SOLARIS™ ZFS AND VERITAS STORAGE FOUNDATION
FILE SYSTEM PERFORMANCE

White Paper
June 2007

Table of Contents

Executive Overview **1**

File System Overview **2**

Solaris™ ZFS 2

Simplified Storage Device and File System Administration 2

Pooled Storage and Integrated Volume Management 2

Strong Data Integrity 3

Immense Capacity 3

Veritas Storage Foundation 3

File System Concepts 4

Benchmark Tests and Results **5**

Hardware and Software Configuration 5

Filebench 5

Filebench Test Results 5

IOzone File System Benchmark 23

IOzone Test Results 23

Summary **31**

For More Information 31

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.

The Veritas Storage Foundation — consisting of the Veritas Volume Manager and Veritas File System — provides a set of integrated tools for storage management. This white paper explores the performance characteristics and differences of Solaris ZFS and the Veritas File System through a series of tests using the new Filebench benchmarking framework which reproduces the I/O patterns of applications, as well as the popular IOzone benchmark which tests specific I/O patterns.

Figure 1-1 illustrates the differences between Solaris ZFS and the Veritas File System in a number of tests. In many cases, Solaris ZFS performs better at the initial release. In some cases Solaris ZFS does not perform as well — 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 Veritas 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.

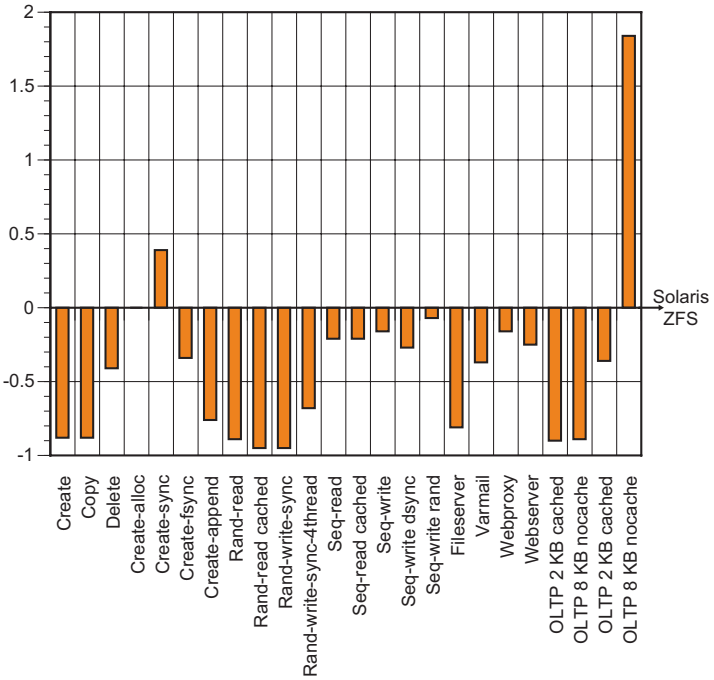

Figure 1-1. Filebench testing summary for the Veritas Storage Foundation versus Solaris ZFS

Chapter 2

File System Overview

This chapter provides a brief technical introduction to Solaris ZFS, the Veritas File System and the Veritas Volume Manager, and highlights the features that can impact performance. More information can be found in the references listed at the end of this document.

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

The Veritas Storage Foundation products make a one-to-one association between a file system and a particular storage device. Using the volume manager, 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.

Strong Data Integrity

The file system mount options in the Veritas Storage Foundation 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

More information on the Veritas File System can be found in the *Veritas Volume Manager 4.1 Administrator's Guide*, March 2005.

The maximum size of a Veritas File System is 32TB[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 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.

Veritas Storage Foundation

The Veritas Storage Foundation consists of the Veritas Volume Manager and the Veritas File System.

- Veritas Volume Manager (VxVM) is a management tool that aims to remove the physical limitations of disk storage so that it can be configured, shared, managed, and optimized for performance without interrupting data availability. The Veritas Volume Manager also provides easy-to-use, online storage management tools to help reduce planned and unplanned downtime.
- Designed for UNIX environments that need to support large amounts of data and require high performance and availability, the Veritas File System (VxFS) provides an extent-based, intent logging file system. It also provides online management capabilities that facilitate the creation and maintenance of file systems.

File System Concepts

The Veritas File System presents physical storage to the application via several layers of abstraction.

- *Physical disk.*A physical disk is the basic storage device (media) upon which data is stored.
- *VM disk.* When a physical disk is placed under Veritas Volume Manager control, a VM disk is assigned to the physical disk. A VM disk is under Veritas Volume Manager control, and is usually assigned to a disk group.
- *Subdisk.* A VM disk can be divided into one or more subdisks. Each subdisk represents a specific portion of a VM disk, and consists of contiguous disk blocks.
- *Plex.* Veritas Volume Manager uses subdisks to build virtual objects, called plexes. A plex consists of one or more subdisks located on one or more physical disks.
- *Disk group.* A disk group is a collection of disks that share a common configuration and are managed by Veritas Volume Manager.
- *Volume.* A volume is a virtual disk device that appears like a physical disk device. A volume consists of one or more plexes contained in a disk group that each hold a copy of the selected data in the volume.
- *File system.* A Veritas File System is constructed on a volume so that files can be stored.

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.

More information can be found in the *Veritas Volume Manager 4.1 Administrator's Guide Solaris*, March 2005. Additional information on Solaris ZFS can be found in the *Solaris ZFS Administration Guide* located at docs.sun.com, document number 817-2271-2.

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

Table 2-1. The file system creation procedure for Solaris ZFS and the Veritas File System

Solaris ZFS	Veritas File System
# **zpool create -f tank** [48 disks] # **zfs create tank/fs**	# **/usr/lib/vxvm/bin/vxdisksetup -i c2t16d0** # **vxdg init dom -dg c2t16d0** # **for i in** [list of 48 disks] **do** **/usr/lib/vxvm/bin/vxdisksetup -i $i** **done** # **for i in** [list of 48 disks] **do** **vxdg -g dom -dg adddisk $i** **done** # **vxassist -g dom -dg -p maxsize layout=stripe** 6594385920 [get size of volume] # **vxassist -g dom -dg make dom -vol 6594385920** **layout=stripe** [feed volume size back in] # **mkfs -F vxfs /dev/vx/rdsk/dom -dg/dom -vol** version 6 layout 6594385920 sectors, 412149120 blocks of size 8192, log size 32768 blocks largefiles supported # **mount -F vxfs /dev/vx/dsk/dom -dg/dom -vol /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.

Hardware and Software Configuration

Table 3-1 describes the hardware, operating system, and software used to create the platforms tested and perform benchmark testing.

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

Servers	
Sun Fire E4900 server with 24 1.5 GHz UltraSPARC® IV+ processors and 98 GB RAM	
Operating System	
Solaris 10 OS Update 2 06/06 (Including Solaris ZFS)	
Storage	
Four Sun StorageTek 3500 arrays, each with 48 x 72 GB disks, Fiber channel interface, 4 x 2 Gb PCI-X 133 MHz	
Software	
Veritas Volume Manager 4.1	Veritas File System 4.1
Filebench 1.64.5	IOzone 3.263

Filebench

Filebench is a benchmarking framework for simulating the effect of applications on file systems. Application workloads are configured as definitions in the Filebench "f" scripting language. Simulations run as several multithreaded processes. Combined with the use of interprocess communication primitives such as POSIX semaphores, and "f" language constructs to manipulate the I/O environment such as `O_SYNC`, these features enable the emulation of complex relational database applications and database benchmarks such as TPC1 and Web applications. Integrated statistics provide for microsecond accurate latency and cycle counts per system call. Several popular file system benchmarks, such as Mongo and Postmark, are emulated in Filebench and are supplied as scripts.

