Filesystem Performance and Scalability in Linux 2.4.17

Originally published in Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference (Berkeley, CA: USENIX Association 2002). Published by Permission.

Ray Bryant, *SGI, raybry@sgi.com*
Ruth Forester, *IBM LTC, rsf@us.ibm.com*
John Hawkes, *SGI, hawkes@sgi.com*

Abstract

The Linux® kernel is unique in that it supports a wide variety of high-quality filesystems. For server systems, the most commonly used are Ext2, Ext3, ReiserFS, XFS and JFS. This paper compares the performance of these filesystems using Linux 2.4.17 and three benchmarks: *pgmeter*, an open source implementation of the Intel® Iometer benchmark; *filemark* (a version of *postmark*); and *AIM Benchmark Suite VII*. The benchmarks were run on three different systems ranging in size from a contemporary single-user workstation to a 28-processor ccNUMA machine. Although the best-performing filesystem varies depending on the benchmark and system used, some larger trends are evident in the data. On the smaller systems, the best-performing file system is often Ext2, Ext3 or ReiserFS. For the larger systems and higher loads, XFS can provide the best overall performance.

1 Introduction

One of the advantages of open source software is the tremendous number of choices available to users of such software. For example, the filesystem menu of the configuration page for the Linux 2.4.17 kernel includes 28 entries, and this does not include the network filesystems. Although a number of these filesystems are special purpose or for compatibility with other operating systems, the fact remains that there are a wide variety of filesystems available for use with Linux.

Given this large list of filesystems, a Linux user might ask the reasonable question: "Which filesystem should I use?" In many cases the answer to this question depends on functional or ease-of-use issues; in other cases performance is a deciding factor. The goal of this paper is to provide a partial answer to this question by comparing the performance of five of the more popular filesystems available under Linux: Ext2, Ext3, ReiserFS, XFS™, and JFS.

As is the case with any benchmark study, we must emphasize that the performance results reported here reflect measurements taken at a particular point in time, with a particular set of machines, with a particular Linux kernel (in our case 2.4.17), using a particular set of benchmarks and parameters, and only truly compare filesystem performance under these restrictive circumstances. Over time, we expect each filesystem discussed herein to continue to be developed, and we expect that the relative orderings of filesystem performance discussed here will change.

Nevertheless, we believe that our benchmarking study does have value in that trends that are identified here can provide broad guidance to individuals who know the characteristics of their system's workloads and who have the ability to choose which filesystem to deploy. It is also possible that these studies may uncover performance problem areas that the filesystem developers can focus on to improve the performance of their filesystems, thus helping to advance Linux kernel development.

We are aware of no other filesystem benchmarking study with as large a scope and rigor as the one we present in this paper. The best other results we are aware of are the benchmark pages at the ReiserFS site [ReiserFS] and von Hagen's benchmark studies [vonHagen]. However, those studies are only for small, uniprocessor systems. This paper is the only one we are aware of that also examines larger systems.

In the remainder of this paper, we first provide a brief overview of the filesystems we are testing. We then describe the benchmarks we will use in this study. Next, we describe the machines we used to run the benchmarks and the rules that defined how the benchmarks were run. Finally, we present the results of our experiments and discuss the implications and trends these experiments have demonstrated.

2 Filesystem Descriptions

In this section we provide a brief overview and history of the filesystems tested in this paper. Further details about these file systems can be found elsewhere [vonHagen, Galli].

We assume that the reader is familiar with basic concepts of Linux filesystems such as the filesystem *buffer cache* and *journaling* filesystems [Bovet, vonHagen].

Version information for the filesystems and supporting tools used in this paper are given in the Appendix.

2.1 Ext2

Designed by Wayne Davidson (with Stephen Tweedie and Theodore Ts'o) as an enhancement of the *ext* filesystem from Remy Card, Ext2 [Ext2] is the standard Linux filesystem. We include Ext2 in our tests because it provides a baseline of performance that is familiar to many users.

2.2 Ext3

An enhancement of Ext2 developed by Stephen Tweedie [Tweedie], Ext3 uses the same disk format and data structures as Ext2, but in addition supports journaling. This makes conversion from Ext2 to Ext3 extremely easy—no data migration or filesystem reformatting is required. All that one needs to do is to install an Ext3-capable kernel, use the *tune2fs* program to create a journal, and remount the file system.

Ext3 is the default filesystem for Red Hat 7.2 [Ext3RH] and has been included in the standard Linux kernel since 2.4.13. It is available with many Linux distributions.

Ext3 is block based, with sequential filename directory search. Ext3 supports three journaling modes to allow the user to tradeoff performance and integrity guarantees. The default journaling mode, *data=ordered*, guarantees consistent writing of the metadata, descriptor and header blocks and provides good performance; *data=writeback* provides a somewhat lower data-integrity guarantee in favor of better performance (old data can be present in a file after a crash); and *data=journal* provides both metadata *and* data journaling.

Which of these modes has the best performance varies according to workload [Ts'o]. One would normally expect that the *data=journal* mode would perform significantly slower that the other modes due to the additional overhead of logging data changes. However, certain workloads have been found to run faster with *data=journal* (especially when reads and writes are to and from the same disks), but the reasons for this are not yet fully understood [Robbins].

In this paper, we report results for *data=writeback* and *data=ordered*. We have not included results with *data=journal* since this is a specialized mode that is not supported by the other journaling filesystems tested here.

2.3 ReiserFS

Developed by Hans Reiser, ReiserFS [ReiserFS] has been part of the standard Linux kernel since 2.4.1 and it is available with many Linux distributions. ReiserFS supports metadata journaling, and it is especially noted for its excellent small-file performance. ReiserFS uses B* Balanced Trees to organize directories, files, and data. This provides fast directory lookups and fast delete operations. Other performance features include support for sparse files and dynamic disk inode allocation.

ReiserFS supports a space-saving option called *tail-packing* that packs small files into the leaves of the B* Tree. However, this option has a performance cost and most benchmarks are run with this feature disabled. In this paper, we follow this practice and report only results with *notail*.

2.4 XFS

Based on SGI®'s Irix™ XFS filesystem technology [XFSirix], the XFS port to Linux, version 1.0, was released May, 2001 [XFS]. A large number of support tools are also distributed with XFS [vonHagen, p. 165-167].

XFS is a journaling filesystem that supports metadata journaling. XFS uses allocation groups and extent-based allocations to improve locality of data on disk. This results in improved performance, particularly for large sequential transfers. Performance features include asynchronous write ahead logging (similar to Ext2 with *data=writeback*), using balanced binary trees for most filesystem metadata, delayed allocation [XFS2000], dynamic disk inode allocation, support for sparse files, space preallocation, and coalescing on deletion. One shortcoming can be the poor file deletion performance, which is constrained by synchronous disk writes[1].

For this paper, we followed the recommendations of the FAQ [XFS] and mounted XFS filesystems using the options *-o logbufs=8,osyncisdsync*; logbufs specifies the number of log buffers kept in memory, which could improve performance[2]; *osyncisdsync* specifies that

---

[1] Asynchronous deletes have been added to more recent versions of XFS (beginning with Linux 2.4.18); however, that version was not used in this paper, since 2.4.18 was not available on all of the systems used in our tests.

[2] We ran the benchmarks on the "large" system (Section 4) with and without *logbufs=8*, with no discernable performance differences.

O_SYNC is treated as O_DSYNC (data-sync only), the default behavior on Ext2 [XFS].

At the present time XFS is not part of the standard Linux kernel; patchsets for recent kernels are available [XFS], and it has been included in recent versions of the Mandrake distribution [Mandrake].

2.5 JFS

IBM®'s JFS [JFS], which originated on AIX®, was then ported to OS/2®, then back to AIX and from there was ported to Linux [vonHagen, p. 106]. It has the advantage that the code has undergone extensive testing under other operating systems. JFS technical features include extent-based storage allocation, variable block sizes (although only 4096 byte blocks are currently supported under Linux), dynamic disk inode allocation, and sparse and dense file support. JFS is a journaling filesystem that supports metadata logging.

JFS is not currently supported on systems where the page size exceeds the filesystem block size, so JFS was not included in our "large"-system benchmarks since the default 16 KB page size on IA64 exceeds the 4 KB JFS block size [Best]. (See Section 4 for description of our "large" system.)

