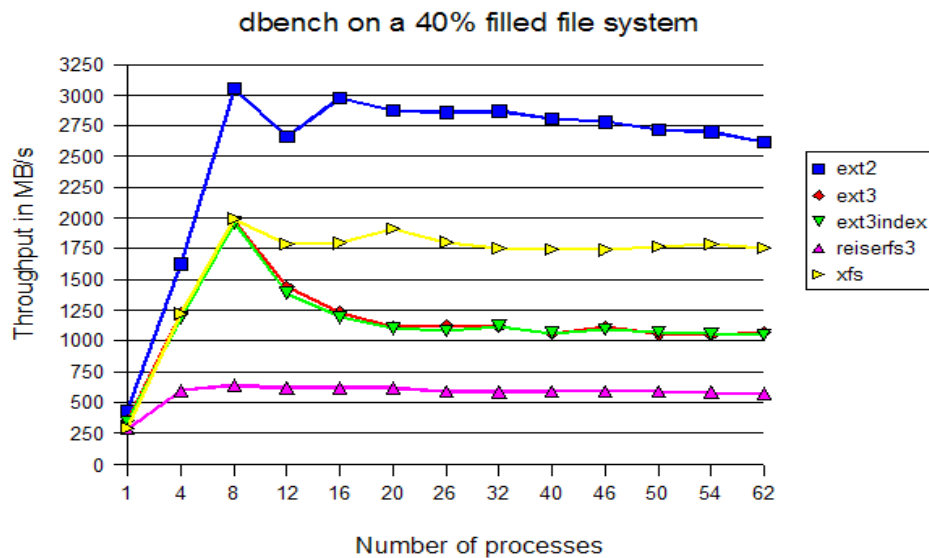
**Conclusion**

The IOzone workload has very little effort for journaling because most of the metadata changes occur during the initial write phase (creating and enlarging the file), which is not monitored here. In our tests, only the rewrite phase is shown, which only has updating the access time in the file and directory inode for metadata changes. The same is true for reading. This workload shows just the overhead related with the journaling mechanisms for the normal read and update operations with minimum meta data changes.

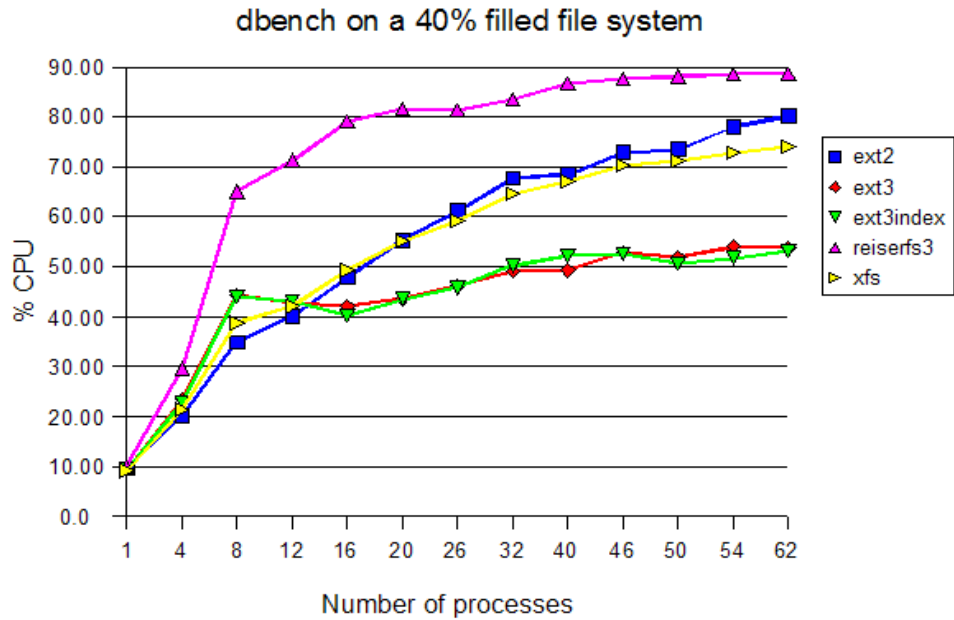
For the IOzone workload we see that XFS is the best overall journaling file system solution for both throughput and CPU utilization. Even if in some categories other file systems might be better, they have significant weaknesses elsewhere. This is also true in comparison to EXT2.

**dbench workload**

[Figure 13](#) and [Figure 14](#) show the results from the dbench workload on a 40% filled 1.6 TB file system.



[Figure 13. dbench workload on a 40% filled file system throughput](#)



[Figure 14. dbench workload on a 40% filled file system CPU utilization](#)

**Note:**

CPU utilization types are explained in [Understanding the CPU utilization](#).