Filebench Test Results

The following sections present the tests run, configuration used, and results obtained using the Filebench tool.

Create and Delete

File creation and deletion is a metadata intensive activity which is key to many applications, such as Web-based commerce and software development. Table 3-2 describes the workload, configuration, and parameters used during testing efforts.

Table 3-2. Create and delete workloads, configurations, and parameters

Personality	Workload	Variables
createfiles	createfiles	nfiles 100000, dirwidth 20, filesize 16k, nthreads 16
deletefiles	deletefiles	nfiles 100000, meandirwidth 20, filesize 16k, nthreads 16

Figure 3-1 shows the number of operations per second obtained during testing efforts.

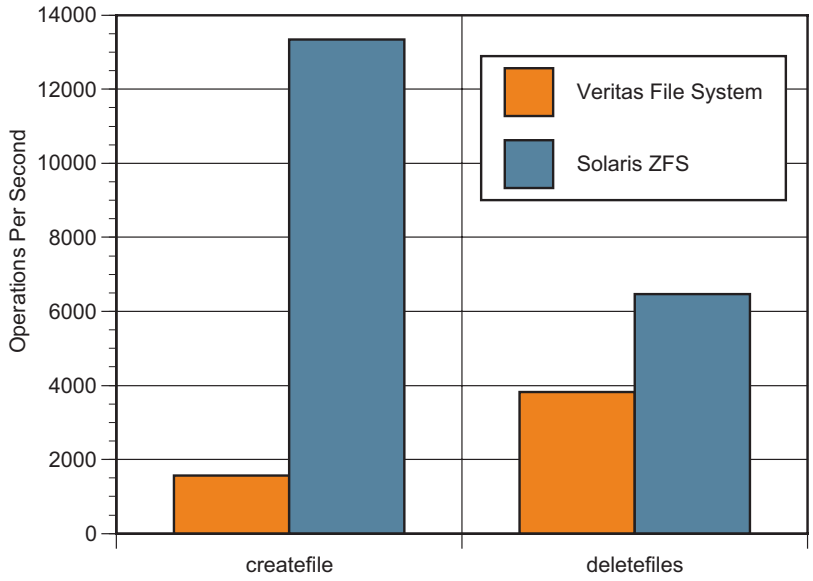

Figure 3-1. Operations per second obtained during create and delete testing

Figure 3-2 shows the CPU time used per operation during testing efforts.

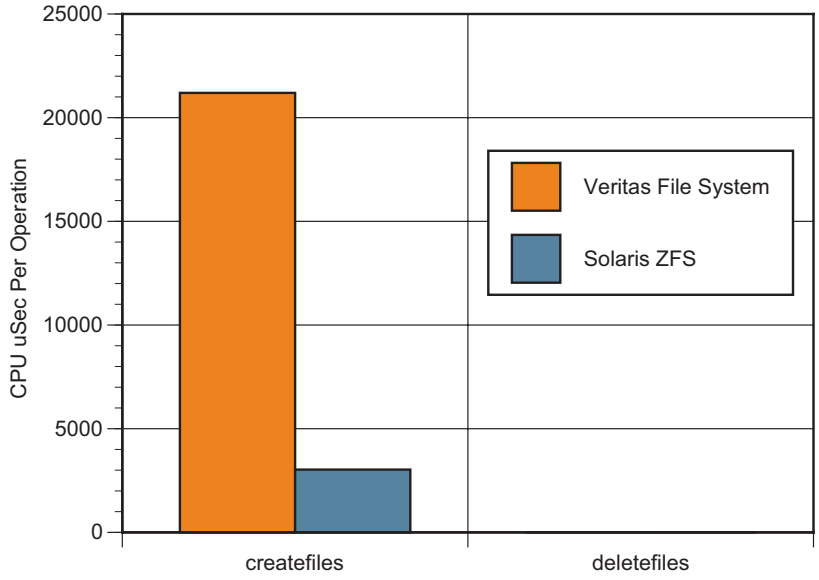

Figure 3-2. CPU time used during create and delete testing

Figure 3-3 shows the latency observed during testing efforts.

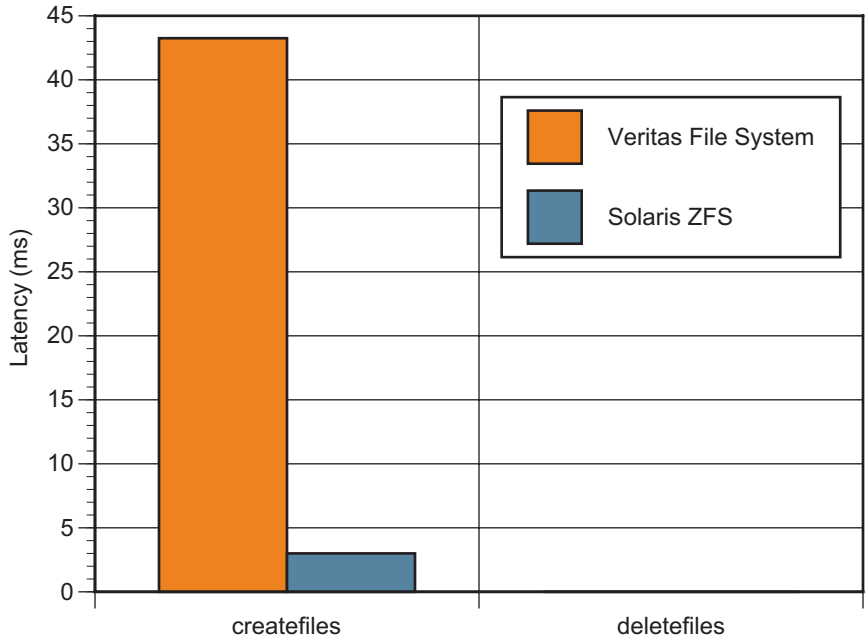

Figure 3-3. Latency observed during create and delete testing

Copy Files

The copyfiles test creates two large directory tree structures and measures the rate at which files can be copied from one tree to the other. Testing efforts used the following copyfile personality and workload, and several parameters (nfiles 100000, dirwidth 20, filesize 16k nthreads 16).

Figure 3-4 shows the number of operations per second obtained during testing efforts.

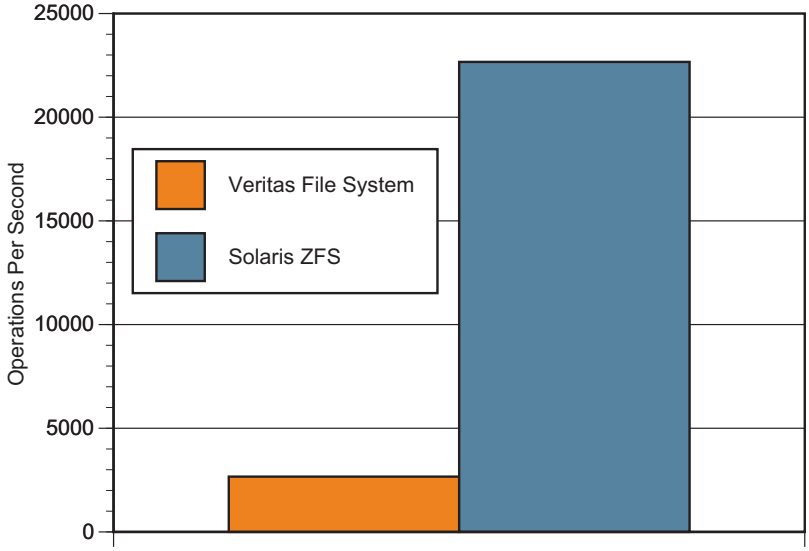

Figure 3-4. Operations per second obtained during copy files testing

Figure 3-5 shows the CPU time used per operation during testing efforts.