Although JFS is not part of the standard Linux kernel, patchsets for recent kernels are available [JFS], and JFS has been incorporated into recent Mandrake distributions [Mandrake].

3 Benchmark Descriptions

The benchmarks we have chosen for this paper are *pgmeter*, *filemark*, and the *AIM Benchmark Suite VII*.

In a previous paper [Pgmeter], we evaluated many of the other filesystem benchmarks commonly in use on Linux today. These included *Bonnie* [Bonnie], *dbench* [Dbench], and the ever-popular kernel-compile benchmark as well as several others. The advantages of the benchmarks used in this paper over these other benchmarks are that they:

1) Support a wide variety of workloads instead of the specific workload implemented by *Bonnie*.
2) Are not trace driven, (unlike *dbench*) so that they are readily scalable from small to very large systems.
3) Produce significant load on underlying filesystems, unlike the kernel-compile workload that is more often than not CPU bound.

We selected the benchmarks for this paper based in part on our previous work and because we believe these benchmarks represent three interesting and illuminating facets of filesystem performance: I/O throughput, file accesses, and overall system performance; they scale well on larger multiprocessor platforms; and they report accurate, repeatable results.

3.1 Pgmeter

Pgmeter is a file-I/O benchmark that measures the rate at which data can be transferred to/from an existing file according to flexible, synthetic workload description. *Pgmeter* is patterned after the Intel *Iometer* benchmark [Iometer]. In previous work we demonstrated that *pgmeter* creates workloads that have the same performance characteristics as those of *Iometer* [Pgmeter].

The workload descriptions used here are based on those distributed with *Iometer*:

- Sequential read and write tests with fixed record sizes.
- The *Web Server* workload, which is a mixture of 512 byte to 512 KB transfers. All transfers in this workload are read operations; each transfer begins at a randomly selected point in the data file.
- The *File Server* workload, which is a mixture of 512 byte to 64 KB transfers. Of these transfers, 80% are read operations and 20% are write operations. Each operation begins at a randomly selected point in the file.
- An *8K OLTP* workload, defined as 8 KB transfers of which 67% are reads and 33% are writes. Each transfer begins at a randomly selected point in the file. This workload is intended to represent an online transaction processing workload.

Additional details of the Web Server and File Server workloads are provided in the Appendix.

We selected these workloads with a goal of covering a broad range of representative workloads while keeping the total number of tests small. For example, we did not include a purely random, read-only, fixed-record size workload, since the Web Server workload is also a random, read-only workload; we felt that many of the characteristics of the purely random workload would show up in the Web Server tests. Similarly, we believed that the fixed record size, random access benchmark would be covered (in part) by the 8K OLTP workload.

The key statistics reported by *pgmeter* are the filesystem I/O bandwidth, measured in MB/s, and the filesystem operations count, measured in ops/s. In this paper we report only the ops/s statistic; the MB/s results show similar results to those presented here.

Pgmeter includes the following features to increase the accuracy of the test results:

1) Each test begins by unmounting and remounting the target filesystem as a method to flush the file-system buffer cache. Thus at the start of each test, the filesystem buffer cache is empty.
2) A warm-up period can be configured to run at the start of each test. During the warm-up period the full benchmark is run, but statistics collection does not begin until the end of the warm-up period.

The purpose of the warm-up period is to exclude measurement of transient startup effects. An example of this would be the abnormally high write-rate that occurs in a sequential write test until the filesystem buffer cache fills up and the filesystem has to actually start writing data to disk.

3.2 Filemark

Filemark is our version of the *Postmark* benchmark [PostMark]. *Postmark* is a filesystem-operation–intensive benchmark. *Postmark* execution consists of three phases:

1) Creation Phase. During this phase, *postmark* creates a specified number of files (See Section 5, "Benchmark Run Rules" for details). These files are randomly distributed among a number of subdirectories of the target directory.
2) Transaction Phase. During this phase, *postmark* executes a number of "transactions" against the set of files created in the Creation Phase. Each transaction consists of the following two steps: (i) Choose a file at random, then according to a second random choice, either read the entire file or append to the file. (ii) According to a random choice either create a new file or delete an existing file.
3) Deletion Phase. During this phase, the files remaining in the file set are deleted.

Our version of *postmark*, which we call *filemark*, includes the following six enhancements to *postmark 1.5:*

1) *Filemark* is multithreaded whereas *postmark 1.5* is single-threaded.[3] This allows us to scale up the *filemark* workload to provide a heavier and more realistic load on the server machines we are testing.

2) *Filemark* uses higher precision timing services than does *postmark*. *Filemark* uses *gettimeofday()* instead of *time()*. The former is nominally accurate to the nearest microsecond while the latter is accurate only to the nearest second.

3) *Filemark* supports repeated *Transaction Phases* following a single *Creation Phase*. This was incorporated to significantly reduce the amount of time required to get multiple repeated trials suitable for confidence interval generation. Although each transaction phase uses basically the same set of created files, the transaction rate was assumed not to depend that much on the particular set of files created. Each transaction phase begins with its own warm-up period and ends by unmounting and remounting the filesystem containing the target directory. In this way, each transaction phase is as much of an independent trial as possible, and standard techniques for confidence interval calculation should apply.

4) *Postmark* only allows the *bias read* and *bias create* probabilities (these are the probability of choosing read over append, or create over delete, in steps (2)(i) and (2)(ii) above) to be specified to the nearest 10 percent; *filemark* allows these probabilities to be specified to the nearest 1 percent.

5) Code has been added to *filemark* to allow the user to request that the Deletion Phase not be run at all. This was included to speed the execution time of the entire test. Our experience is that the file Deletion Rate as reported by *filemark* (or by *postmark*) has such high variation as to be almost meaningless. Also, for our run rules (see Section 5), we always rebuilt (reformatted) the filesystem after a *filemark* run, so we did not need to run the Deletion Phase.

3.3 AIM7

The *AIM Benchmark Suite VII*, referred to here as *AIM7*, has been widely used for more than a decade by many Unix computer system vendors. AIM Technology, Inc. originally developed and licensed the AIM Benchmark suites. Caldera International, Inc., has acquired the AIM Benchmark license and has recently released the sources for *Suite VII* and *Suite IX* under the GPL [AIM7].

AIM7 is a C-language program that forks multiple processes (called *tasks* in *AIM7)*, each of which concurrently executes a common, randomly-ordered set of subtests called *jobs*. Each of the 53 kinds of jobs exercises a particular facet of system functionality, such as disk-file operations, process creation, user virtual memory operations, pipe I/O, and compute-bound

[3] While writing this paper, we became aware of a multithreaded version of *postmark*, version 1.99. [Katcher]. Due to time constraints we have continued to use *filemark* in this paper rather than convert to this new version of *postmark*.

arithmetic loops. *AIM7* includes disk subtests for sequential reads, sequential writes, random reads, random writes, and random mixed reads and writes.

An AIM7 run consists of a series of subruns with the number of tasks, *N*, being increased after the end of each subrun. Each subrun continues until each task completes the common set of jobs. The performance metric, "Jobs completed per minute", is reported for each subrun. The result of the entire *AIM7* run is a table showing the performance metric versus the number of tasks, *N*.

Typically, as *N* increases, the jobs per minute metric increases to a peak value as CPU idle time gets used for real work and as economies of scale continue to succeed; thereafter it declines due to software and hardware bottlenecks. This peak throughput value provides the primary metric of interest. We report the peak throughput achieved with each filesystem tested as our *AIM7* statistic for that filesystem.

3.4 Benchmark Summary

To reiterate, *pgmeter* measures the rate at which data can be transferred to or from an existing file. It is intended to model the behavior of large applications as they read and update their associated disk files. *Filemark* is intended to model the kind of operations that a file server or mail server might execute against a filesystem. By choosing these two benchmarks, we have thus chosen opposite ends of the spectrum of possible filesystem workloads. Whereas *pgmeter* measures the filesystem's ability to transfer data under a wide variety of workloads, *filemark* measures the filesystem's ability to create and update a number of small files. We feel that real-user applications lie somewhere between these two extremes and that our benchmarks should therefore encompass the actual workloads of many real-user programs.