**Observations**

The best performance for our dbench workload is seen with EXT2, which has no additional effort for journaling. The journaling file system with the best throughput is XFS, especially with a high number of workload generators. XFS also has up to 70% more throughput than the second best journaling file system, EXT3. Looking at the cost in terms of throughput per 1% CPU, XFS drives more throughput with the same amount of CPU.

The worst performance with the highest cost is seen with ReiserFS.

**Conclusion**

The dbench workload makes a lot of changes in metadata (creating, modifying, deleting a high number of files and directories), which causes a significant effort for journaling. This workload is much closer to a customer-like workload. On a 40% filled large volume, XFS is the best journaling file system solution with the dbench workload.

Measurements on an empty file system

On an empty file system we used the dbench workload to take some reference numbers.

dbench workload

We used the dbench workload to compare the results of the empty large file system with the results from our test case Measurements on a 40% filled file system. This test was intended to show how a 40% filled system impacts throughput. We varied the file systems. The charts show only results for EXT3 with an index because the results without the index were nearly identical.

The results for a certain file system have the same color, the empty file system curve uses the square (■) symbol, the curve for the 40% utilized file system uses

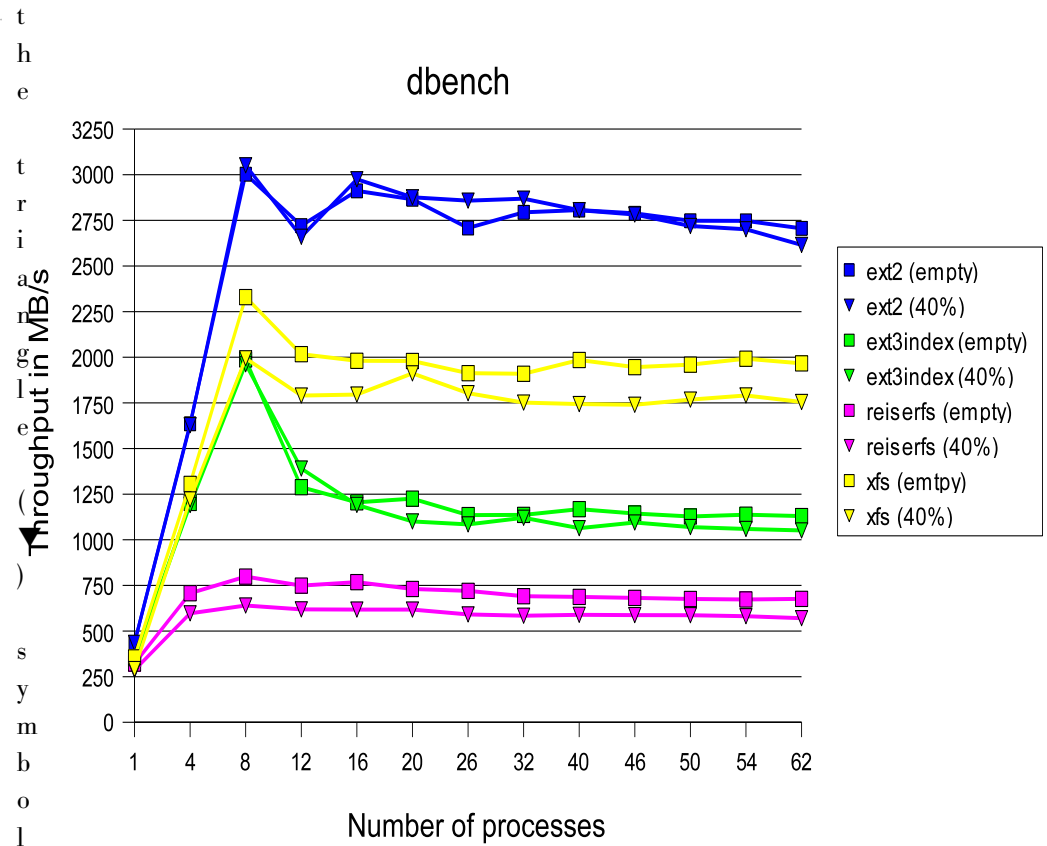


Figure 15. dbench workload throughput

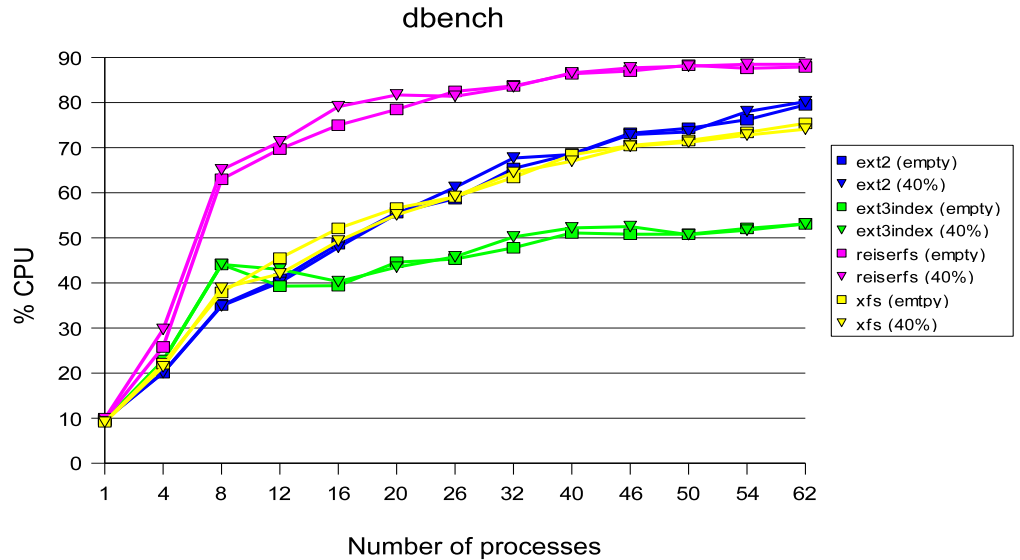


Figure 16. dbench workload CPU utilization

**Note:**

CPU utilization types are explained in [Understanding the CPU utilization.](#)

**Observations**

Throughput results for most of the file systems are only slightly degraded when the file system is filled to 40%. XFS shows a degradation in throughput of 14% between an empty and 40% filled file system, which is the largest impact so far. However, it is still much faster than the other journaling file systems. Once again there is no difference for EXT3 regardless of whether an index is used.

For all file systems, the CPU utilization is very similar for the empty and the 40% filled file system.

**Conclusion**

Comparing the results for the empty file system with the 40% filled file system shows no major differences. This was not expected, especially for EXT2 and EXT3 because they distribute the metadata all over the disk. Only XFS shows degradation in throughput, but it is still much faster than the other journaling file systems.

**Appendix A: Other Sources of Information**

- *For information on IOzone see:*  
[www.iozone.org](http://www.iozone.org)
- *For information on dbench see:*  
[samba.org/ftp/tridge/dbench/](http://samba.org/ftp/tridge/dbench/)