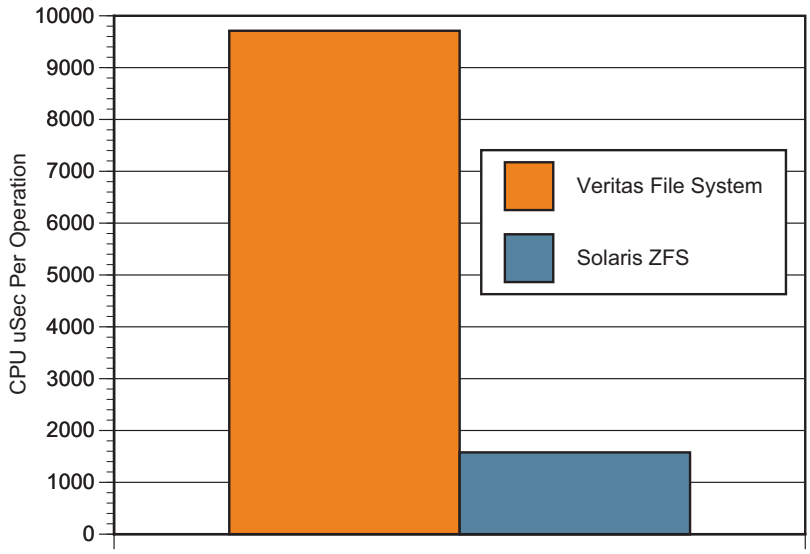

Figure 3-5. CPU time used during copy files testing

Figure 3-6 shows the latency observed during testing efforts.

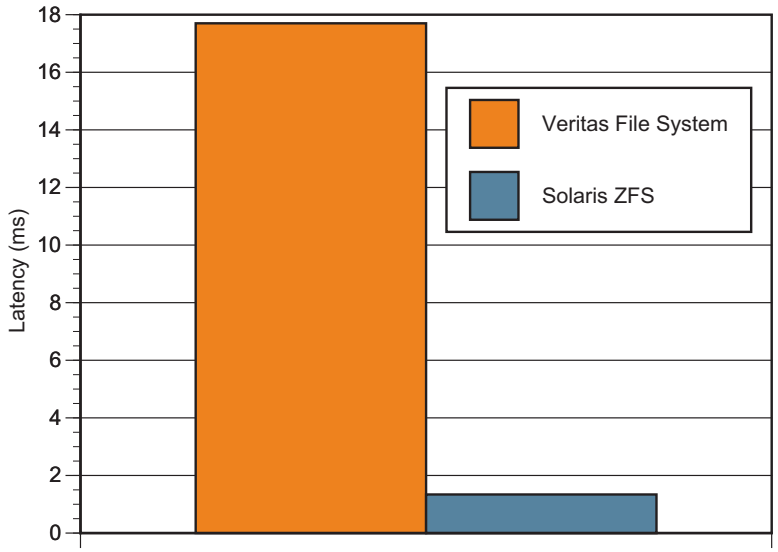

Figure 3-6. Latency observed during copy files testing

File Creation

The file creation test creates a directory tree and populates it with files of specified sizes. The file sizes are chosen according to a gamma distribution of 1.5, with a mean size of 16 KB. The different workloads used are designed to test different types of I/O, such as open, sync, fsync and more. Information on I/O can be found in the Solaris OS `open(2)`, `sync(2)` and `fsync(3C)` man pages.

Table 3-3 describes the workload, configuration, and parameters used during file creation testing efforts.

Table 3-3. File creation workloads, configurations, and parameters

Personality	Workload	Variables
filemicro_create	createandalloc	nfiles 100000, nthreads 1, iosize 1m, count 64
	createallocsync	nthreads 1, iosize 1m, count 1k, sync 1
filemicro_writefsync	createallocfsync	nthreads 1
filemicro_createrand	createallocappend	nthreads 1

Figure 3-7 shows the number of operations per second obtained during file creation testing efforts.

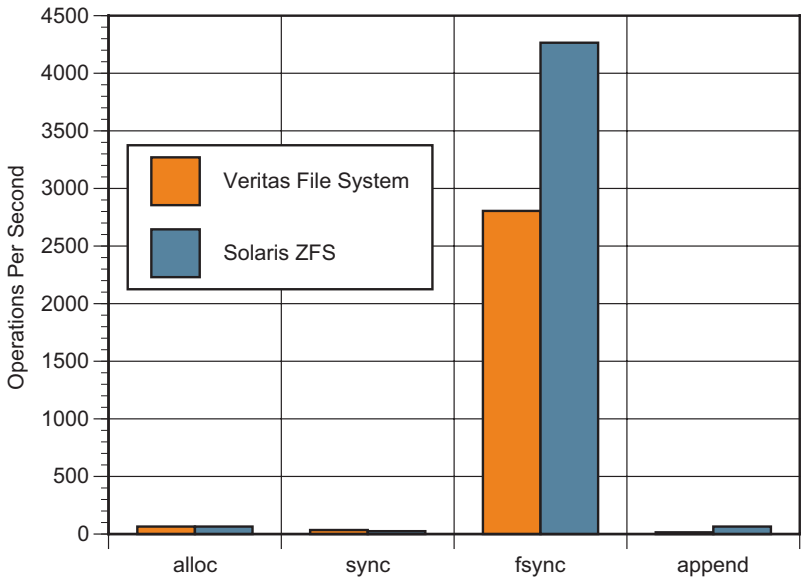

Figure 3-7. Operations per second obtained during file creation testing

Figure 3-8 shows the CPU time used per operation during file creation testing efforts.

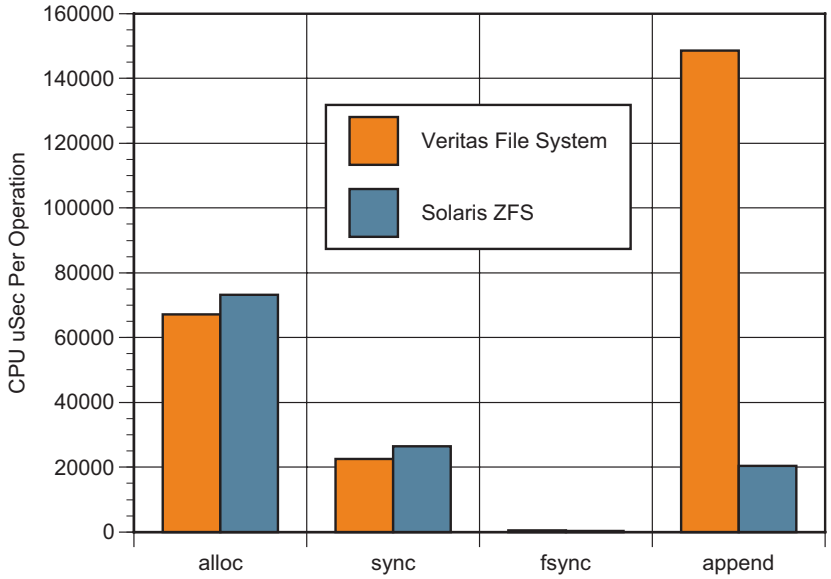

Figure 3-8. CPU time used during file creation testing

Figure 3-9 shows the latency observed during file creation testing efforts.