AIM7 serves as a multifaceted system-level benchmark that exercises more than just filesystem activity. We make no claim that an *AIM7* workload represents a "real world" job mix. Rather, we claim that *AIM7* imposes a workload that shows how a particular Unix system scales under the stress of an ever-increasing load. We believe that since the *AIM7* workload includes a mix of both CPU-bound and I/O-bound jobs, that *AIM7* provides an estimate of overall system throughput that is not provided by *pgmeter* or *filemark*.

4 Experimental Testbeds

In this paper we have executed our benchmarks using three systems:

1) The *small* system. This is a single-processor 1.7 GHZ Pentium 4 with 512MB of memory and an 80 GB IDE disk. For the experiments of this paper, this machine was booted with either 128MB or 512MB of RAM.
2) The *medium* system. This is a 4-CPU 700 MHZ Pentium® III Xeon® system with 5 SCSI disks, 8 GB each. For the experiments of this paper, this system was booted with 900 MB of RAM.
3) The *large* system. This is an SGI prototype of a ccNUMA machine using the Intel IPF processor. Currently based on the Itanium and a system interconnect similar to that of the SGI Origin® 3000 [SGIorigin], this machine allows us to run a modified Linux 2.4.17 kernel with up to 28 processors, 16 GB of main memory, and 10 Fibre Channel controllers attached to a total of 120 disk drives.

The above set of machines provides a sampling from a wide variety of machines used to run Linux today. Although it may still be that Linux is most commonly run on uniprocessor machines, the 4-CPU system represents a "sweet spot" for Linux mid-range servers and thus is an important system to include. Similarly, while the 28-processor SGI machine reflects a class of system that is rarely found in the Linux world today, we believe that such systems will be more commonplace in the next few years. Moreover, benchmarking on such machines tends to exaggerate and thus more clearly expose performance problems that are less apparent on smaller hardware configurations.

5 Benchmark Run Rules

Filesystem benchmarking requires careful setup [Tang]. An issue one must often contend with is how to defeat the effects of the filesystem buffer cache. Without careful experimental design, all of the filesystem requests could be satisfied in the cache and no disk activity would occur. A common way to avoid this problem is to use a total file size that exceeds the amount of main memory available on the system.

Another approach is to use a file-access mode that bypasses the filesystem buffer cache, such as O_DIRECT. We chose to not use O_DIRECT for this paper in order to focus on the default filesystem behavior that most users will encounter.

The second issue that one must address is estimating the accuracy of the results of the test. In our experience, filesystem benchmarks are notorious for being

nonrepeatable, bimodal, and full of hysteresis effects, making it a challenge to get consistent results.

In this paper, we estimate the accuracy of the tests using 95% confidence intervals calculated using a Student's-T statistic. Such statistics usually require 10 or more independent trials to get a reasonably tight confidence interval. However, the large number of filesystems examined in this paper, the number of tests per filesystem and the running times of the experiments meant that running the test multiple times was not feasible. Our solution was to use intermediate test results to create quasi-independent observations of the test statistics, and then to generate confidence intervals based on these observations.

Finally, one must be aware of internal bottlenecks within the hardware I/O subsystem that can otherwise skew results. For example, we run all of our benchmarks (except for the ones on the small system) using multiple physical disks. We do this because this improves parallelism and allows higher I/O rates to be achieved; this results in a higher load being placed on the filesystem that we are testing. Additionally, this is sometimes necessary to avoid excessive disk-head movement that would otherwise invalidate the test. For example, if multiple sequential read tests are executed at the same time on a single disk drive, the head movement pattern may be nearly indistinguishable from that of a random access workload, and the resulting transfer rates will be much slower than otherwise might be achieved.

5.1 Pgmeter

For the *pgmeter* tests, the total size of the test file(s) used was always chosen to be significantly larger than the main memory of the system. For the sequential read and write tests, this implies that each user-level I/O request results in a disk request once the filesystem buffer cache is filled (except for the effect of read ahead in the Linux kernel). We used a warm-up period of 30 seconds. (This was based on examining the *pgmeter* output when no warm-up period was specified and making a judgment about how long it took to get the system to steady-state behavior.) We used an experiment runtime of 5 minutes for each *pgmeter* test case.

For the random access tests, the effect of the filesystem cache is that certain requests will be satisfied out of cache. At the start of the test, the cache is empty (since we remount the target filesystems at the start of the test), and as the test progresses more and more of the target file is read into memory. Thus, the effect of the filesystem cache in the random access tests is to cause the filesystem data transfer rate to appear to increase as the test continues to run. To minimize this effect, we chose a file size large enough that only a small fraction of the file is read into memory during the test. Examination of the *pgmeter* output also shows that the data rate in each 10-second interval of the test was very nearly constant in all of the random access tests conducted for this paper.

On the small system, we ran *pgmeter* against a single 1 GB file, with the sequential read and write cases using an I/O size of 8 KB. We ran trials with 1, 4, 16, and 64 outstanding I/O requests against this file. Main memory on the system was 512MB.

On the medium system, we ran *pgmeter* using 4 files, each 2 GB in size, with each file being on a distinct physical disk. The I/O size for the sequential read and write tests was 64 KB. We ran trials with 1, 4, 16, and 64 outstanding I/O requests against each of the files. Main memory on the system was 900MB.

On the large system, we ran *pgmeter* using 28 files, each 2 GB in size, with each file being on a distinct physical disk. (We also report some results using 112 files.) The I/O size for the sequential read and write tests was 64 KB. We ran trials with 1, 2, 4, and 8 outstanding I/O requests against each of the 28 files. These numbers were reduced from the small and medium cases due to the large (total) number of processes that would otherwise be created in this case. The main memory size was 16 GB.

We calculated confidence intervals for *pgmeter* as follows. Interval statistics were recorded by *pgmeter* every 10 seconds. That is, we calculated the MB/s and ops/s for every 10 seconds of the run. The statistic reported by the benchmark itself is the average of all of the interval statistics over the run. We used the interval statistics to calculate an estimate of the mean and 95% confidence intervals for the mean.

5.2 Filemark

We ran *filemark* for 1, 8, 64, and 128 threads for each of the five filesystems available for testing on IA32. The filesystems were recreated using *mkfs* at the start of each test for each number of threads. The system was not rebooted either at the start of a new filesystem test nor at the start of a new number of threads test. As previously discussed, only one Creation Phase was run per case and then the repeated Transaction Phases were run without running another Creation Phase. No Deletion Phase was run. Instead, the filesystem was rebuilt after each *filemark* run.

After the benchmark runs completed, the first transaction was discarded and we used the remaining transac-

tion rates to calculate the mean transaction rate and a 95% confidence interval for the mean. Our experience was that the first trial was often an outlier, due to startup effects. Additional trials were conducted for any test where the confidence interval half-width was larger than 10% of the mean.

It was only through this approach that we are able to obtain meaningful statistics from the *filemark* benchmark. Random variations between tests were otherwise so high as to make the measured statistics meaningless. Repeating the entire series of tests enough times to generate confidence intervals was not feasible given the extended run times of each series of tests (16–18 hours for a complete run over all file systems).

Of course, this approach did not allow us to calculate confidence intervals for the Creation Rate; only by using repeated trials could we provide that data. However, we regard the most important statistic reported by *filemark* to be the Transaction Rate, since this represents the rate at which a file or mail server would be able to process requests. This is the statistic reported in this paper and we do not include Creation Rate data.

Finally, it was necessary to make the file creation and deletion probabilities equal in order to keep the number of files nearly constant. (In *filemark*, this is done by setting *bias create* to 50.) Without this change, the results of each trial were not taken from a stationary distribution and the confidence interval calculation was not statistically valid.

In all of our *filemark* runs, *bias read* was 75 (this is the probability expressed as a percent of reading versus appending to a file in part (i) of a *filemark* transaction) and *bias create* was 50 (this is the probability of creating versus deleting a file in part (ii) of a *filemark* transaction).

For the small system, we ran *filemark* with the following parameters: 1 target directory with 10 subdirectories, 10,000 total files, 30-second warm-up per transaction phase, 2-minute transaction phases, 12 transaction phases per run. The number of subdirectories was chosen, in part, from input received from the ReiserFS mailing list [ReiserFS]. Because the file set here is relatively small, the system was booted with only 128MB of memory (instead of 512 MB).

