SOLARIS™ ZFS AND RED HAT ENTERPRISE LINUX EXT3 FILE SYSTEM PERFORMANCE

White Paper
June 2007

Table of Contents

Chapter 1
Executive Overview

The Solaris™ 10 06/06 Operating System (OS) introduces a major new data management technology — the Solaris ZFS file system. Replacing the traditionally separate file system and volume manager functionality found in most operating environments, Solaris ZFS provides immense capacity and performance coupled with a proven data integrity model and simplified administrative interface.

Today, file systems aim to provide efficient access to data storage and information. While traditional UNIX® file systems and Solaris ZFS alike can benefit from additional performance enhancements, the performance profile and characteristics of Solaris ZFS are qualitatively different from file systems such as the ext3 file system used in the Red Hat Enterprise Linux 4 Enterprise Server environment and similar architectures. This white paper explores the performance characteristics and differences of Solaris ZFS and the ext3 file system through a series of benchmarks based on use cases derived from common scenarios, as well as the IOzone File System Benchmark (IOzone benchmark) which tests specific I/O patterns. Testing results reveal:

- Solaris ZFS outperforms the ext3 file system in tests that represent the storage and data requests typically made by database, mail server, and Web applications.
- The various ext3 file system mount options available require making a trade-off between data integrity and performance — creating implications for environments in which continuous access to data is critical. Such trade-offs are not necessary in Solaris ZFS.

Figure 1-1 illustrates the differences between Solaris ZFS and the ext3 file system (mounted as ordered and journalled) in a number of tests. In many cases, Solaris ZFS performs better at the initial release. In some cases performance levels are worse — but in almost all cases Solaris ZFS performs differently. These results, as well as supporting testing data described in this document, strive to give organizations sufficient detail on the differences between Solaris ZFS and the ext3 file system so that intelligent decisions can be made about when each technology could or should be used. Indeed, the results presented here can help enterprises reach performance goals, if the goals can be quantified.

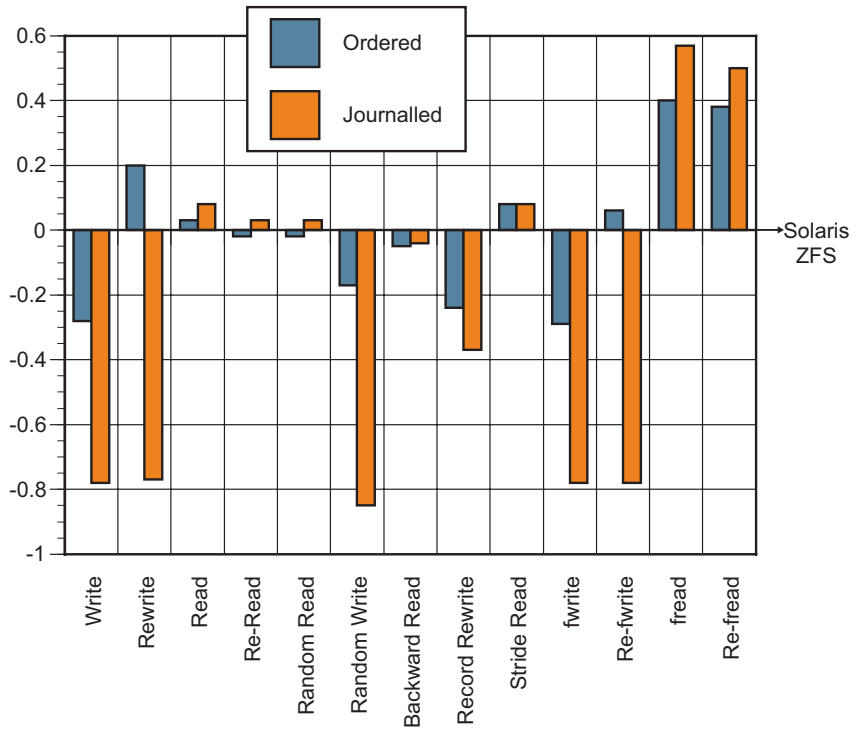

Figure 1-1. IOzone testing summary for the Red Hat Enterprise LInux ext3 file system versus Solaris ZFS

Chapter 2
File System Overview

More information on ReiserFS can be found at http://namesys.com

The Solaris OS and Linux environments offer a choice of file systems to users. For example, applications that run on these operating systems and require high-performance bulk I/O or a storage area network (SAN) aware file system can utilize the Sun StorageTek™ QFS software. Hierarchical storage management is made available in the Solaris OS through the Sun StorageTek SAM-FS software. In addition to the ext3 file system, Linux users can select from the ReiserFS, JFS, and XFS file systems.

This chapter provides a brief technical introduction to Solaris ZFS and the ext3 file systems, and highlights the features that can impact performance. More information can be found in the references listed at the end of this document. A good general primer on Linux file systems is provided in *Linux File Systems* by William von Hagen. Further detail on the design and operation of Solaris ZFS can be found in the *Solaris ZFS Administration Guide* available at *docs.sun.com*, as well as the OpenSolaris Project Web site located at *opensolaris.org/os/community/zfs*.

Note – The term *metadata* is used throughout this document. When a new file is written data must be stored, as well as overhead information that keeps track of where data is located on the storage media. Such metadata consists of directory information, space allocation, and any other data associated with a file that is not part of the file contents.

Solaris™ ZFS

Solaris ZFS is designed to overcome the limitations of existing file systems and volume managers in UNIX environments.

Simplified Storage Device and File System Administration

In many operating systems, disk partitioning, logical device creation, and new file system formatting tend to be detailed and slow operations. Because these relatively uncommon tasks are only performed by system administrators, there is little pressure to simplify and speed such administrative tasks. Mistakes are easy to make and can have disastrous consequences. As more users handle system administration tasks, it can no longer be assumed that users have undergone specialized training.

In contrast, Solaris ZFS storage administration is automated to a greater degree. Indeed, manual reconfiguration of disk space is virtually unnecessary, but is quick and intuitive when needed. Administrators can add storage space to, or remove it from, an existing file system without unmounting, locking, or interrupting file system service.

Administrators simply state an intent, such as *make a new file system*, rather than perform the constituent steps.

Pooled Storage and Integrated Volume Management

Red Hat Enterprise Linux makes a one-to-one association between a file system and a particular storage device. Using the Logical Volume Manager (LVM2), the file system is assigned to a specific range of blocks on the logical storage device. Such a scheme is counterintuitive — file systems are intended to virtualize physical storage, and yet a fixed binding remains between the logical namespace and a logical or physical device.

Solaris ZFS decouples the namespace from physical storage in much the same way that virtual memory decouples address spaces from memory banks. Multiple file systems can share a pool of storage. Allocation is moved out of the file system into a storage space allocator that parcels out permanent storage space from a pool of storage devices as file systems make requests. In addition, the volume management and file system model used by the ext3 file system and other implementations makes it difficult to expand and contract file systems, share space, or migrate live data. By folding the functionality and namespace of the volume manager into the file system level, operations can be undertaken in terms of files and file systems rather than devices and blocks.

Strong Data Integrity

The file system mount options in the Red Hat Enterprise Linux environment often require users to make a trade-off between performance and data integrity. On the other hand, Solaris ZFS provides consistent on-disk data and error detection and correction, ensuring data consistency while maintaining high performance levels.

File system corruption can be caused by disk corruption, hardware or firmware failures, or software or administrative errors. Validation at the disk block interface level can only catch some causes of file system corruption. Traditionally, file systems have trusted the data read in from disk. However, if the file system does not validate read data, errors can result in corrupted data, system panics, or more. File systems should validate data read in from disk in order to detect downstream data corruption, and correct corruption automatically, if possible, by writing the correct block back to the disk. Such a validation strategy is a key design goal for Solaris ZFS.

Immense Capacity

The maximum size of an ext3 file system is 8 TB[1]. However, one petabyte datasets are plausible today, and storage capacity is currently doubling approximately every nine to 12 months. Assuming the rate of growth remains the same, 16 exabyte (EB) datasets may begin to emerge in only ten years. The lifetime of a typical file system implementation is measured in decades. Unlike many of today's file systems, Solaris

1. An Overview of the Red Hat Enterprise Linux 4 Product Family, Revision 4b., February 2005. http://www.redhat.com/f/pdf/rhel4/RHEL4OverviewWP.pdf

ZFS uses 128-bit block addresses and incorporates scalable algorithms for directory lookup, metadata allocation, block allocation, I/O scheduling, and other routine operations, and does not depend on repair utilities such as `fsck` to maintain on-disk consistency.