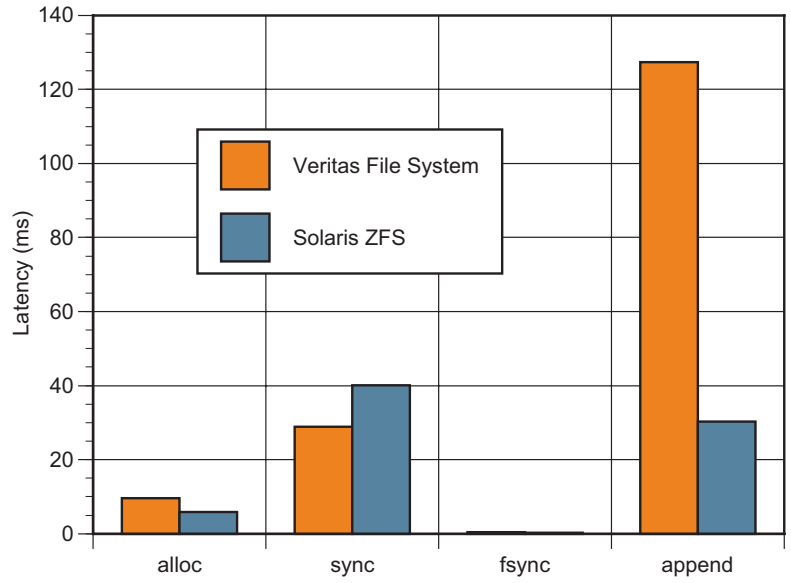

Figure 3-9. Latency observed during file creation testing

Random Reads

The random read test performs single-threaded, 2 KB random reads from a 5 GB file. Table 3-4 describes the workload, configuration, and parameters used during random read testing efforts.

Table 3-4. Random read workloads, configurations, and parameters

Personality	Workload	Variables
filemicro_rread	randread2k	cached 0, iosize 2k
	randread2kcached	cached 1, iosize 2k

Figure 3-10 shows the number of operations per second obtained during random read testing efforts.

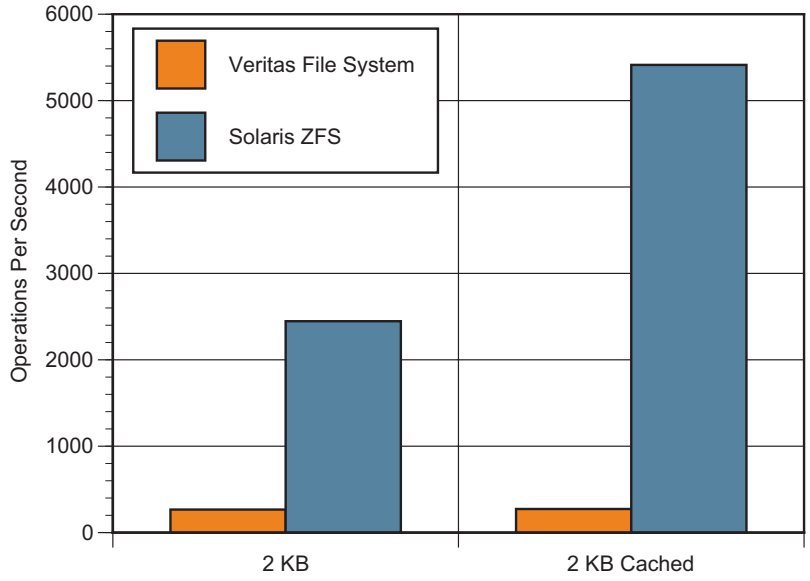

Figure 3-10. Operations per second obtained during random read testing

Figure 3-11 shows the CPU time used per operation during random read testing efforts.

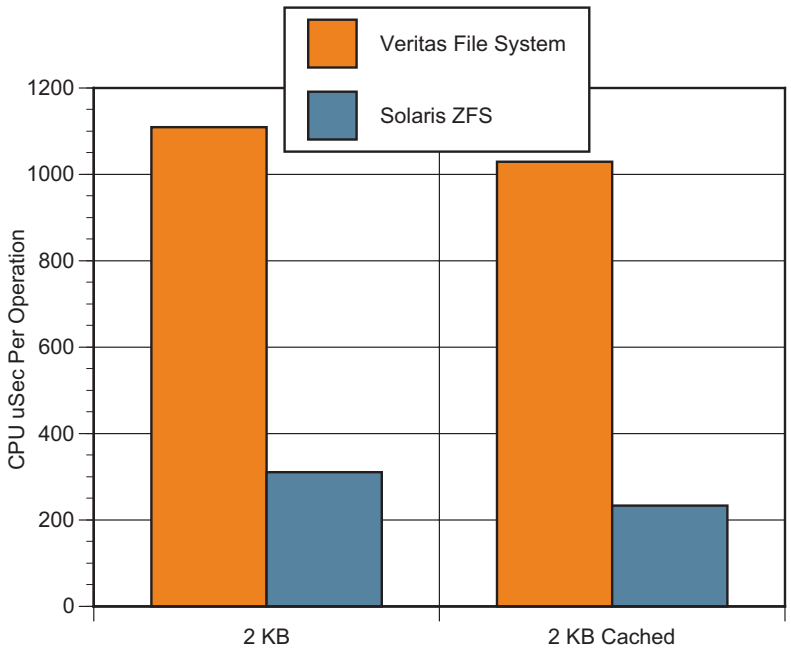

Figure 3-11. CPU time used during random read testing

Figure 3-12 shows the latency observed during random read testing efforts.

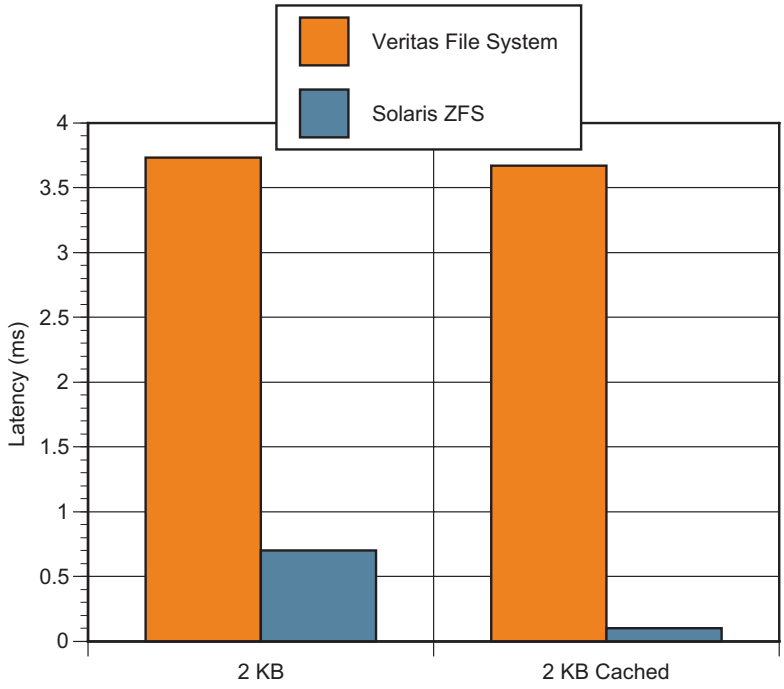

Figure 3-12. Latency observed during random read testing

Random Write

The random write test consists of multithreaded writes to a single 5 GB file. Table 3-5 describes the workload, configuration, and parameters used during random write testing efforts.

Table 3-5. Random write workloads, configurations, and parameters

Personality	Workload	Variables
filemicro_rwrite	randwrite2ksync	cached 1, iosize 2k
	randwrite2ksync4thread	iosize 2k, nthreads 4, sync1

Figure 3-10 shows the number of operations per second obtained during random write testing efforts.