For the medium system, we ran *filemark* with 4 target directories, each on a different physical disk, 2000 subdirectories per target directory, 100,000 total files, 30-second warm-up per transaction phase, 2-minute transaction phases, and 6, 12, or 24 transaction phases per run, depending on what was required to get the confidence interval half-width to be smaller than 10% of the mean. As before, we discarded the first trial of the series before the confidence intervals were calculated.

In addition to the 10,000 and 100,000 file cases, we tested two different file-size distributions with *filemark*. In the first case, which we will refer to as the *small-file workload*, file sizes were chosen uniformly between from 512 bytes to 9.77 KB (this is the *postmark* default) and the read and write I/O sizes were 512 bytes. In what we will refer to as the *large-file workload*, file sizes ranged from 4 KB to 16 KB, and the read and write I/O sizes were 4 KB. Since the block size for all of the filesystems used in this paper is also 4 KB, we designed the large-file case to avoid filesystem read-before-write overheads.

The total size of the file set for the small-file workload is roughly 50 MB for the 10,000-file case and 500 MB for the 100,000-file case. For the big-file workload the file set has size 100 MB in the 10,000-file case and approximately 1 GB in the 100,000-file case. The large-file case is thus clearly larger than the main memory in the small system 128 MB case and the medium system's 900 MB case, while for the small-file case the entire file set could fit into the filesystem buffer cache. This was one reason for running both the small and large-file cases. The other reason was to examine the effect of the request size change from 512 bytes to 4 KB.[4]

We did not run *filemark* for the large system because it is a small-server benchmark. Even for the medium system, the entire file-set consists of only 500 MB of files. The large system has 16 GB of RAM. We have thus far been unable to scale up *filemark* sufficiently that it would reliably generate more than 16 GB of files.

5.3 AIM7

Although *AIM7* can be executed against a single disk drive, our experience is that the filesystem jobs are disk-transaction-rate constrained. With too few disk drives available, the *AIM7* test will be almost completely I/O bound. The small and medium systems used for our tests were thus too small to run a meaningful *AIM7* test. For our large system we have determined that 120 disk drives is more than sufficient to produce a compute-bound *AIM7* test.

[4] We also ran the same benchmark on the small system with larger memory (512 MB instead of 128 MB) and on the medium system with smaller memory (512 MB instead of 900 MB) and found no significant change in the results from those reported in Section 6.

For our *AIM7* benchmark runs, we chose the *AIM7 workfile.shared* workload. *AIM7* documentation [AIM7] describes this job mix as a "Multiuser Shared System Mix".

We repeated the benchmark runs on the 28 CPU large system with the system booted with 2, 4, 8, 16, 24 and 28 processors. At each CPU count we executed the *AIM7* benchmark against each of the four filesystem types (JFS is not currently available on IA64); we recorded the maximum jobs completed per minute statistic. We rebooted the system before each test run. Each of the 120 disk drives used was formatted with a single filesystem partition.

We executed only a single trial of each *AIM7* benchmark for each filesystem and CPU configuration. This is due to the length of each *AIM7* run, requiring one to ten hours to reach a peak throughput. A single try should be sufficient because each *AIM7* run actually consists of a number of subruns (one for each workload level), and examination of the graph of "jobs completed/min" versus the number of simultaneous runs would allows us to detect and discard anomalous results.

6 Results

We concentrate here on where the benchmarks show significant differences between the filesystems. A significant difference between filesystems is assumed to exist if the 95%-confidence intervals for the two filesystems do not overlap. Additionally, we typically require a difference of at least 10% in the mean values for a difference to be considered significant.

6.1 Small System

On the small system, the filesystems all produced about the same results on all of the *pgmeter* tests (the 95% confidence intervals for the results overlapped). Comparison of the *pgmeter* statistics for the sequential read tests with those of *iostat* shows that each of the filesystems drives the disk at nearly its maximum rate of around 2200 I/O operations/s. (*hdparm –t –T* shows that the disk can maintain 17.4 MB/s; this would correspond to around 2227 I/O operations/s for an 8 KB request size). We conjecture that these *pgmeter* tests are all disk-I/O limited and thus produce equivalent results for all filesystems.

The *filemark* results for the small system do show significant differences among the tested filesystems. Figure 1 shows the results for the small file workload. In this figure, ReiserFS, Ext2, and Ext3 are in the upper group and XFS and JFS are in the lower group. ReiserFS scales well with increasing number of threads, and it provides significantly better performance than Ext2 and Ext3. Neither XFS nor JFS improve very much as the number of threads increases, and XFS and JFS provide performance at basically the same level. These results are consistent with the *Bonnie* tests for random seeks as reported by von Hagen [vonHagen, p. 240].

We surmise that the poor performance of XFS and JFS in this environment is due to the large-system heritages of these filesystems; optimizations appropriate for a larger system may not be beneficial in a small-system environment.

Figure 2 shows the same experiment with the big-file workload. In this test, Ext2 and Ext3 begin with higher transaction rates for one thread; as the number of threads grows, ReiserFS performance increases until it nearly reaches the rates of Ext2 and Ext3 with *data=ordered*. Ext3 with *data=writeback* is significantly slower than the other filesystems at the larger number of threads. These results are consistent with those available on the ReiserFS web site [ReiserFS], where ReiserFS performance is often the best for smaller file sizes.

XFS and JFS still perform the slowest; but in this case there is a clearer distinction between XFS and JFS.

The trends exposed by this set of experiments on the small system appear to be the following:

1) For *file-I/O intensive workloads* on the small system, all of the filesystems provide equivalent performance.
2) For small files, where the I/O request size is less than the filesystem block size, and using filesystem-operation intensive workloads, ReiserFS may be fastest.
3) For somewhat larger files, where the I/O request size is equal to the filesystem block-size, and using filesystem-operation intensive workloads, Ext2 or Ext3 with *data=ordered* may be fastest.

6.2 Medium System

When we looked at the medium system results, we did find differences among the filesystems when running the *pgmeter* benchmark. Figure 3 shows the results of the *pgmeter Sequential Read 64 KB* workload on the medium system. This graph shows a clear performance advantage for JFS and XFS over the other filesystems, particularly as the number of outstanding I/O's per file increases. It appears that the large-system heritage of these filesystems starts to pay off on the medium system.

This trend did not carry through for the other *pgmeter* workloads, however, where the results for all filesys-

tems were equivalent. In particular, all of the filesystems provided nearly identical performance for the *Web Server* case, indicating that for a read-only workload with variable request sizes, all of the filesystems are equivalent. Figure 4 shows the results of *pgmeter File Server* workload and it indicates that for a mixed read-write workload with a variable request size, XFS and ReiserFS provide marginally less performance than the other filesystems. Figure 5 shows that XFS moves back into the upper group for a mixed read/write workload with constant 8 KB record sizes.

Figure 6 shows the results of the *filemark* benchmark for the medium system and the small-file workload. If we compare this figure to Figure 1 we see that in this case XFS has moved into the upper group. XFS, Ext2, and ReiserFS provide equivalent levels of performance. Examination of the 95% confidence intervals shows that the ReiserFS 64 thread and 128 thread results in this figure are statistically indistinguishable. We therefore do not regard the peak that occurs for ReiserFS at 64 threads in Figure 6 to be significant. Both versions of Ext3 are in the second group for 64 and 128 threads and are equivalent to the top three filesystems for the 1 and 8 thread cases. JFS continues to trail the performance of the other filesystems for this benchmark.

Figure 7 shows the results of the *filemark* benchmark for the medium system and the large-file workload. This graph looks basically the same as the small-file case shown in Figure 6. The corresponding comparison for the small system (see Figures 1 and 2) found significant differences between the small-file and large-file workloads. We have not determined precisely why these two comparisons are different. However, for the small system, large-file workload case, the entire file set fits into memory (100 MB of files and 128 MB of memory). In the medium system, large-file workload case, the entire file set is too large to fit into memory (1 GB of files and 900 MB of memory). This may explain why the differences between Figures 1 and 2 are not also seen between Figures 6 and 7.

The trends exposed by this set of experiments are the following:

1) XFS and JFS appear to perform better (in relation to the other filesystems tested) on sequential workloads on multiple processor systems than they did on uniprocessor systems.
2) Ext3 appears to have a scaling bottleneck not present in Ext2 (possibly journal related) that keeps its throughput from increasing as the number of threads increase under the *filemark* benchmark.
3) In spite of good performance on the sequential read benchmark of *pgmeter*, JFS did not scale well with either multiple processors or increasing numbers of threads under the *filemark* benchmark.