Red Hat Enterprise Linux ext3 File System

More information on the original Linux file system can be found in *Operating System Design and Implementation* by Tanenbaum and Woodhull, Prentice Hall, 1987.

Designed for educational purposes, the original Linux file system was limited to 64 MB in size and supported file names up to 14 characters. In 1992, the ext file system was created, and increased the file system size to 2 GB and file name length to 255 characters. However, file access, modification, and creation times were missing from file system data structures and performance tended to be low. Modeled after the Berkeley Fast File System, the ext2 file system used a better on disk layout, extended the file system size limit to 4 TB and file name sizes to 255 bytes, delivered improved performance, and emerged as the de facto standard file system for Linux environments.

More information on the logging capabilities of the ext3 file system can be found in *EXT3, Journaling File System* by Dr. Stephen Tweedie located at http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html.

An evolution of the ext2 file system, the ext3 file system adds logging capabilities to facilitate fast reboots following system crashes. Key features of the ext3 file system include:

- *Forward and backward compatibility with the ext2 file system.* An ext3 file system can be remounted as an ext2 file system and vice versa. Such compatibility played a role in the adoption of the ext3 file system.
- *Checkpointing.*The ext3 file system provides checkpointing capabilities, the logging of batches of individual transactions into compound transactions in memory prior to committing them to disk to improve performance. While checkpointing is in progress, a new compound transaction is started. While one compound transaction is being written to disk, another is accumulating. Furthermore, users can specify an alternative log file location which can enhance performance via increased disk bandwidth.
- *Volume management.*The ext3 file system relies on the LVM2 package to perform volume management tasks.

Logging in the ext3 File System

The ext3 file system supports different levels of journalling which can be specified as mount options. These options can impact data integrity and performance. This section describes the mount options, and the testing results presented in Chapter 3 demonstrate the effects of their use.

- *data=journal*
  Originally, the ext3 file system was designed to perform full data and metadata journalling. In this mode, the file system journals all changes to the file system, whether the changes affect data or metadata. Consequently, data and metadata can

be brought back to a consistent state. Full data journalling can be slow. However, performance penalties can be mitigated by setting up a relatively large journal.

- *data=ordered*
  The ext3 file system includes an operational mode that provides some of the benefits of full journalling without introducing severe performance penalties. In this mode, only metadata is journalled. As a result, the ext3 file system can provide overall file system consistency, even though only metadata changes are recorded in the journal. It is possible for file data being written at the time of a system failure to be corrupted in this mode. Note that ordered mode is the default in the ext3 file system.
- *data=writeback*
  While the writeback option provides lower data consistency guarantees than the journal or ordered modes, some applications show very significant speed improvement when it is used. For example, speed improvements can be seen when heavy synchronous writes are performed, or when applications create and delete large volumes of small files, such as delivering a large flow of short email messages. The results of the testing effort described in Chapter 3 illustrate this topic.

  When the writeback option is used, data consistency is similar to that provided by the ext2 file system. However, file system integrity is maintained continuously during normal operation in the ext3 file system. However, in the event of a power failure or system crash, the file system may not be recoverable if a significant portion of data was held only in system memory and not on permanent storage. In this case, the file system must be recreated from backups. Often, changes made since the file system was last backed up are inevitably lost.

File System Concepts

The ext3 file system and LVM2 present physical storage to the application via several layers of abstraction which must be created by system administrators.

- *Physical volume.*While a physical volume is typically a hard disk, it may be a device that appears to the system like a hard disk, such as a RAID device. The physical volume encapsulates the disk and brings it under LVM2 control.
- *Logical volume.* A logical volume is visible as a standard block device. As a result, a logical volume can contain a file system.
- *Volume group.* The highest level of abstraction used within the LVM, the volume group gathers a collection of Logical Volumes and Physical Volumes into one administrative unit.
- *File system.* The file system defines the namespace for storing, caching, and accessing files.

In contrast, Solaris ZFS presents storage in pools and file systems. The pool contains all the disks in the system, and can contain as many file systems as are needed.

Table 2-1 lists the activities required to create usable storage using Solaris ZFS and the ext3 file system, as well as the time observed for these tasks.

Table 2-1. The file system creation procedure for Solaris ZFS and the ext3 file system

Solaris ZFS	ext3 and LVM2
# **zpool create -f tank** (32 disks) # **zfs create tank/fs**	# **for i in** (32 disks) **do** **parted /dev/$i mklabel sun** **done** # **pvcreate /dev/sdb /dev/sdc** ... (32 disks) # **vgcreate v0** (32 disks) # **lvcreate -v -i32 -L2T -nd0 v0** # **mkfs -t ext3 /dev/v0/d0** mke2fs 1.35 (28-Feb-2004) file system label= OS type: Linux Block size=4096 (log=2) Fragment size=4096 (log=2) 268435456 inodes, 536870912 blocks 26843545 blocks (5.00%) reserved for the super user First data block=0 Maximum file system blocks=4294967296 16384 block groups 32768 blocks per group, 32768 fragments per group 16386 inodes per group Superblock backups stored on blocks: 32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208, 4096000, 7962624, 11239424, 20480000, 23887872, 71663616, 78675968, 102400000, 214990848, 512000000 Writing inode tables: done Creating journal (8192 blocks): done Writing superblocks and file system accounting information: done This file system will be automatically checked every 39 mounts or 180 days, whichever comes first. Use tune2fs -c or -i to override. # **mount -t ext3 -o data=writeback /dev/v0/d0 /mnt**
Time: 17.5 seconds	**Time:** 30 minutes

Chapter 3
Benchmark Tests and Results

This chapter describes the hardware platforms, operating system and software versions, and benchmarks used for testing efforts, as well as the results observed. In some cases, Solaris ZFS outperforms the ext3 file system. As expected, Solaris ZFS does not perform as well in other scenarios.

Reproducing the Benchmark

Having default parameters that become outdated creates two problems. First, there is no such thing as a standard configuration. In addition, different workloads exercise the system differently and results across research papers are not comparable. Second, not all research papers precisely describe the parameters used, making it difficult to reproduce results[1].

1. *Benchmarking File System Benchmarks*, N. Joukov, A. Traeger, CP Wright, Zadok, ETechnical Report FSL-05-04b, CS Department, Stony Brook University, 2005. http://www.fsl.cs.sunysb.edu/docs/fsbench/fsbench.pdf

During competitive system software testing, It is important to ensure the underlying server and storage hardware — and the benchmarking code — is as close to identical as possible so that proper comparisons can be drawn. The testing effort described in this document aimed to ensure comparability through:

- Use of the same physical server and storage hardware for all tests
- Use of identical benchmark source code compiled on both platforms
- Use of operating systems that were installed and used out of the box, with no special tuning

Hardware and Operating System Configuration

Table 3-1 describes the platforms on which the testing was conducted. Postmark version 1.5 and BenchW 1.1 were used on both platforms. PostgreSQL version 8.0.1 was used on the Solaris OS, while version 8.1 was used on the Red Hat Enterprise Linux platform. The only tuning performed involved increasing the shared memory segment parameter on the Red Hat Enterprise Linux platform. This tuning was done to ensure the same PostgreSQL configuration parameters could be used on both the Solaris and Linux platforms. The */etc/sysctl.conf* file was amended to set `kernel.shmmax=1073741824`.