- *For information on Linux on System z see:*  
[www.ibm.com/servers/eserver/zseries/os/linux/](http://www.ibm.com/servers/eserver/zseries/os/linux/)
- *For information about FCP disks and other Linux device driver specifics see: Linux on System z - Device Drivers, Features, and Commands - SC33-8289*  
[www.ibm.com/developerworks/linux/linux390/](http://www.ibm.com/developerworks/linux/linux390/)
- *For information on IBM open source projects see:*  
[www.ibm.com/developerworks/opensource/index.html](http://www.ibm.com/developerworks/opensource/index.html)
- *See the following Redbooks/Redpapers for additional information:*
- *IBM System Storage DS8000 Series: Architecture and Implementation - SG24-6786*  
[www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246786.html?OpenDocument](http://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246786.html?OpenDocument)
- *Linux on zSeries: Samba-3 Performance Observations*  
[www.redbooks.ibm.com/abstracts/redp3988.html](http://www.redbooks.ibm.com/abstracts/redp3988.html)
- *Linux Performance and Tuning Guidelines*  
[www.redbooks.ibm.com/redpieces/abstracts/redp4285.html](http://www.redbooks.ibm.com/redpieces/abstracts/redp4285.html)

October 2007



© Copyright IBM Corporation 2007

IBM Corporation  
New Orchard Rd.  
Armonk, NY 10504  
U.S.A.

Produced in the United States of America  
10/07

All Rights Reserved

IBM, IBM logo, DS800, ECKD, FICON, IBM System Storage, System z, System z9, Tivoli, zSeries are trademarks or registered trademarks of International Business Machines Corporation of the United States, other countries or both.

The following are trademarks or registered trademarks of other companies

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries or both.

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

Intel is a trademark of Intel Corporation in the United States, other countries or both.

Linux is a registered trademark of Linus Torvalds in the United States and other countries.

SUSE is a registered trademark of Novell, Inc., in the United States and other countries.

Other company, product and service names may be trademarks or service marks of others.

Information concerning non-IBM products was obtained from the suppliers of their products or their published announcements. Questions on the capabilities of the non-IBM products should be addressed with the suppliers.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

THE INFORMATION CONTAINED IN THIS DOCUMENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS DOCUMENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM<sup>(R)</sup> WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS DOCUMENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS DOCUMENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT GOVERNING THE USE OF IBM SOFTWARE

ZSW03027-USEN-00