6.3 Large System

The differences between filesystems are clear when we examine the output for *pgmeter* on the large system. Figures 8, 9, and 10 show results of the *pgmeter* runs for *Sequential Read 64KB*, *Sequential Write 64KB*, and *File Server* respectively. (Recall that JFS was not available for the large system.) Figures 8 and 9 show that XFS provides significantly better performance than the other filesystems for this constant-request-size and regular workload. To continue the previous discussion, we can now observe that this is where the large-system heritage of XFS is truly beneficial.

On the other hand, Figure 6 shows that for a random request size and 100% random access, Ext3 with *data=writeback*, and ReiserFS are the best performing filesystems. Ext3 with *data=ordered* is next, and XFS is last. This suggests that for random access and variable request size workloads, one of these other filesystems might be superior to XFS.

Figure 11 shows the peak jobs-per-minute throughput curves for the *AIM7* benchmark for each tested filesystem type at each CPU count.

Note that the jobs-per-minute y-axis values in Figure 11 are expressed as numbers for each filesystem type relative to the baseline 2-processor-Ext2 value. This is because the large system is a prototype system, and SGI management requested that we not publish absolute *AIM7* performance numbers for this system.

The *AIM7* benchmark indicates that the four filesystems separate into two performance groupings, with Ext2 and XFS performing roughly equivalently, and Ext3 and ReiserFS performing at significantly lower levels. Ext2 and XFS are essentially equivalent performers up to 16 processors. Above 16 processors XFS achieves marginally superior scaling, and at 28 processors XFS reaches a peak throughput about 14% higher than Ext2's peak. Ext3 and ReiserFS perform at about 65% of the level of Ext2 at 2 processors, and neither of these two filesystem types gains much improvement in peak throughput at higher processor counts. Both Ext3 and ReiserFS see a small decrease in peak throughput above 8 processors. A similar small decrease is seen with Ext2 above 16 processors, although overall Ext2 peak throughput is substantially higher than Ext3 and ReiserFS.

We interpret these results as indicating that both Ext3 and ReiserFS are constrained by bottlenecks that are significantly more severe than those that affect Ext2

and XFS. Past experience with this ccNUMA platform leads us to believe that the bottlenecks are primarily highly contended kernel spinlocks.

By adding Lockmeter [Lockmeter] to our 2.4.17 kernel, we can measure spinlock contention. Table 1 shows the percent of all available CPU cycles that were consumed by busy waiting for three spinlocks while running the *AIM7* benchmark with the indicated filesystem. (The data is for the 500 *AIM7* processes case on our large system.) Both Ext3 and ReiserFS are dominated by the so-called *Big Kernel Lock* (BKL), with 85% to 88% of all CPU cycles being consumed by busy waiting for that spinlock. The BKL is still dominant for Ext2, although the runqueue_lock (protecting the single global CPU runqueue) now emerges as being almost as significant. For XFS the BKL is even less important, and the runqueue_lock is the primary scaling bottleneck.

Filesystem	BKL	runqueue_lock	pagecache_lock
Ext2	30%	25%	5%
ReiserFS	88%	3%	< 1%
Ext3	85%	5%	< 1%
XFS	12%	35%	10%

Table 1. Percent of Total CPU Cycles in Spin Wait for Locks during AIM7 runs on the Large System

The effects of spinlock contention are nonlinear and sometimes surprising. For example, one might imagine that a multiqueue CPU scheduler patch (e.g., the IBM-contributed patch [IBM-MQ]) would eliminate runqueue_lock contention and thus gain back those cycles consumed by spinlock waits. For XFS, where the predominant *AIM7* bottleneck in the baseline 2.4.17 kernel is the runqueue_lock, when we apply the IBM Multiqueue Scheduler patch, the *AIM7* peak at 28 processors increases to 40% (up from 14%) above the unpatched Ext2 performance. However, for Ext2 with its 30% wasted CPU cycles waiting on the BKL and 25% waiting on the runqueue_lock, an unfortunate side-effect of the IBM Multiqueue scheduler is that contention on the BKL increases nonlinearly to a 70% waste of CPU cycles, resulting in a 30% drop in the *AIM7* peak throughput. The lesson here is that the highest contending spinlocks should be attacked first.

This spinlock analysis leads us to conjecture that XFS might perform better than the other filesystems when CPU utilization is high. For example, for the *pgmeter* sequential read and write tests, CPU utilization was nearly 100% for all filesystems except XFS. Our experience indicates that this is likely due to spinlock contention for those other filesystems. For the cases where CPU utilization is lower, then XFS does not perform as well as the other filesystems (e.g., the File System workload in Figure 10). This conjecture suggests that XFS should perform better on the mixed workload cases if we could increase the load to the point where CPU utilization becomes a limiting factor for the other filesystems.

Figure 12 shows the result of a 112-file test case on our large system. This represents a four-fold increase in system load over the previous 28-file case. For this case, CPU utilization is very high for all of the file systems except XFS, and XFS performs better than the other filesystems.

The trends exposed by the experiments on the large system are the following:

1) At the highest loads, all of the filesystems start to become CPU bound. XFS suffers less from this problem than the other filesystems; hence it is able to deliver better service at these higher loads. A significant part of the CPU load for the other filesystems is due to spinlock contention that is higher than that found in XFS.
2) In cases where the system is not quite so busy, filesystems other than XFS can provide better performance than XFS does.
3) Ext2 and XFS scale the best as the number of processors increase (as measured by the *AIM7* workload). XFS is the only journaling filesystem able to provide the same level of scaling as Ext2.
4) Ext3 appears to have internal scaling bottlenecks associated with the Ext3 specific code since Ext2 scales well on *AIM7*, but Ext3 does not.
5) ReiserFS has similar scaling problems to those of Ext3.

7 Conclusions

We stated specific conclusions for each of the small, medium and large filesystems in Section 6. In this section we attempt to provide what appear to be performance trends that are true over the entire set of experiments:

1) In general, we found the performance of Ext2 to be better than that of Ext3. In most cases, we found the performance of Ext3 with *data=writeback* to be the same as with *data=ordered*. In certain cases, we found Ext3 with *data=ordered* to be faster; in other cases *data=writeback* is faster. On the large system, however, we found performance of *data=writeback* to be even better than that of Ext2

(See Figure 10). It is therefore not clear to us how Ext3 users might be able to predict which journaling mode should be used for their workload.

2) For small files (5 KB average size or less) and file-system-operation–intensive workloads, ReiserFS can often provide the best performance.
3) For File-I/O intensive workloads like *pgmeter*, all of the filesystems are equivalent on systems like our small system.
4) When performance is of primary importance, neither XFS nor JFS are good choices for systems like our small system and filesystem operation-intensive workloads like *filemark*.
5) On small SMP systems like our medium system, XFS and JFS are good choices for sequential read workloads.
6) JFS appears to have difficulty scaling with SMP or the number of concurrent threads in a filesystem operation-intensive workload like *filemark*.
7) XFS currently performs best on systems like our large system under heavy I/O workloads.

8 Future Work

In this paper we have observed behavior without providing a detailed analysis of why the various filesystems perform as they do. This is ongoing work and we expect to provide such insight in future papers on filesystem performance, as well as large-system performance and scalability under the Linux operating system. However, this is a difficult problem, with sometimes surprising results [Robbins].

Finally, nothing in the Linux kernel stays the same for long. Over time, studies such as this one will need to be repeated in order to update the results to new versions of Linux and its filesystems.

9 Software Availability

The *pgmeter* and *filemark* benchmarks are available under the GPL and the Artistic License, respectively at the SourceForge sites *pgmeter.sf.net* and *filemark.sf.net*. *AIM7* is available at Caldera's Web site [AIM7].

10 Acknowledgements

The authors would like to acknowledge the support and assistance of the management at SGI and IBM in the creation of this paper. We gratefully acknowledge the help and assistance of the anonymous referees, as well as the assistance of the shepherd for this paper, Erez Zadok of Stony Brook University. We additionally acknowledge the contributions that VA Linux, Inc. has made through the SourceForge project. Roger Sunshine wrote much of the code for *pgmeter* and we continue to greatly appreciate his contributions. This work represents the view of the authors and does not necessarily represent the view of IBM or SGI.