Table 3-1. Hardware and operating system configuration used for testing efforts

Operating System	Server	Storage
Red Hat Enterprise Linux Application Server (Linux release 2.6.9-22.ELsmp)	• Sun Fire™ x4200 server • Two 2.2 GHz AMD Opteron™ processors (four core) • 8 GB RAM	• Sun StorEdge™ 3500 arrays (4) with 32 x 72 GB disks • Fiber channel interface 4 x 2 Gb PCI-X 133 MHz
Solaris 10 OS Update 2 06/06 (Including Solaris ZFS)	• Sun Fire x4200 server • Two 2.2 GHz AMD Opteron processors (four core) • 8 GB RAM	• Sun StorEdge 3500 arrays (4) with 32 x 72 GB disks • Fiber channel interface 4 x 2 Gb PCI-X 133 MHz

IOzone File System Benchmark

For more information, see http:// linuxperf.sourceforge.net/iozone/iozine.php

Available on a wide variety of systems and operating systems, the IOzone file system benchmark generates and measures a variety of file operations. It was used to test the following I/O operations: read, write, re-read, rewrite, read backwards, record re-write, read strided, fread, fwrite, re-fread, re-fwrite, random read, and random write. IOzone was selected for testing for a variety of reasons, including:

- IOzone is freely available, enabling readers to reproduce the results presented.
- IOzone provides data in convenient spreadsheet formats for post-processing, as well as tools for graphical manipulation of the output.
- IOzone tests multiple dimensions of I/O, iterating over differing file sizes, record sizes and I/O patterns.
- Previous studies of Linux file system performance have used the IOzone benchmark, and it is highly regarded in the community.

IOzone Test Results

The following sections present the results of the IOzone benchmark testing efforts.

IOzone Write

The IOzone write test measures the performance of writing a new file. Typically, initial write performance is lower than that of rewriting a file due to metadata overhead. The throughput of a journalled ext3 file system is low and constant, regardless of the size of the data buffer transferred. Figure 3-1 shows the results obtained during IOzone write testing. In all cases, Solaris ZFS throughput is as great as ext3 file system throughput. For writes greater than 32 KB, Solaris ZFS is 150 Mb/s faster on the system tested.

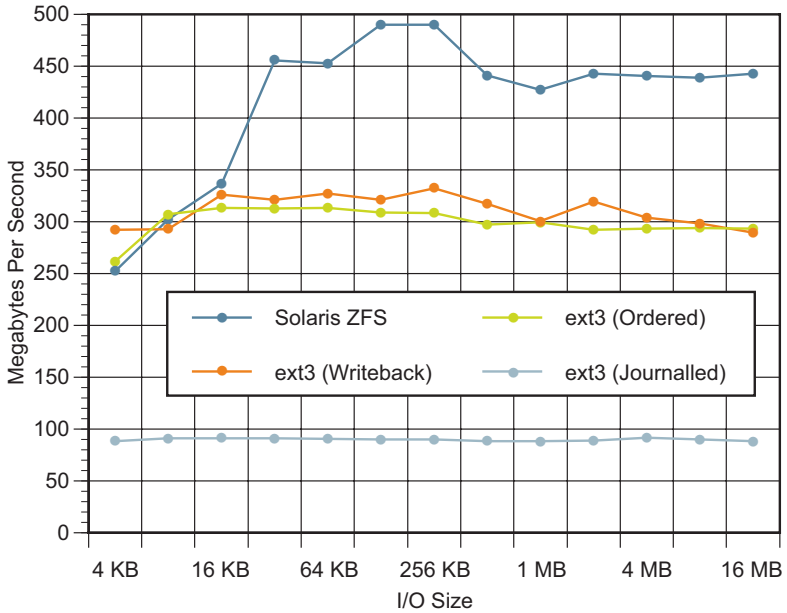

Figure 3-1. IOzone write test results

IOzone Rewrite

The IOzone rewrite test measures the performance of writing a file that already exists. Writing to a file that already exists requires less as the metadata already exists. Typically, rewrite performance is higher than the performance of writing a new file. Figure 3-2 details the IOzone rewrite results obtained during testing efforts.

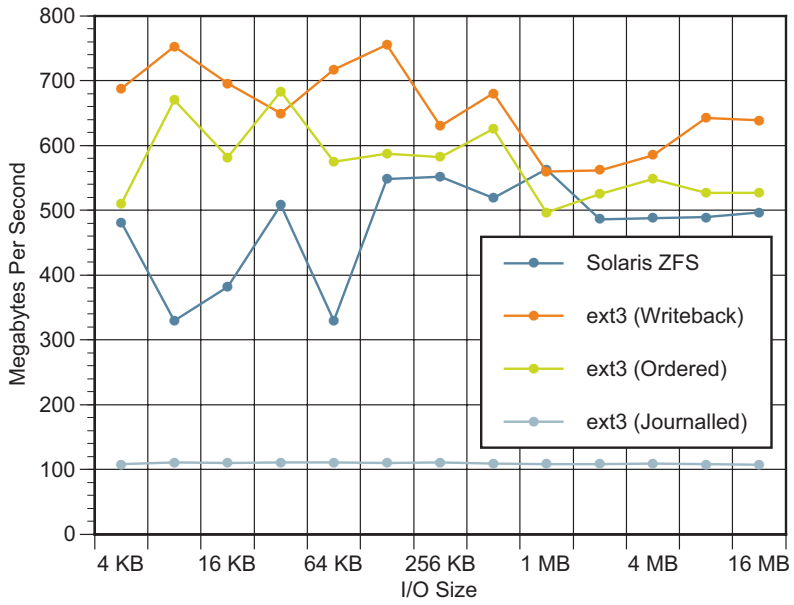

Figure 3-2. IOzone rewrite test results

IOzone Read

The IOzone read test measures the performance of reading an existing file. Figure 3-3 shows the IOzone read results obtained during testing efforts.

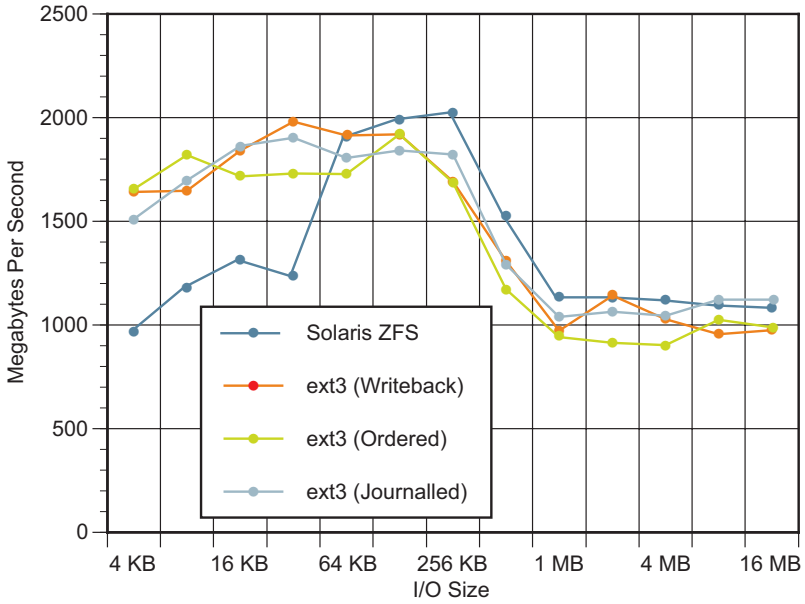

Figure 3-3. IOzone read test results

IOzone Re-read

The IOzone re-read test measures the performance of reading a file that was recently read. Re-read performance can be higher as the file system can maintain a data cache for files read recently, which can be used to satisfy read requests and improve throughput. Figure 3-4 shows the results of the IOzone re-read test.

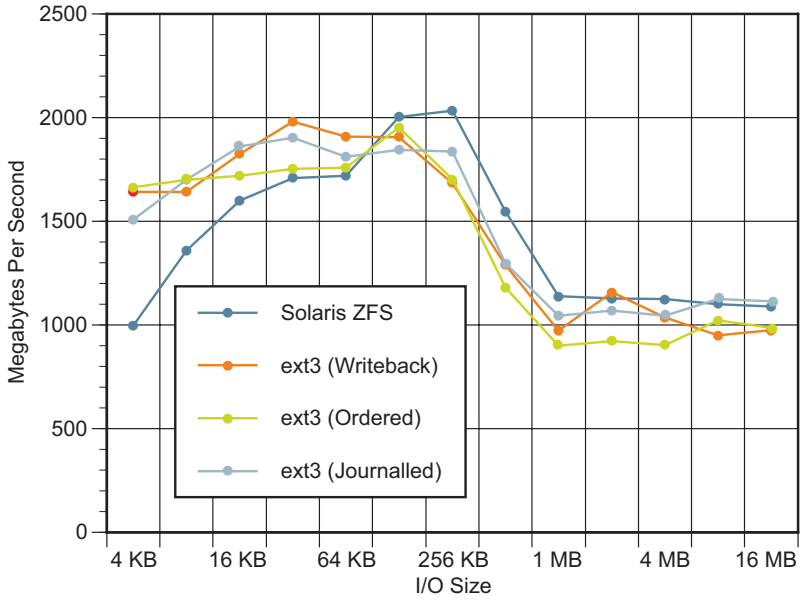

Figure 3-4. IOzone re-read test results

IOzone Record Rewrite

The IOzone record rewrite test measures the performance of writing and re-writing a section of a file. Figure 3-5 illustrates the results obtained during testing efforts.