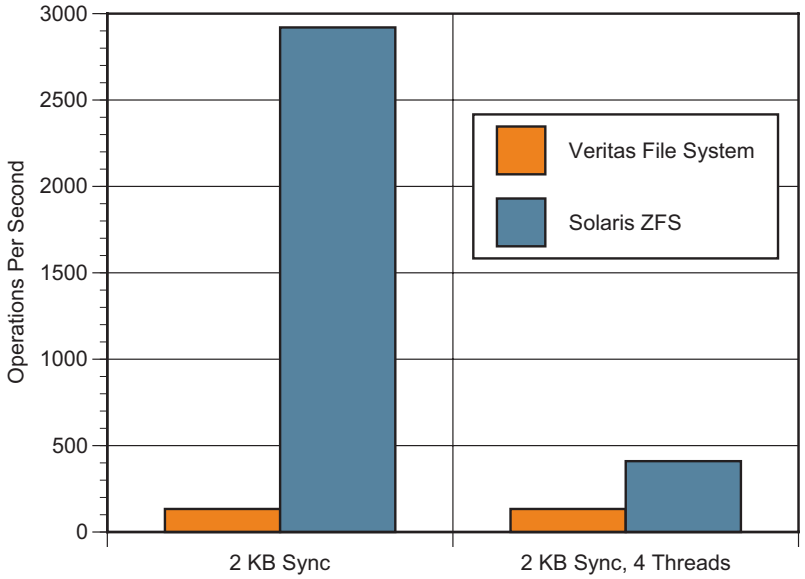

Figure 3-13. Operations per second obtained during random write testing

Figure 3-14 shows the CPU time used per operation during random write testing efforts.

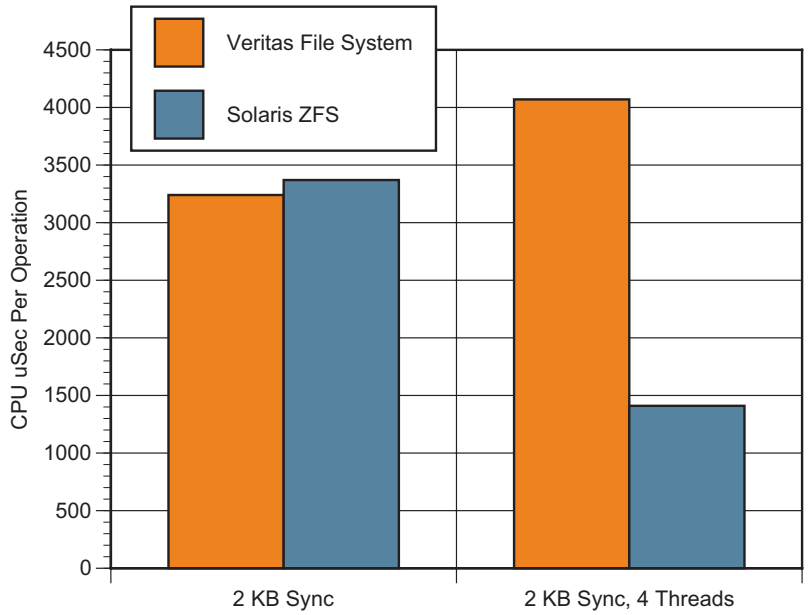

Figure 3-14. CPU time used during random write testing

Figure 3-15 shows the latency observed during random write testing efforts.

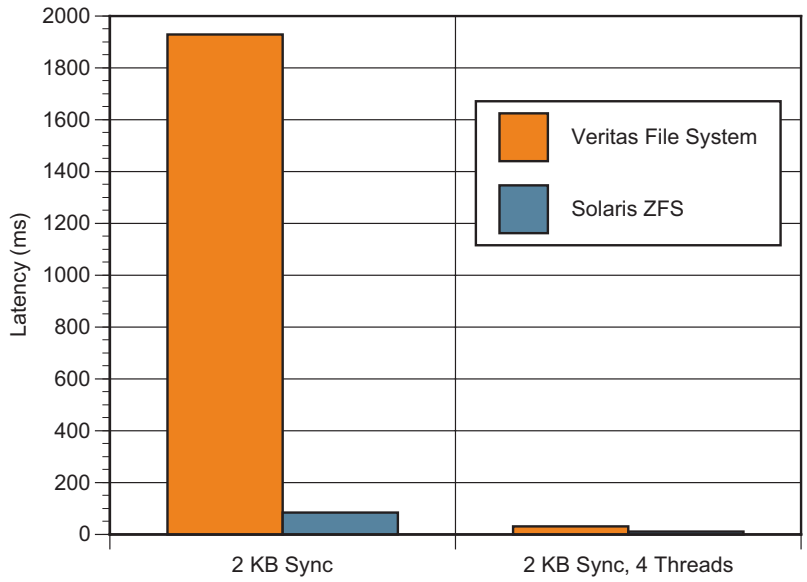

Figure 3-15. Latency observed during random write testing

Sequential Read

The sequential read test performs a single threaded read operation from a 5 GB file. Table 3-6 describes the workload, configuration, and parameters used during sequential read testing efforts.

Table 3-6. Sequential read workloads, configurations, and parameters

Personality	Workload	Variables
filemicro_seqread	seqread32k	iosize 32k, nthreads 1, cached 0, filesize 5g
	seqread32kcached	iosize 32k, nthreads 1, cached 1, filesize 5g

Figure 3-16 shows the number of operations per second obtained during sequential read testing efforts.

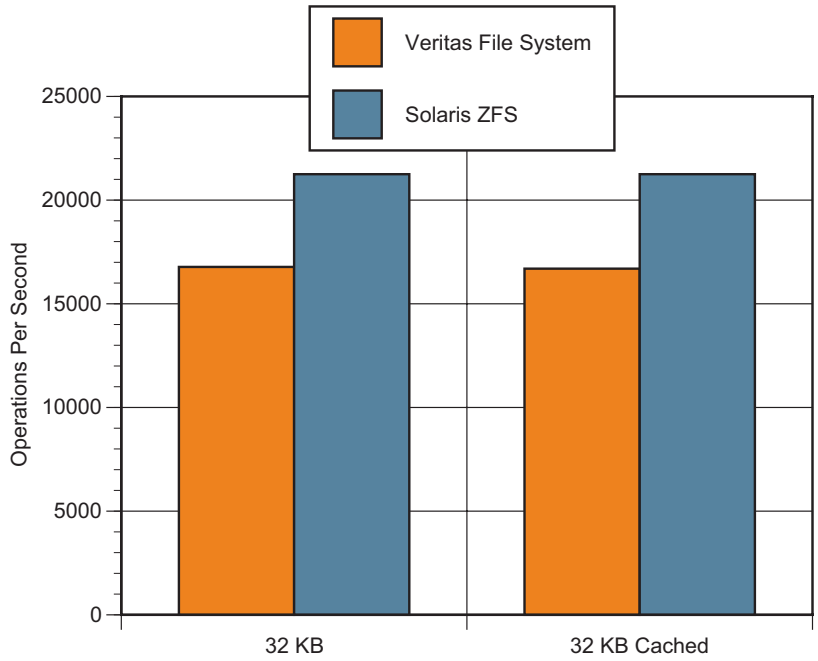

Figure 3-16. Operations per second obtained during sequential read testing

Figure 3-17 shows the CPU time used per operation during sequential read testing efforts.