11 References

[AIM7] http://www.caldera.com/developers/community/contrib/aim.html

[Best] Steve Best, personal communication, March 4, 2002

[Bonnie] http://www.textuality.com/bonnie

[Bovet] Daniel P. Bovet and Marco Cesati, *Understanding the Linux Kernel*, O'Reilly and Associates (2001), ISBN 0-596-00002-2.

[Dbench] http://www.samba.org

[Ext2] Rémy Card, Theodore Ts'o, and Stephen Tweedie, "Design and Implementation of the Second Extended Filesystem," *Proceedings of the First Dutch International Symposium on Linux*, 1994, ISBN 90-367-0385-9, also available at http://web.mit.edu/tytso/www/linux/ext2intro.html.

[Ext3RH] http://www.redhat.com/support/wpapers/redhat/ext3/index.html

[Galli] Ricardo Galli, "Journal File Systems in Linux," *Upgrade*, Vol. II, No. 6 (December 2001), pp. 50-56, http://www.upgrade-cepis.org/issues/2001/6/up2-6Galli.pdf

[IBM-MQ] http://lse.sourceforge.net/scheduling

[Iometer] http://developer.intel.com/design/servers/devtools/iometer/index.htm

[JFS] http://oss.software.ibm.com/jfs

[Katcher] Jeffrey Katcher, personal communication, February 26, 2002.

[Lockmeter] http://oss.sgi.com/projects/lockmeter

[Mandrake] http://www.mandrake.com/en/features.php3

[Pgmeter] Ray Bryant, Dave Raddatz, and Roger Sunshine, "PenguinoMeter: A New File-I/O Benchmark for Linux," *Proceedings of the 5th Annual Linux Showcase and Conference*, Nov 8-10, 2001. http://www.linuxshowcase.org/tech.html.

[PostMark] Jeffrey Katcher, "PostMark a New Filesystem Benchmark," Network Appliance Technical Report TR3022, http://www.netapp.com/tech_library/3022.html.

[ReiserFS] http://www.reiserfs.org

[Robbins] Daniel Robbins, "Common threads: Advanced filesystem implementer's guide, Part 8," IBM Developer Works, http://www-106.ibm.com/developerworks/linux/ library/l-fs8.html?dwzone=linux.

[SGIorigin] http://www.sgi.com/origin/3000

[Tang] Diane Tang and Margo Seltzer, "Lies, Damned Lies, and File System Benchmarks," in VINO: The 1994 Fall Harvest, Harvard Division of Applied Sciences Technical Report tr-34-94, 1994, ftp://ftp.deas.harvard.edu/techreports/tr-34-94.ps.gz.

[Ts'o] Theodore Ts'o, personal communication, March 8, 2002.

[Tweedie] Stephen Tweedie, "Ext3, Journaling Filesystem," Ottawa Linux Symposium 2000, html version available at OLS Transcription Project, http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html.

[vonHagen] William von Hagen, *Linux Filesystems*, Sams Publishing, 2002. ISBN 0-672-32272-2.

[XFS] http://oss.sgi.com/projects/xfs

[XFSirix] Adam Sweeney, Doug Doucette, *et al*, "Scalability in the XFS File System," *Proceedings of the 1996 Usenix Annual Technical Conference*, San Diego, Cal., (January 22-26, 1996), http://oss.sgi.com/projects/xfs/papers/xfs_usenix

[XFS2000] Jim Mostel, *et al*, "Porting the SGI XFS File System to Linux," *Proceedings of the 2000 Usenix Annual Technical Conference*, San Diego, Cal, (June 18-23, 2000), http://www.usenix.org/publications/library/proceedings/usenix2000/freenix/mostek.html .

Trademarks

Appendix

Filesystem Versions Used

The version of Ext3 used in this paper is the default version available in Linux Kernel 2.4.17. The Ext3 used tools used are from e2fsprogs-1.25 and util-linux-2.11g.

The version of ReiserFS used in this paper is 3.6.25; this is the default version available in Linux kernel 2.4.17. The version of ReiserFS tools used is 3.x.0k_pre11.

The version of XFS used here on our small and medium systems is the 2.4.17 XFS patch available from SGI [XFS] (xfs-2.4.17-all.i386.bz2). A comparable version, developed at SGI for our ccNUMA prototype, is used on our large system.

The version of JFS used here is 1.0.15 for Linux Kernel 2.4.17. The patch files are: jfs-2.4.common-1.0.15-patch, jfs-2.4.17-1.0.15-patch, and jfsutils-1.0.15.tar.gz, all taken from [JFS].

Web Server Workload

The Web Server workload (as distributed with *Iometer* [Iometer]) is defined by the following parameters:

Request size in bytes	Percent requests this size	Percent read (vs. write)	Percent random (vs. sequential)
512	22	100	100
1024	15	100	100
2048	8	100	100
4096	23	100	100
8192	15	100	100
16384	2	100	100
32768	6	100	100
65536	7	100	100
131072	1	100	100
524288	1	100	100

File Server Workload

The File Server workload (as distributed with *Iometer* [Iometer]) is defined by the following parameters:

Request size in bytes	Percent requests this size	Percent read (vs. write)	Percent random (vs. sequential)
512	10	80	100
1024	5	80	100
2048	5	80	100
4096	60	80	100
8192	2	80	100
16384	4	80	100
32768	4	80	100
65536	10	80	100