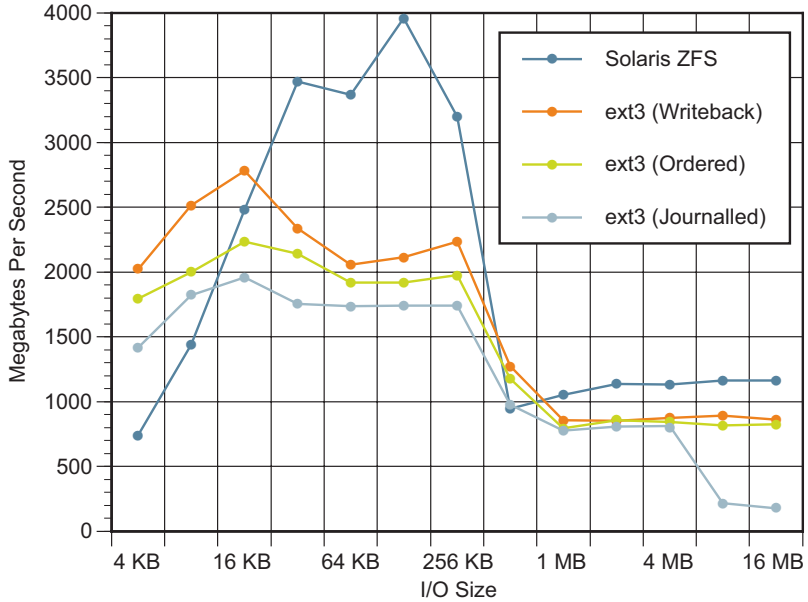

Figure 3-5. IOzone record rewrite test results

IOzone Random Read

The IOzone random read test measures the performance of reading a file at random locations. System performance can be impacted by several factors, such as the size of operating system cache, number of disks, seek latencies, and more. The optimum size for Solaris ZFS random writes is 256 KB. As is illustrated in Figure 306, Solaris ZFS provides approximately the same throughput as the ext3 file system.

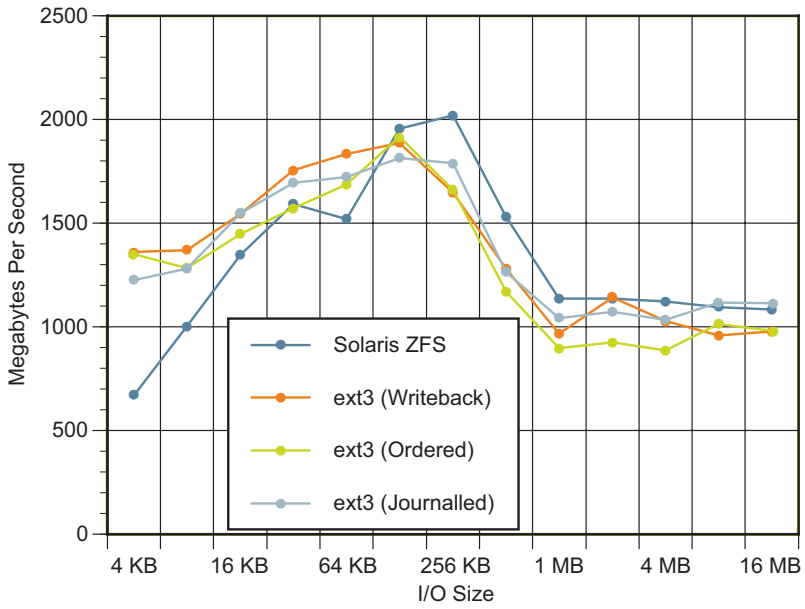

Figure 3-6. IOzone random read test results

IOzone Random Write

The IOzone random write test measures the performance of writing a file a random locations. System performance can be impacted by several factors, such as the size of operating system cache, number of disks, seek latencies, and more. Efficient random write performance is important to the operation of transaction processing systems. Figure 3-7 depicts the results of the IOzone random write test. As illustrated, Solaris ZFS performs better with larger transfer sizes. With smaller sizes, the caching write-behind of the ext3 file system in the writeback and journalled modes proves more effective. Test results confirm that journalling provides a consistently high overhead.

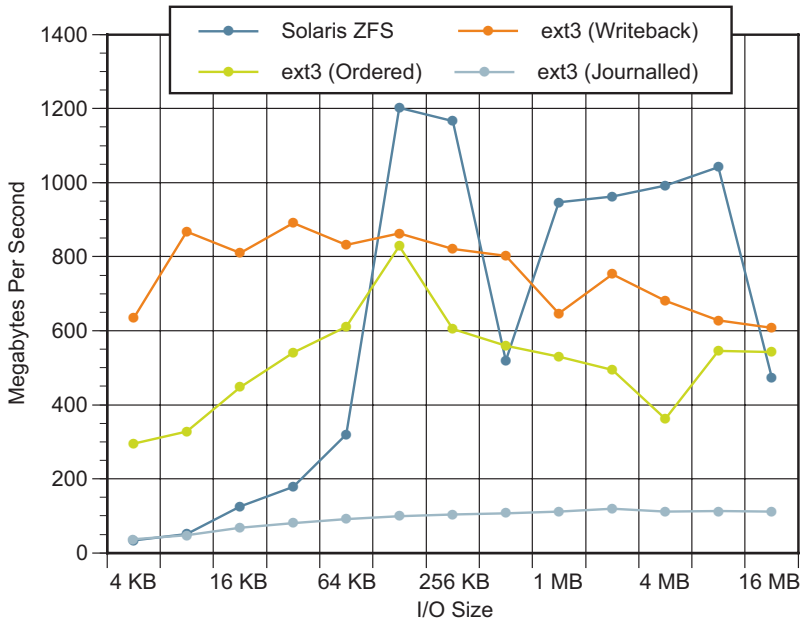

Figure 3-7. IOzone random write test results

IOzone Backwards Read

The IOzone backwards read test measures the performance of reading a file backwards. Many applications perform backwards reads, such as MSC Nastran and video editing software. While many file systems include special features that speed forward file reading, few detect and enhance the performance of reading a file backwards. Figure 3-8 shows that Solaris ZFS and the ext3 file system perform similarly on the backwards read test.

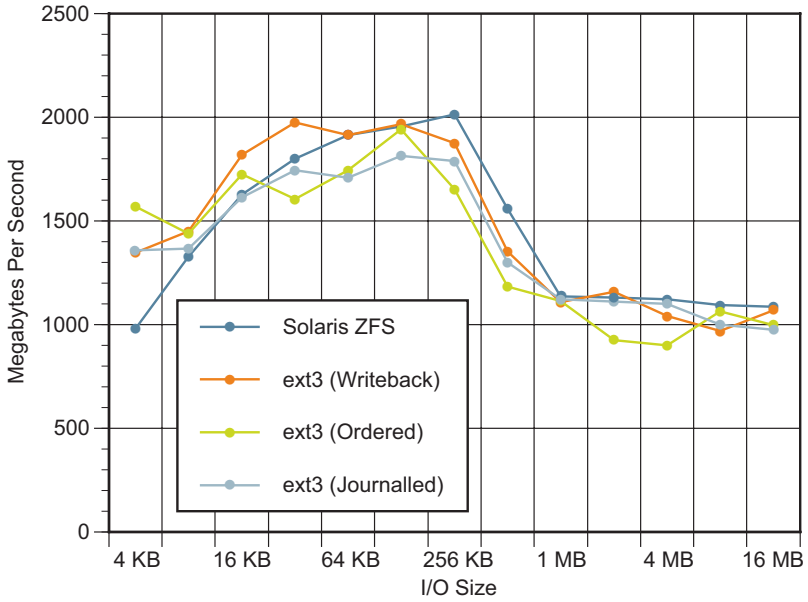

Figure 3-8. IOzone backwards read test results

IOzone Strided Read

The IOzone strided read test measures the performance of reading a file with strided access behavior. For example, the test might make the following types of read requests: Read at offset zero for a length of 4 KB, seek 200 KB, read for a length of 4 KB, seek 200 KB, and so on. Figure 3-9 depicts the results of the IOzone strided read test. During the test, the system read 4 KB, did a seek of 200 KB, and repeated the pattern. Such a patterns is typical behavior for applications accessing a particular region of a data structure that is contained within a file. Most file systems do not detect such behavior or implement techniques to enhance access performance. Note that this type of access behavior can produce interesting performance anomalies, such as the lower performance of Solaris ZFS for 16 KB reads.