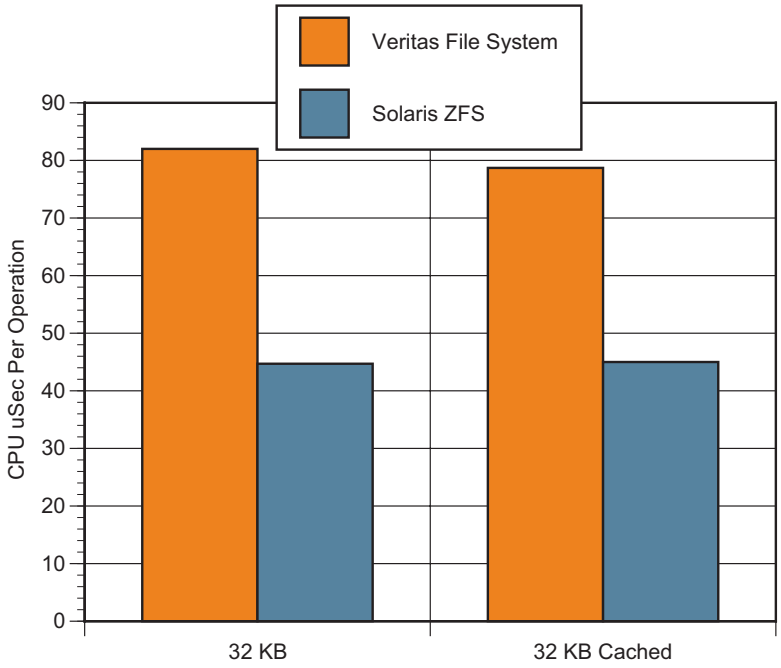

Figure 3-17. CPU time used during sequential read testing

Figure 3-18 shows the latency observed during sequential read testing efforts.

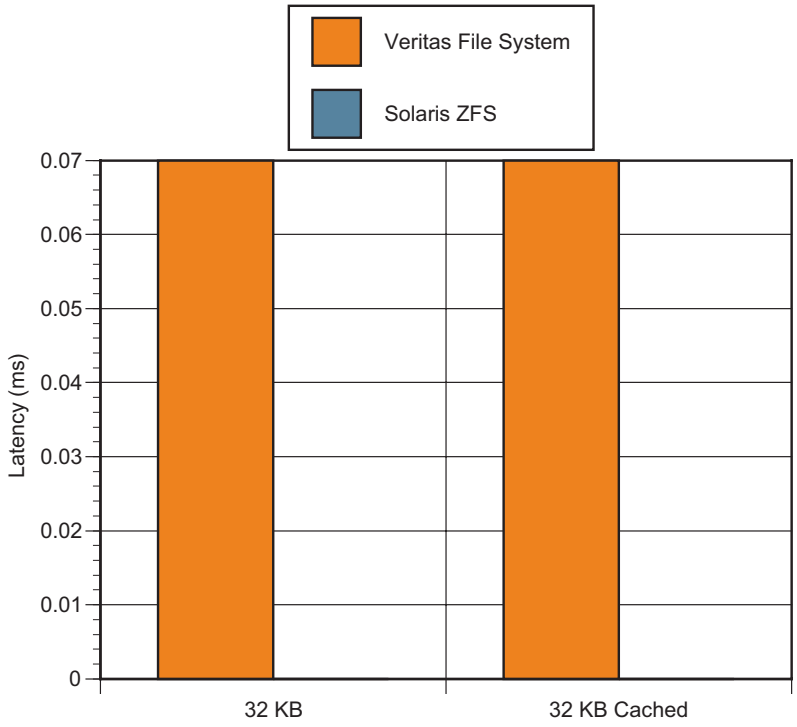

Figure 3-18. Latency observed during sequential read testing

Sequential Write

The sequential write test performs single threaded write operations to a 5 GB file. Table 3-7 describes the workload, configuration, and parameters used during sequential write testing efforts.

Table 3-7. Sequential write workloads, configurations, and parameters

Personality	Workload	Variables
filemicro_seqwrite	seqwrite32k	iosize 32k, count 32k, nthreads 1, cached 0, sync 0
	seqwrite32kdsync	iosize 32k, count 32k, nthreads 1, cached 0, sync 0
filemicro_seqwriterand	seqread32kcache	iosize 8k, count 128k, nthreads 1, cached 0, sync 0

Figure 3-19 shows the number of operations per second obtained during sequential write testing efforts.

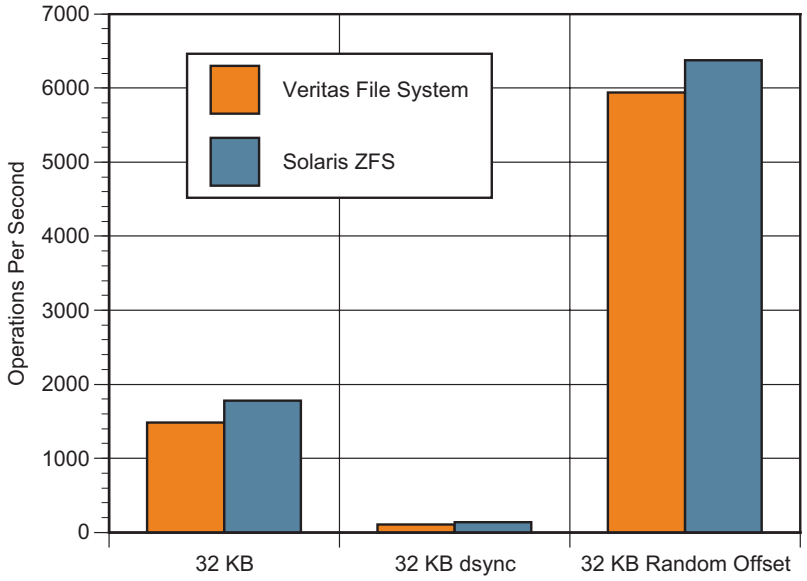

Figure 3-19. Operations per second obtained during sequential write testing

Figure 3-20 shows the CPU time used per operation during sequential write testing efforts.

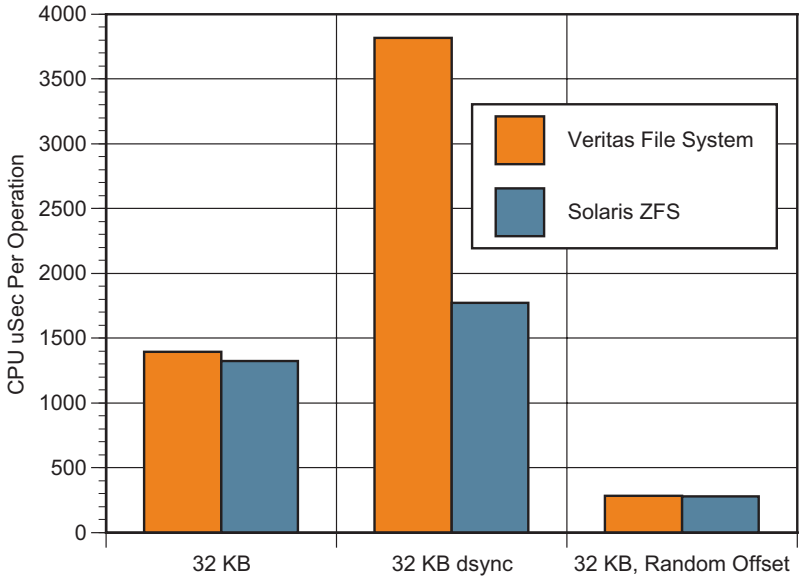

Figure 3-20. CPU time used during sequential write testing

Figure 3-21 shows the latency observed during sequential write testing efforts.