Figure 1: Filemark, Small System, Small-File Workload

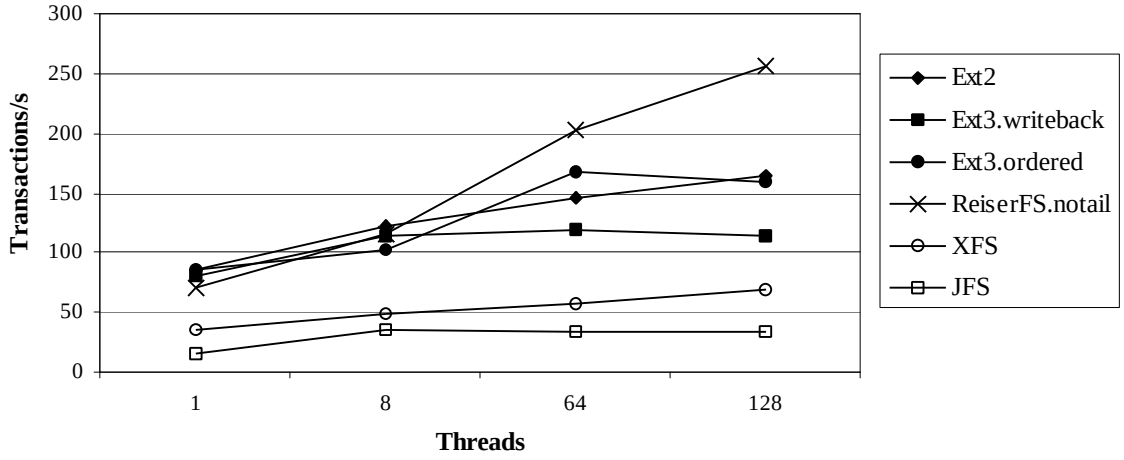

Figure 2: Filemark, Small System, Large-File Workload

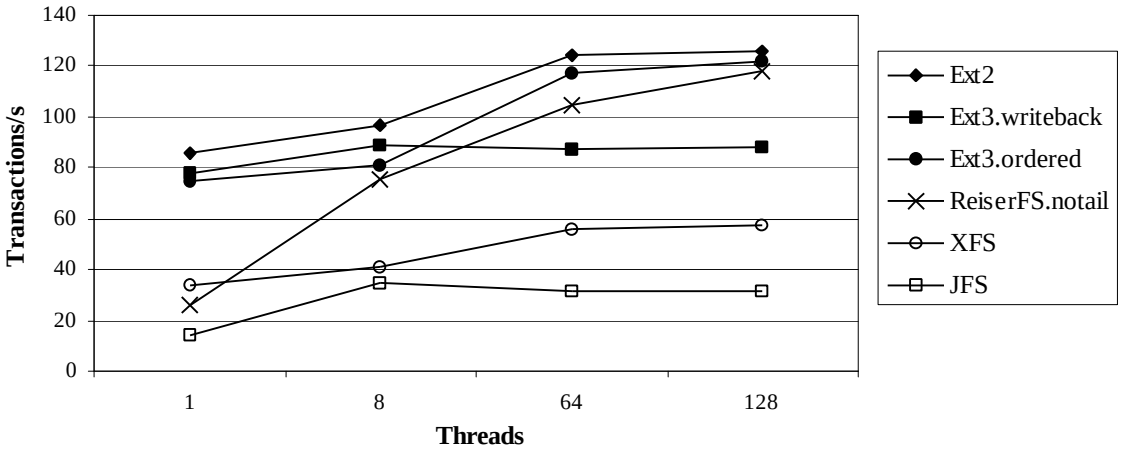

Figure 3: Pgmeter, Medium System, Sequential Read 64 KB Workload

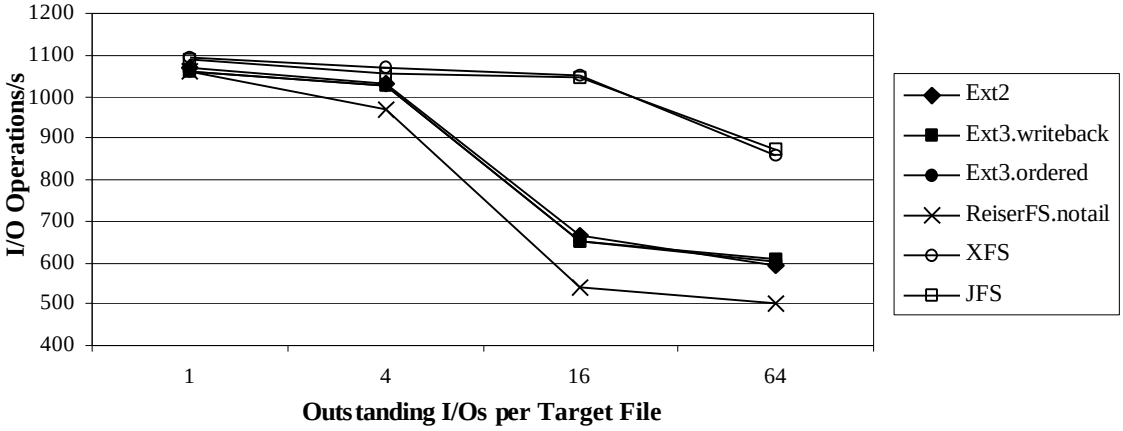

Figure 4: Pgmeter, Medium System, FileServer Workload

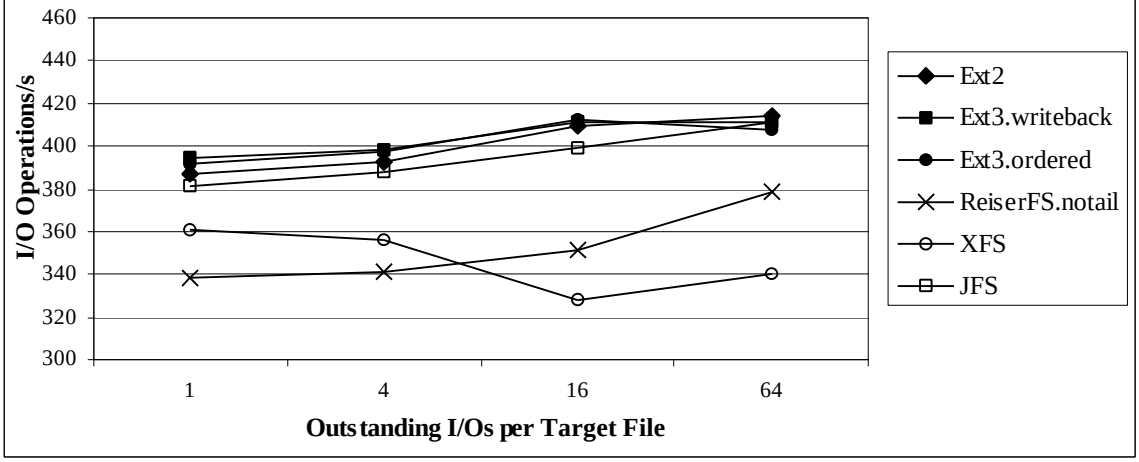

Figure 5: Pgmeter, Medium System, 8K OLTP Workload

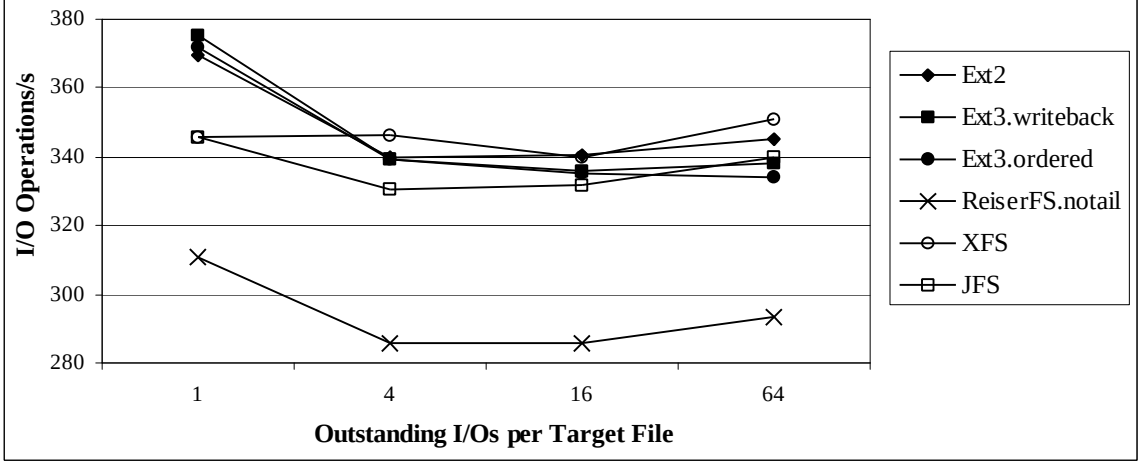

Figure 6: Filemark, Medium System, Small-File Workload

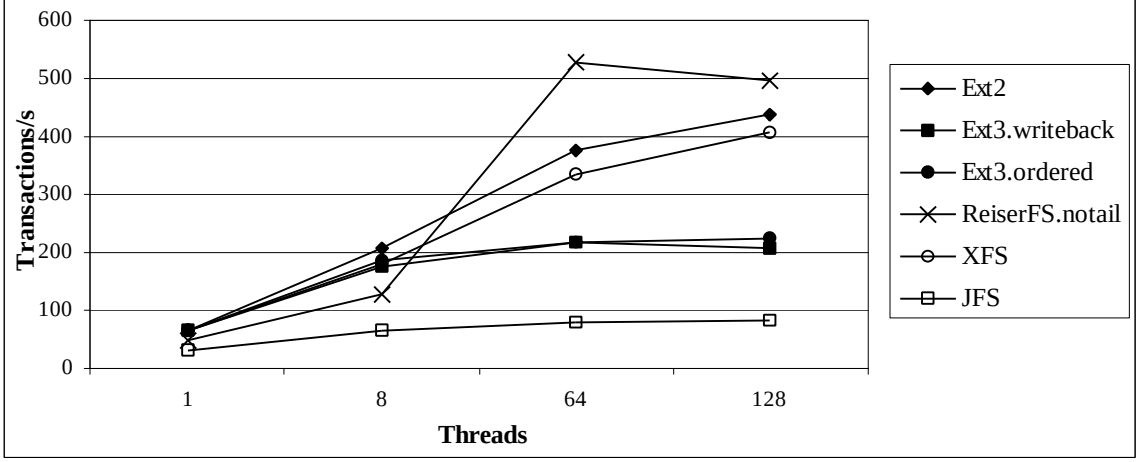