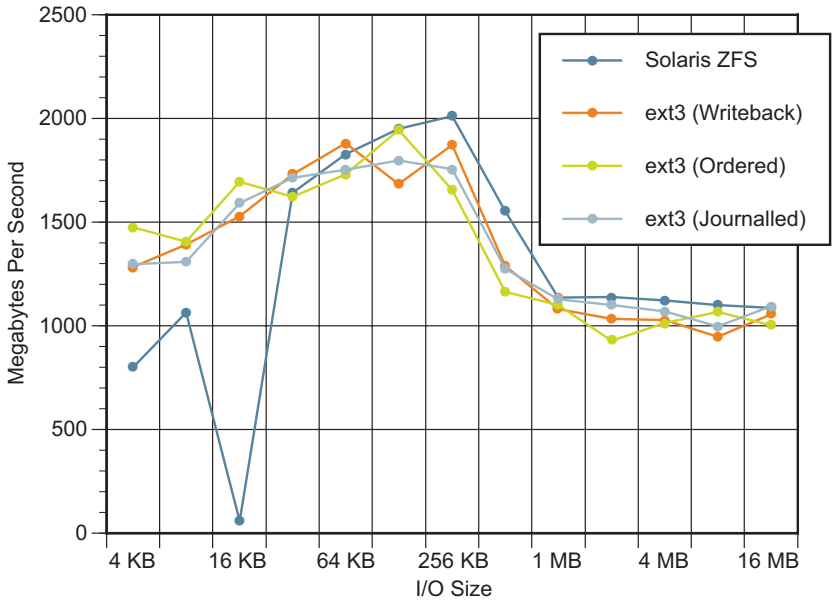

Figure 3-9. IOzone strided read test results

IOzone fwrite

The IOzone fwrite est measures the performance of writing a file using the `fwrite()` library routine that performs buffered write operations using a buffer within the user's address space. If an application writes small amounts per transfer, the buffered and blocked I/O functionality of the `fwrite()` routine can enhance the performance of the application by reducing the number of operating system calls and increasing the size of transfers. Figure 3-10 shows the test results obtained. Note the test writes a new file so metadata overhead is included in the measurement. When compared with the IOzone write results depicted in Figure 3-1, testing results reveal the following:

- The overhead associated with running the ext3 file system in journalled mode results in consistently low throughput regardless of the presence of an application level buffer.

- Writes larger than 128 KB yield constant throughput for the ext3 file systems running in writeback and ordered modes, as well as Solaris ZFS. In the case of the ext3 file system, the throughput is the same for the fwrite and write tests. For Solaris ZFS, throughput degrades with the double-buffering associated with the fwrite test.

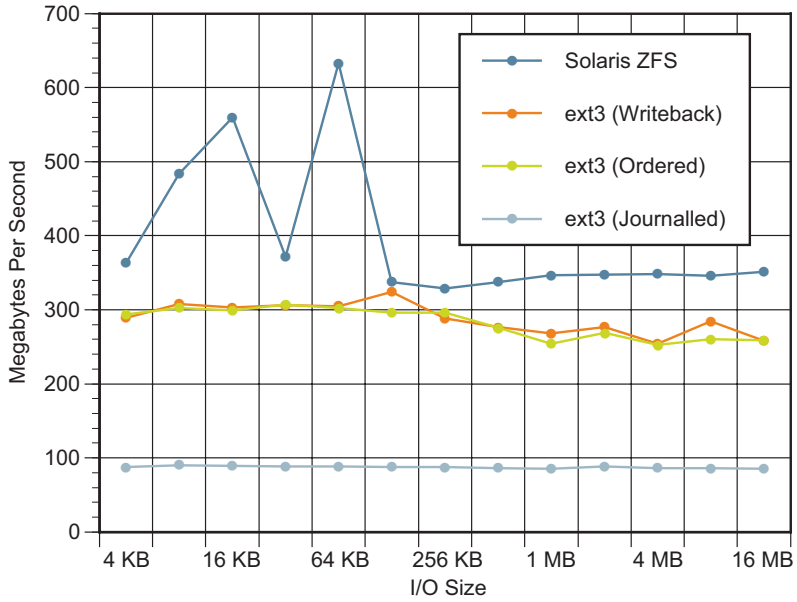

Figure 3-10. IOzone fwrite test results

IOzone Re-fwrite

The IOzone re-fwrite test performs repetitive rewrites to portions of an existing file using the `fwrite()` interface. Figure 3-11 illustrates the results obtained.

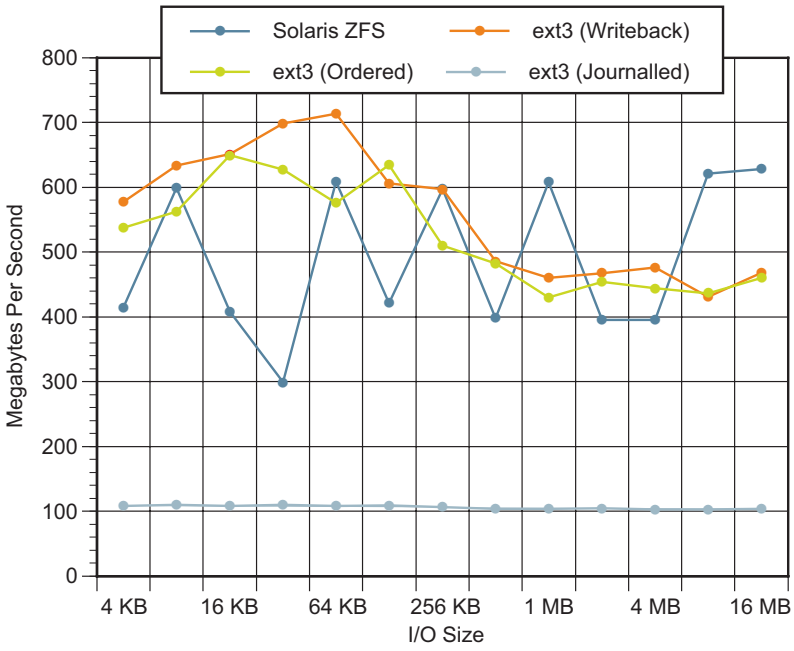

Figure 3-11. IOzone re-fwrite test results

IOzone fread

The IOzone fread test measures file read performance using the `fread()` routine, which performs buffered and blocked read operations using a buffer located in the user's address space. If applications use very small transfers, the buffered and blocked I/O functionality of the `fread()` routine can enhance performance by using fewer operating system calls and larger transfer sizes. Figure 3-12 shows the results obtained.