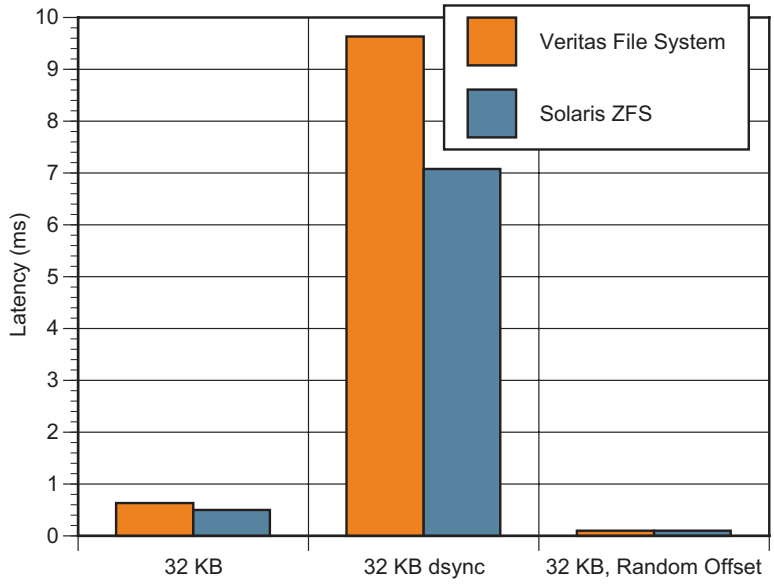

Figure 3-21. Latency observed during sequential write testing

Application Simulation

Filebench includes several scripts that emulate the behavior of applications.

Information on the SPECsfs benchmark can be found at http://spec.org. SPECsfs is a registered trademark of the Standard Performance Evaluation Corporation (SPEC).

More information on the Postmark benchmark can be found in *Postmark: A New File System Benchmark* located at http://netapp.com/tech_library_3022.html

- *Fileserver*
  The Fileserver script emulates file system workloads. Similar to the SPECsfs® benchmark, the Fileserver workload performs a series of creates, deletes, appends, reads, writes, and attribute operations on a file system. A configurable, hierarchical directory structure is used for the file set.
- *Varmail*
  The Varmail script emulates the Network File System (NFS) mail server, `/var/mail`, found in UNIX environments. Similar to Postmark benchmark workloads — but multithreaded — the Varmail script consists of a multithreaded set of open/read/close, open/append/delete, and delete operations in a single directory.
- *Web proxy*
  The Web proxy script performs a mix of create/write/close and open/read/close operations, as well as the deletion of multiple files in a directory tree, and file append operations that simulate Web proxy log operation. The script uses 100 threads, and 16 KB of data is appended to the log for every 10 read and write operations.
- *Web server*
  The Web server script performs a mix of open/read/close operations on multiple files in a directory tree, as well as file append operations that simulates Web server log operation. The script appends 16 KB of data to the Web log for every 10 reads performed.

Table 3-8 describes the workload, configuration, and parameters used during application emulation testing efforts.

Table 3-8. Application emulation workloads, configurations, and parameters

Personality	Workload	Variables
fileserver	fileserver	nfiles 100000, meandirwidth 20, filesize 2k, nthreads 100, meaniosize 16k
varmail	varmail	nfiles 100000, meandirwidth 1000000, filesize 1k, nthreads 16, meaniosize 16k
webproxy	webproxy	nfiles 100000, meandirwidth 1000000, filesize 1k, nthreads 100, meaniosize 16k
webserver	webserver	nfiles 100000, meandirwidth 20, filesize 1k, nthreads 100

Figure 3-22 shows the number of operations per second obtained during application emulation testing efforts.

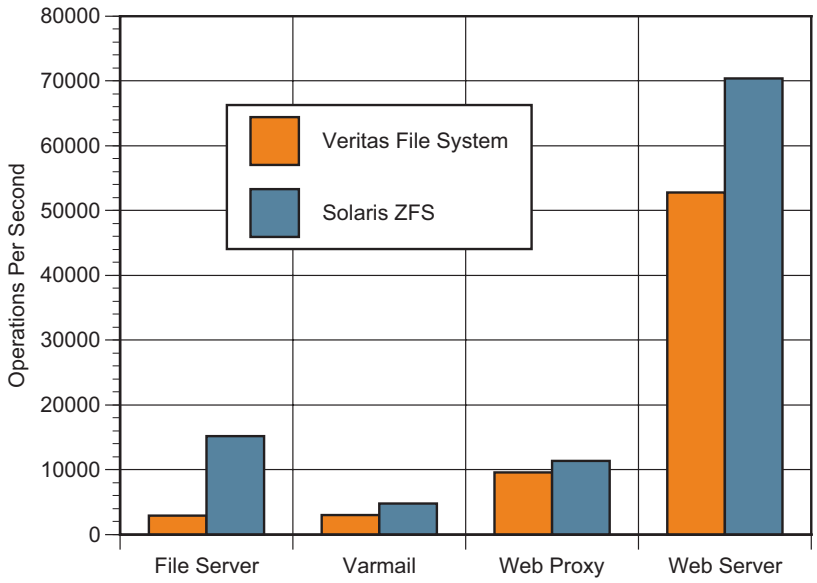

Figure 3-22. Operations per second obtained during application emulation testing

Figure 3-23 shows the CPU time used per operation during application emulation testing efforts.

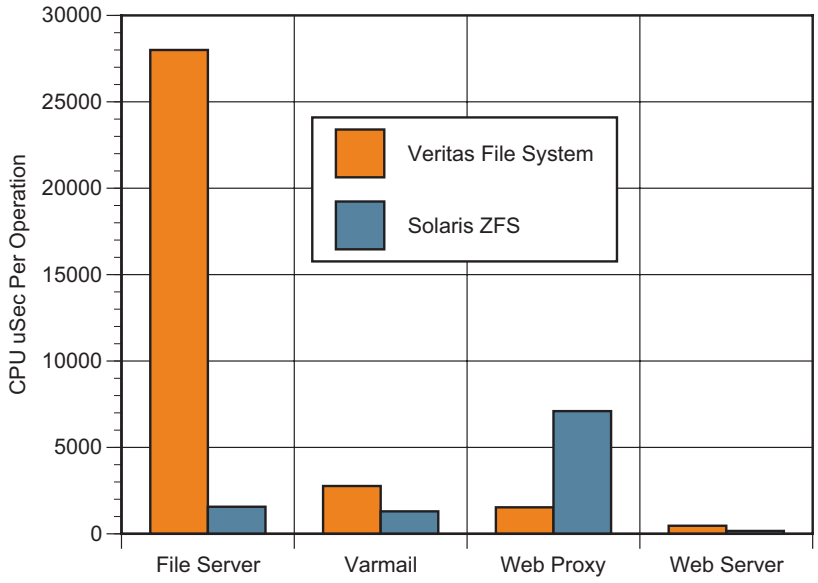

Figure 3-23. CPU time used during application emulation testing

Figure 3-24 shows the latency observed during application emulation testing efforts.