Figure 7: Filemark, Medium System, Large-File Workload

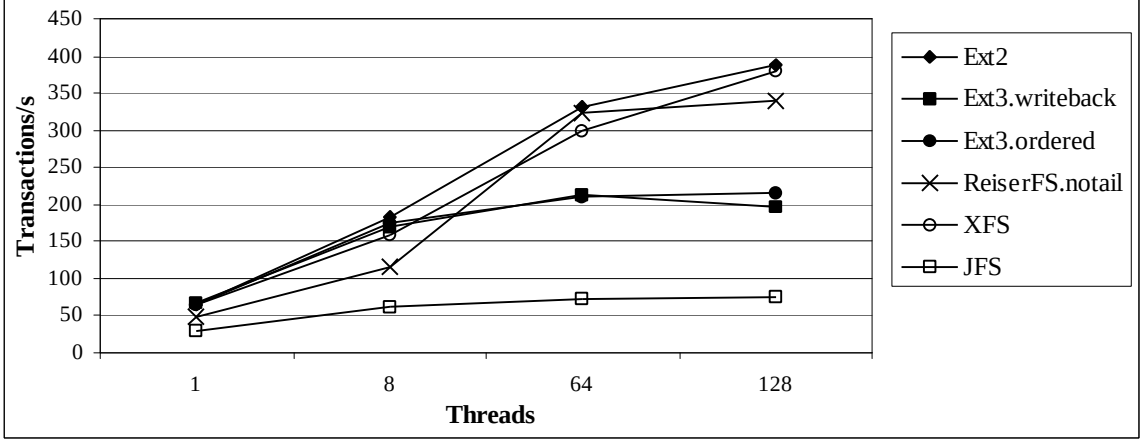

Figure 8: Pgmeter, Large System, Sequential Read 64 KB Workload

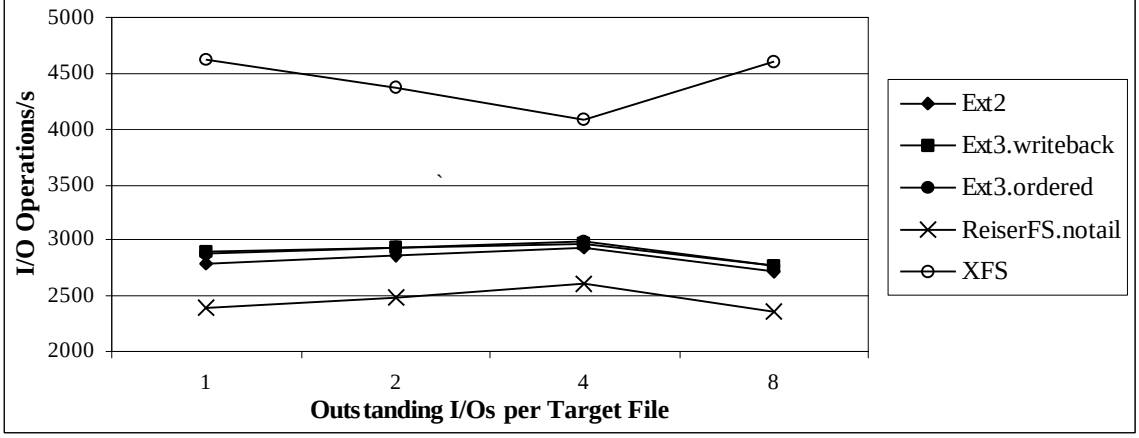

Figure 9: Pgmeter, Large System, Sequential Write 64 KB Workload

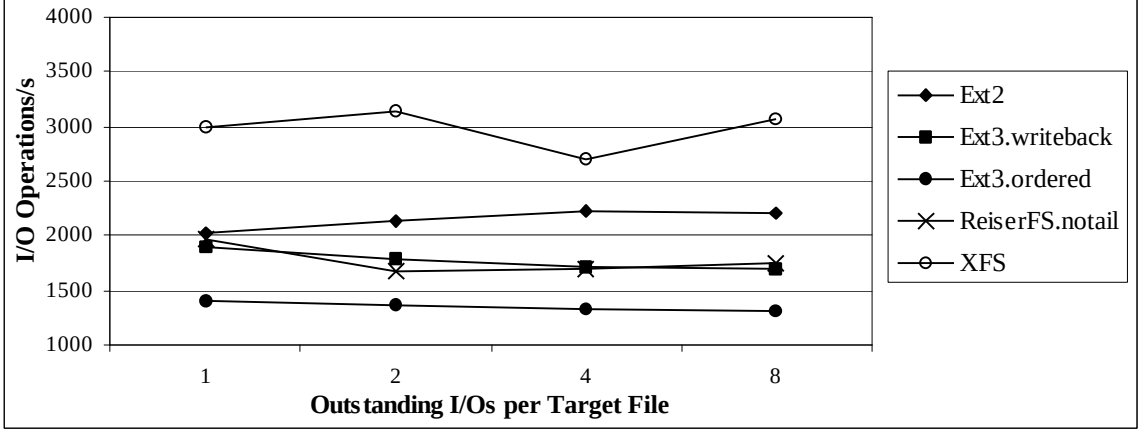

Figure 10: Pgmeter, Large System, File Server Workload

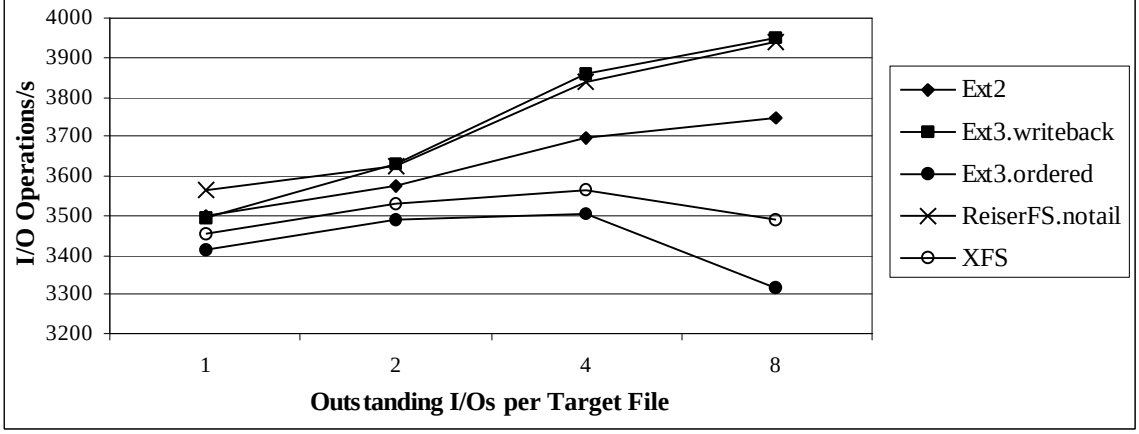

Figure 11: AIM7, Large System, 28-File Case

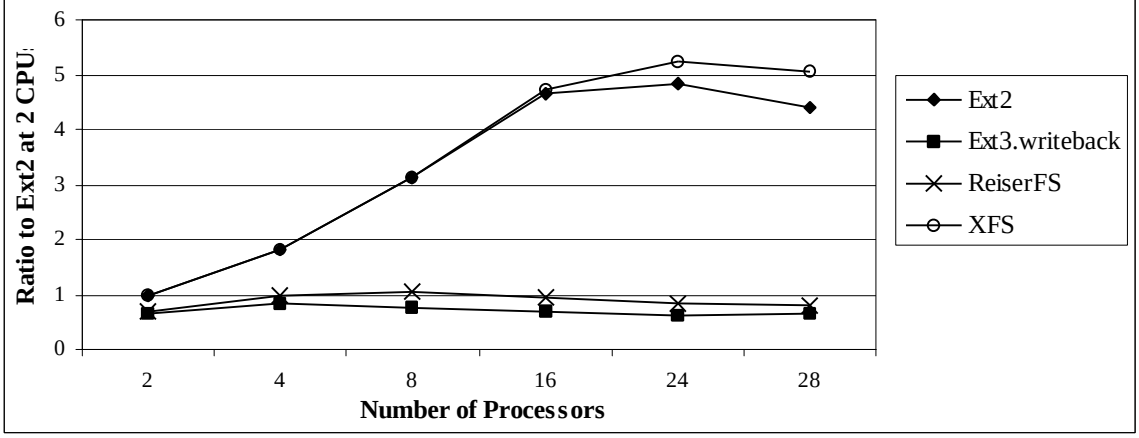

Figure 12: Pgmeter, Large System, Web Server Workload, 112-File Case