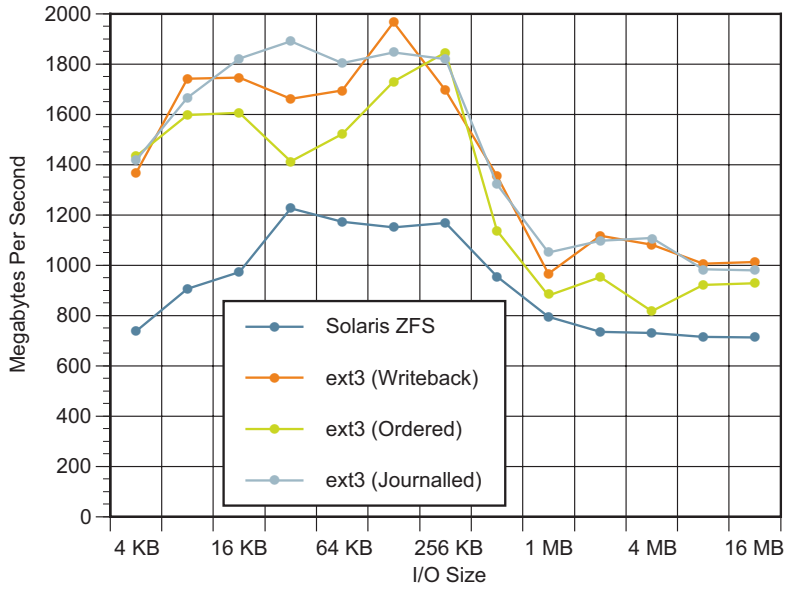

Figure 3-12. IOzone fread test results

IOzone Re-fread

The Iozone re-fread test is similar to the IOzone fread test, except that the file being read was read in the recent past. Reading recently read data can result in higher performance as the file system is likely to have the file data stored in a cache.

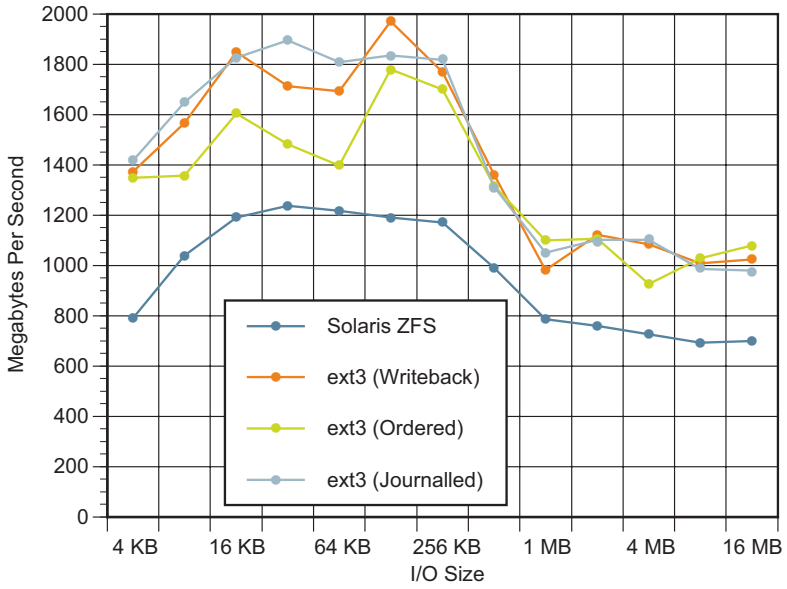

Figure 3-13. IOzone re-fread test results

PostgreSQL and BenchW — A Data Warehousing Benchmark

More information can be found at http://postgresql.org and http://benchw.sourceforge.net

PostgreSQL is an open source relational database manager released under a BSD license. Note that PostgreSQL is integrated into the Solaris 10 06/06 OS, and Sun provides support to organizations looking to develop and deploy open source database solutions and use PostgreSQL in enterprise environments. PostgreSQL was used for testing efforts as it is can be deployed in both Solaris OS and Red Hat Enterprise Linux environments.

BenchW is a data warehouse benchmarking toolkit that aims to compare the capabilities of several different database managers for data warehouse activities, such as data loading, index creation, and query performance. The BenchW benchmark attempts to keep things simple and realistically model the environment in which many ad hoc query tools work. As a result, many of the elaborate tuning optimizations for data warehousing are not used. The comparison concentrates on bulk queries rather than testing multiple threads or concurrent updates. The testing effort document here used the BenchW benchmark because of its wide availability and ability to work with PostgreSQL. In addition to PostgreSQL, BenchW can generate appropriate code and data for several commercial and community-based relational database managers.

BenchW Test

BenchW uses a data model in an idealized *star* schema with three default dimension tables: `dim0`, `dim1`, `dim2`. The first dimension is time, and includes a date column representing a time measure. The other two dimensions are generic. A fact table, `fact0`, represents generic observations of interest. The *star* relationship is expressed through the inclusion of a foreign key column in the fact table for each primary key of the dimension tables. These relationships are not formally represented as constraints, since they cannot be implemented by all database managers. During testing efforts, the code was modified to encapsulate the whole suite of tests in one script and capture timing information in SQL.

The default tuning settings described in the BenchW documentation were used across all operating systems for the testing effort. The settings are listed below. Only one variable, `checkpoint_segments`, used a non-default value. The value was raised for both the Solaris OS and Red Hat Enterprise Linux environments to prevent excessive checkpointing and warning messages. While using default values constrains database managers, it provides a level basis across platforms.

```
shared_buffers = 10000
sort_mem = 20480
effective_cache_size = 48000
random_page_cost = 0.8
wal_buffers = 1024
checkpoint_segments = 32
```

The SQL script used during testing efforts is listed in Appendix A. The tests documented here used the `loadgen` utility to generate a 40 GB main table.

BenchW Test Results

The following sections present the results of the BenchW benchmark testing efforts.

BenchW Unindexed Queries

The BenchW unindexed queries test loads data in a series of 8 KB sequential writes. The database performs entire table scans and writes the results to temporary files for sorting. A metric of success is the reduction in time taken. Measured results can be found in Figures 3-14 and 3-15.

When operating in writeback mode, the ext3 file system has an advantage. Ordering and journalling create considerable overhead, particularly in the data load phase. However, since data loading can be repeated in the event of a crash, few database administrators use the additional safeguards of these modes for such operations. The file system can be mounted in writeback mode for the data load, and remounted in ordered or journalled mode for online access. Such considerations do not apply to Solaris ZFS, which maintained a level of performance in during the test between that of of the ext3 file system in writeback and journalled modes.

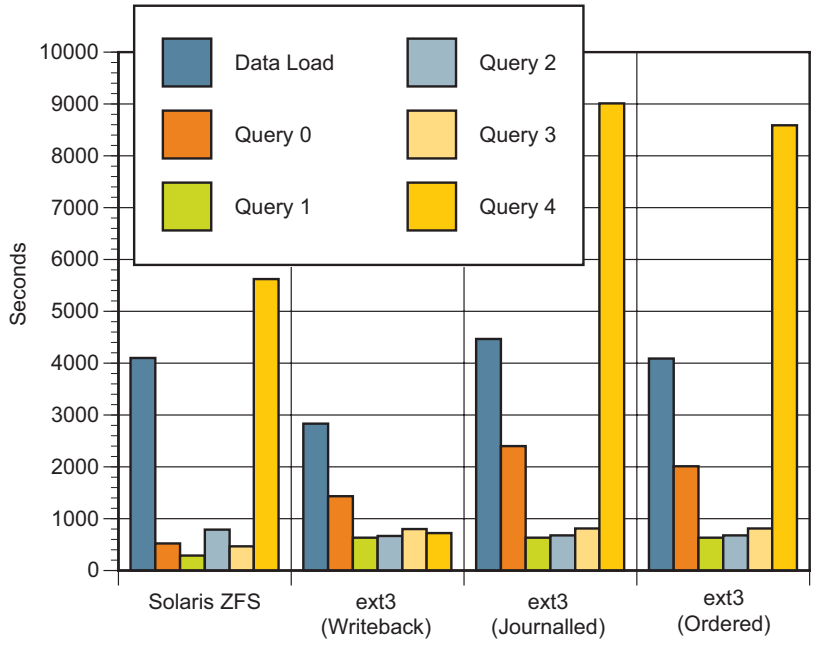

Figure 3-14. BenchW unindexed queries results

BenchW Indexed Queries

Indexed queries do not rely on the file system for metadata transactions and block allocation as much as unindexed queries. In the tests performed, the ext3 file system performed similarly in all three operating modes, while Solaris ZFS took between half and two-thirds of the time to perform the same queries. Figure 3-15 shows the results.

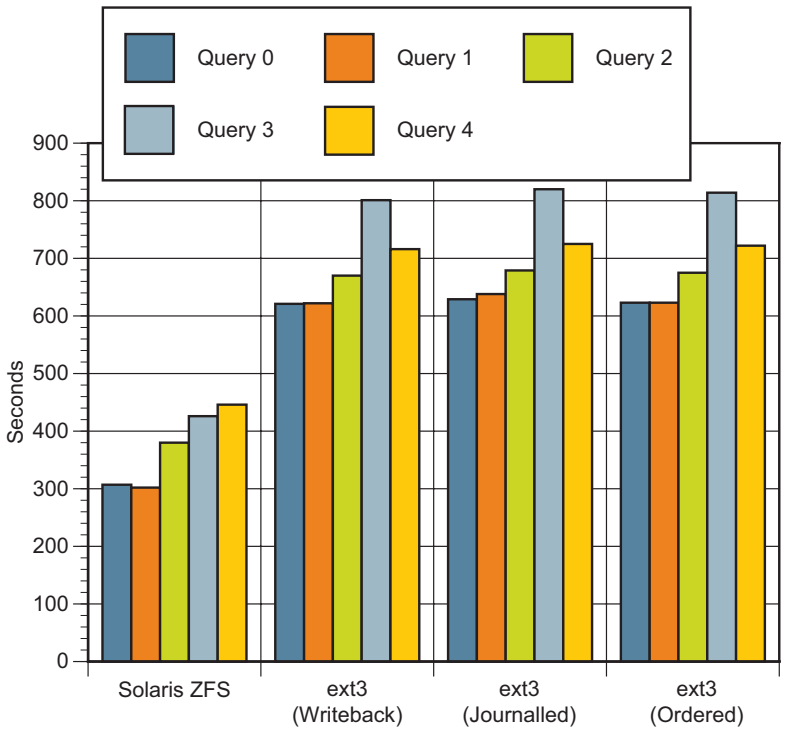

Figure 3-15. BenchW indexed queries results

Postmark — A Web and Mail Server Benchmark

Designed to emulate applications such as software development, email, newsgroup servers and Web applications, the Postmark utility from Network Appliance has emerged as an industry-standard benchmark for small file and metadata-intensive workloads. More information on the Postmark benchmark can be found in *Postmark: A New File System Benchmark*.

Postmark works by creating a pool of random text files that range in size between configurable high and low bounds. During the test, the files are modified continually. Building the file pool enables the production of statistics on small file creation performance. Transactions are then performed on the pool. Each transaction consists of a pair of create/delete or read/append operations. The use of randomization and the ability to parameterize the proportion of create/delete to read/append operations helps overcome the effects of caching in various parts of the system.

The Postmark test runs as a single process that iterates over an increasing number of created files and transactions. Table 3-2 defines the terms small, medium, and large that are referenced in the graphs and discussions that follow.

Table 3-2. Single threaded Postmark variables

Test	Files	Transactions
Small	1,000	50,000
Medium	20,000	50,000
Large	20,000	100,000

Table 3-3 lists the parameters and settings used during testing efforts.

Table 3-3. Single threaded Postmark parameters

Parameter Syntax	Comment
`set number 1000`	Number of files (variable)
`set transactions 50000`	Number of transactions (variable)
`set subdirectories 1000`	Directories across which the files are scattered
`set size 10000 15000`	Range of file sizes (9.77 KB to 14.65 KB)
`set location /cygdrive/s`	Root directory of the test (target file system)
`set buffering false`	Do not use buffering in the C library
`set report verbose`	
`random`	The random number generator seed (default 42)

Postmark Test Results

In this test, the higher the number of transactions performed per second the better. The small workload executes 20,000 transactions over 1,000 files. The entire workload, including all files and metadata, can be held in memory. In this case, the ext3 file system running in either writeback or ordered mode has an advantage. However, as soon as the file population overruns the cache and the transaction workload becomes sustained (50,000 or 100,000 transactions over 20,000 files), Solaris ZFS emerges as the frontrunner. While the additional processing incurred by the ext3 file system when operating in journalling mode makes it less suitable for this type of workload, it is unlikely to be used in such environments. Journalling mode is often used in production Web and mail servers. The test results are summarized in Figure 3-16.

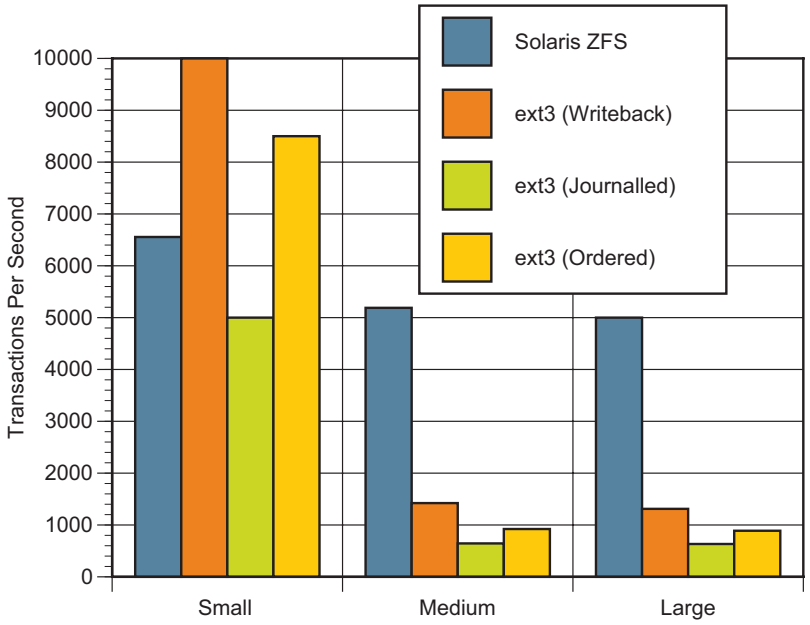

Figure 3-16. Postmark test results

Chapter 4
Summary

This white paper details the performance characteristics of Solaris ZFS and the ext3 file system through IOzone, BenchW, and Postmark benchmark testing. For the conditions tested, Solaris ZFS can outperform the Red Hat Enterprise Linux ext3 file system for many workloads, especially Postmark and relational database indexed queries. In other cases, Solaris ZFS exhibits comparable performance but does not require the performance or data integrity trade-offs that are inherent in the ext3 file systems when running in ordered and writeback modes. Conclusions drawn suggest that Solaris ZFS provides the same or greater data protection as the Red Hat Enterprise Linux ext3 file system running in journalled mode. However, the ext3 file system often delivers the lowest performance, while Solaris ZFS can provide equal or greater performance than the ext3 file system operating in the ordered or writeback mode.

The following points should be taken into consideration:

- The tests were performed on system incorporating powerful AMD Opteron™ processors, a large memory configuration, and a very wide interface to an array of high-speed disks to ensure tat the fewest possible factors would inhibit file system performance. It is possible that the file system differences could be reduced on a less powerful system simply because all file systems could run into bottlenecks in moving data to the disks.
- A file system and its performance do not exist in a vacuum. The file system performs only as well as the hardware and operating system infrastructure surrounding it, such as the virtual memory subsystem, kernel threading implementation, and device drivers. As such, Sun's overall enhancements in the Solaris 10 Operating System, combined with powerful Sun servers, can provide customers with high levels of performance for applications and network services. Additionally, proof-of-concept implementations are invaluable in supporting purchasing decisions for specific configurations and applications.
- Benchmarks provide general guidance to performance. The test results presented in this document suggest that in application areas such as e-mail, netnews and electronic commerce, Solaris ZFS performs the same or better in a side by side comparison with the Red Hat Enterprise Linux ext3 file system. Proof-of-concepts and real world testing can help evaluate performance for specific applications and services.
- With low acquisition and ownership costs, integrated Sun and open source applications, and enterprise-class security, availability and scalability, the Solaris OS provides x86 users with an attractive price/performance ratio over solutions from other vendors.

For More Information

More information on Solaris ZFS can be found in the references listed in Table 4-1 below.

Table 4-1. Related Web sites

Description or Title	URL
File System Manager User's Guide	docs.sun.com (Document Number: 819-5448-10)
OpenSolaris Project	opensolaris.org
Solaris Operating System	sun.com/solaris
Solaris ZFS	sun.com/solaris
Solaris ZFS Administration Guide	docs.sun.com (Document Number: 817-2271-2)
Sun Announces Support for Postgres Database on Solaris 10	sun.com/smi/Press/sunflash/2005-11/sunflash.20051117.1.xml
Sun StorageTek QFS Configuration and Administration Guide	docs.sun.com (Document Number: 819-4332-10)

More information on the ext3 file system and related topics can be found in the following:

JFS Layout: How the Journaled File System Handles the On-Disk Layout, Steve Best and Dave Kleikamp, 2000. http://www-106.ibm.com/developerworks/library-l-jfslayout

JFS Overview: How the Journaled File System Cuts System Restart Times to the Quick, Steve Best, 2000. http://www-106.ibm.com/developerworks/library/l-jfs.html

Benchmarking File System Benchmarks, N. Joukov, A. Traeger, CP Wright, Zadok, ETechnical Report FSL-05-04b, CS Department, Stony Brook University, 2005. http://www.fsl.cs.sunysb.edu/docs/fsbench/fsbench.pdf

Postmark: A New File System Benchmark, Jeffrey Katcher, Netapps Tech Report 3022, 1997. http://www.netapp.com/tech_library/3022.html

An Overview of the Red Hat Enterprise Linux 4 Product Family, Revision 4b, February, 2005. http://www.redhat.com/f/pdf/rhel4/RHEL4OverviewWP.pdf

Operating System Design and Implementation, Andrew S. Tannenbaum and Albert S. Woohull, Prentice Hall, 1987.

EXT3, Journaling File System, Dr. Stephen Tweedie, 2000. http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html

Linux File Systems, William von Hagen, SAMS Press, 2002.

Appendix A
BenchW SQL Used During Testing

```
Appendix - Copyright for BenchW

Copyright 2002 Mark Kirkwood. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification,are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE MARK KIRKWOOD OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the software and documentation
are those of the author(s).
```

```
-- Rolled up benchw transaction scripts with timing functionality.
-- Timing info not part of original benchmark
-- Before this script run "psql -d template1 -c "CREATE DATABASE benchw"
-- Run this script as "psql -d benchw -f main.sql
-- After this script run "psql -c "DROP DATABASE benchw"

SET DATESTYLE TO 'ISO';

CREATE TABLE tasklog(taskname char(15), timebegun timestamp, timefinished timestamp);

--  benchw schema : generated by schemagen

CREATE TABLE dim0 (d0key INTEGER NOT NULL, ddate DATE NOT NULL,
dyr INTEGER NOT NULL, dmth INTEGER NOT NULL, dday INTEGER NOT NULL);

CREATE TABLE dim1 (d1key INTEGER NOT NULL, dat VARCHAR(100) NOT NULL,
dattyp VARCHAR(20) NOT NULL, dfill VARCHAR(100) NOT NULL);

CREATE TABLE dim2 (d2key INTEGER NOT NULL,dat VARCHAR(100) NOT NULL,
dattyp VARCHAR(20) NOT NULL, dfill VARCHAR(100) NOT NULL);

CREATE TABLE fact0 (d0key INTEGER NOT NULL, d1key INTEGER NOT NULL, d2key
INTEGER NOT NULL, fval INTEGER NOT NULL,ffill VARCHAR(100) NOT NULL);

COPY dim0 FROM '/dump/dim0.dat' USING DELIMITERS ',';
COPY dim1 FROM '/dump/dim1.dat' USING DELIMITERS ',';
COPY dim2 FROM '/dump/dim2.dat' USING DELIMITERS ',';

INSERT INTO tasklog (taskname, timebegun) VALUES ('MainTableLoad', 'now');
COPY fact0 FROM '/dump/fact0.dat' USING DELIMITERS ',';
UPDATE tasklog SET timefinished = 'now' WHERE taskname = 'MainTableLoad' ;

--Start: Pre-index tests : query type 0

INSERT INTO tasklog (taskname, timebegun) VALUES ('qtype0noindex', 'now');
SELECT d0.dmth, count(f.fval ) FROM dim0 AS d0, fact0 AS f WHERE
d0.d0key = f.d0key AND d0.ddate BETWEEN '2010-01-01' AND '2010-12-28'
GROUP BY d0.dmth ;
UPDATE tasklog SET timefinished = 'now' WHERE taskname = 'qtype0noindex' ;

-- : query type 1

INSERT INTO tasklog (taskname, timebegun) VALUES ('qtype1noindex', 'now');
SELECT d0.dmth,count(f.fval )FROMdim0 AS d0,fact0 AS f WHERE
d0.d0key = f.d0key AND d0.dyr = 2010
GROUP BY d0.dmth ;
UPDATE tasklog SET timefinished = 'now' WHERE taskname = 'qtype1noindex' ;
```

```
-- : query type 2

INSERT INTO tasklog (taskname, timebegun) VALUES ('qtype2noindex', 'now');
SELECT d0.dmth,d1.dat,count(f.fval )FROM dim0 AS d0, dim1 AS d1,fact0 AS f
WHERE d0.d0key = f.d0key AND d1.d1key = f.d1key AND d0.dyr BETWEEN 2010 AND 2015
AND d1.dattyp BETWEEN '10th measure type' AND '14th measure type'
GROUP BY d0.dmth, d1.dat ;
UPDATE tasklog SET timefinished = 'now' WHERE taskname = 'qtype2noindex' ;

-- : query type 3

INSERT INTO tasklog (taskname, timebegun) VALUES ('qtype3noindex', 'now');
SELECT d0.dmth, d1.dat, d2.dat, count(f.fval )FROM dim0 AS d0, dim1 AS d1,
dim2 AS d2, fact0 AS f
WHERE d0.d0key = f.d0key AND d1.d1key = f.d1key AND d2.d2key = f.d2key
AND d0.dyr BETWEEN 2010 AND 2015 AND d1.dattyp BETWEEN '10th measure type' AND '14th
measure type' AND d2.dattyp BETWEEN '1th measure type' AND '4th measure type'
GROUP BY d0.dmth, d1.dat, d2.dat ;
UPDATE tasklog SET timefinished = 'now' WHERE taskname = 'qtype3noindex' ;

-- : query type 4

INSERT INTO tasklog (taskname, timebegun) VALUES ('qtype4noindex', 'now');
SELECT d0.dyr, count(f.fval )FROM dim0 AS d0, fact0 AS f WHERE d0.d0key = f.d0key
AND d0.dyr < 2010
GROUP BY d0.dyr ;
UPDATE tasklog SET timefinished = 'now' WHERE taskname = 'qtype4noindex';

--  benchw (suggested) indexes : generated by schemagen

INSERT INTO tasklog (taskname, timebegun) VALUES ('IndexCreate', 'now');
CREATE UNIQUE INDEX dim0_d0key ON dim0(d0key) ;
CREATE UNIQUE INDEX dim1_d1key ON dim1(d1key) ;
CREATE UNIQUE INDEX dim2_d2key ON dim2(d2key) ;
CREATE INDEX fact0_d0key ON fact0(d0key) ; CREATE INDEX fact0_d1key ON fact0(d1key) ;
CREATE INDEX fact0_d2key ON fact0(d2key) ;
UPDATE tasklog SET timefinished = 'now' WHERE taskname = 'IndexCreate' ;

ANALYSE dim0; ANALYSE dim1; ANALYSE dim2; ANALYSE fact0;
-- The same queries are then run again with the benefit of indexes and analysis.

SELECT taskname, timefinished - timebegun AS timespent FROM tasklog;
```

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 1-650-960-1300 or 1-800-555-9SUN (9786) **Web** sun.com

© 2007 Sun Microsystems, Inc. All rights reserved. © 2006-2007 Sun Microsystems, Inc. Sun, Sun Microsystems, the Sun logo, Solaris, StorageTek, StorEdge, and Sun Fire are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. AMD Opteron and the AMD Opteron logo are a trademarks or registered trademarks of Advanced Micro Devices, Inc. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. Information subject to change without notice.

SunWIN #505689 Lit #STWP12873-0 6/07