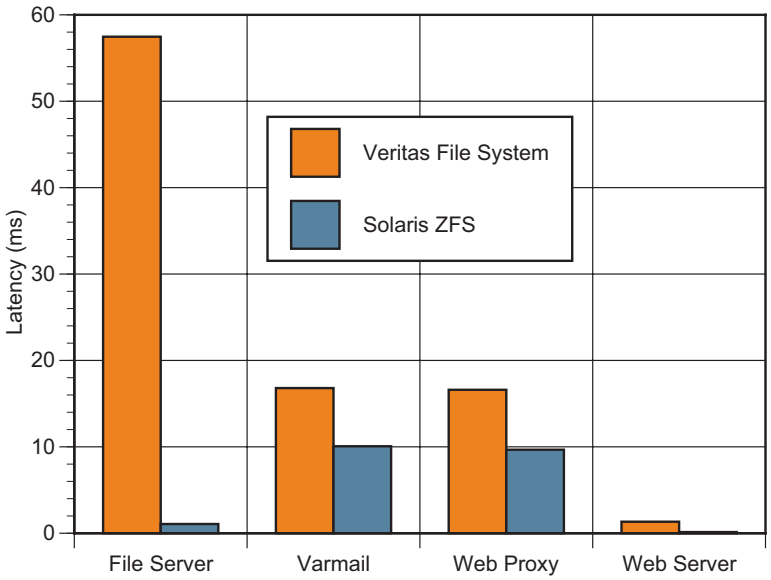

Figure 3-24. Latency observed during application emulation testing

Online Transaction Processing Database Simulation

Filebench includes tests that emulate database operations. The test performs transactions on a file system using the I/O model used in Oracle 9i Database. The

workload tests the performance of small random read and write operations, and is sensitive to the latency of moderate (128 KB plus) synchronous write operations that are typical of a database log file. The test launches 200 reader processes, 10 asynchronous write processes, and a log writer process. Intimate shared memory (ISM) is used similarly to Oracle Database, and is critical to I/O efficiency (`as_lock` optimizations).

Table 3-9 describes the workload, configuration, and parameters used during online transaction processing (OLTP) database emulation testing efforts.

Table 3-9. OLTP emulation workloads, configurations, and parameters

Personality	Workload	Variables
oltp	large_db_oltp_2k_cached	cached 1, directio 0, iosize 2k, nshadows 200, ndbwriters 10, usermode 20000, filesize 5g, memperthread 1m, workingset 0
oltp	large_db_oltp_2k_uncached	cached 0, directio 1, iosize 2k, nshadows 200, ndbwriters 10, usermode 20000, filesize 5g, memperthread 1m, workingset 0
oltp	large_db_oltp_8k_cached	cached 1, directio 0, iosize 8k, nshadows 200, ndbwriters 10, usermode 20000, filesize 5g, memperthread 1m, workingset 0
oltp	large_db_oltp_8k_uncached	cached 0, directio 1, iosize 8k, nshadows 200, ndbwriters 10, usermode 20000, filesize 5g, memperthread 1m, workingset 0

Figure 3-25 shows the number of operations per second obtained during OLTP database emulation testing efforts.

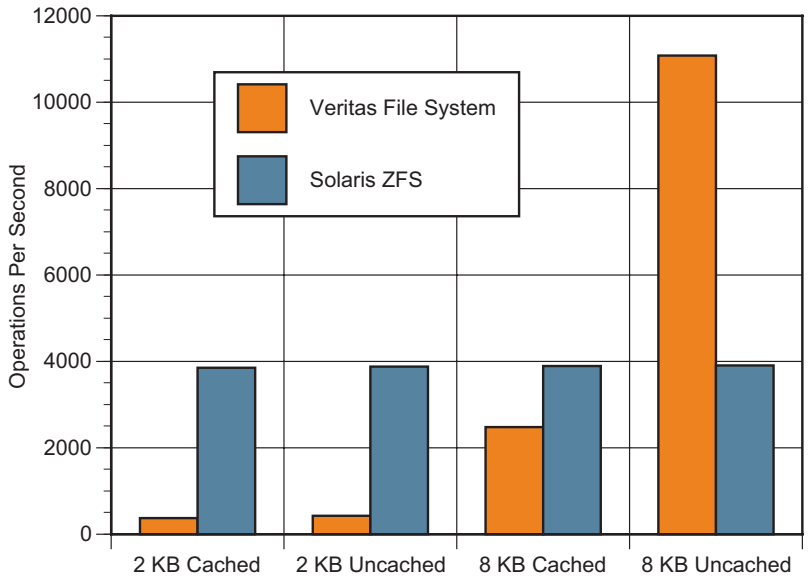

Figure 3-25. Operations per second obtained during OLTP database emulation testing

Figure 3-26 shows the CPU time used per operation during OLTP database emulation testing efforts.

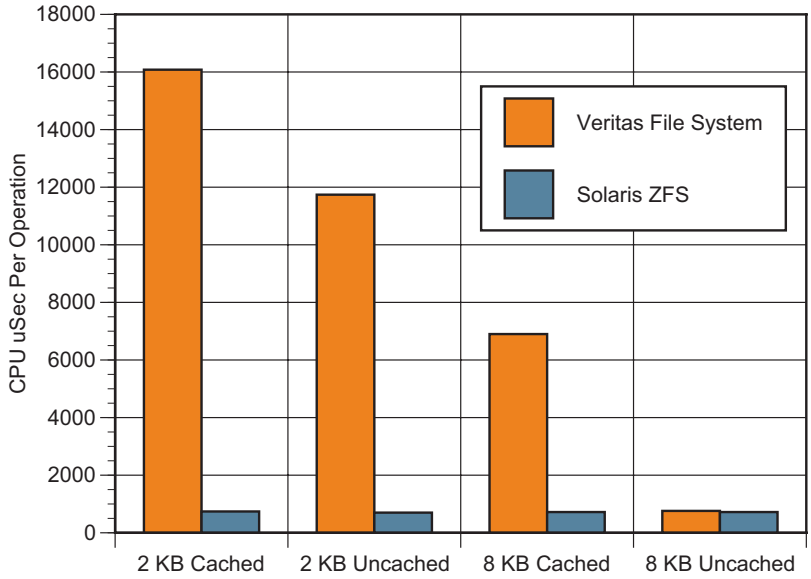

Figure 3-26. CPU time used during OLTP database emulation testing

Figure 3-27 shows the latency observed during OLTP database emulation testing efforts.

Figure 3-27. Latency observed during OLTP database emulation testing

IOzone File System Benchmark

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.

The IOzone command line used during testing efforts included the following arguments and options.

```
iozone -R -a -z -b file.wks -g 4G -f testile
```

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. Figure 3-28 shows the results obtained during IOzone write testing.

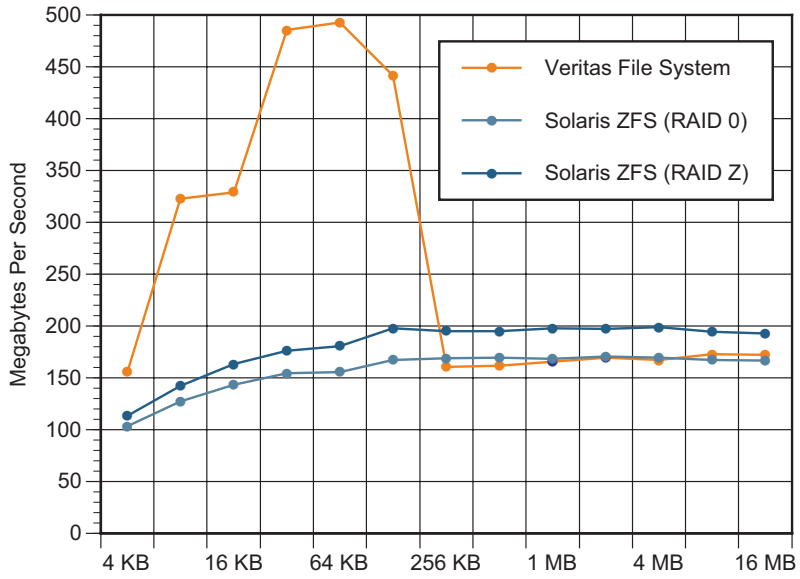

Figure 3-28. IOzone write test results

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 work as the metadata already exists. Typically, rewrite performance is higher than the performance of writing a new file. Figure 3-29 details the IOzone rewrite results obtained during testing efforts.

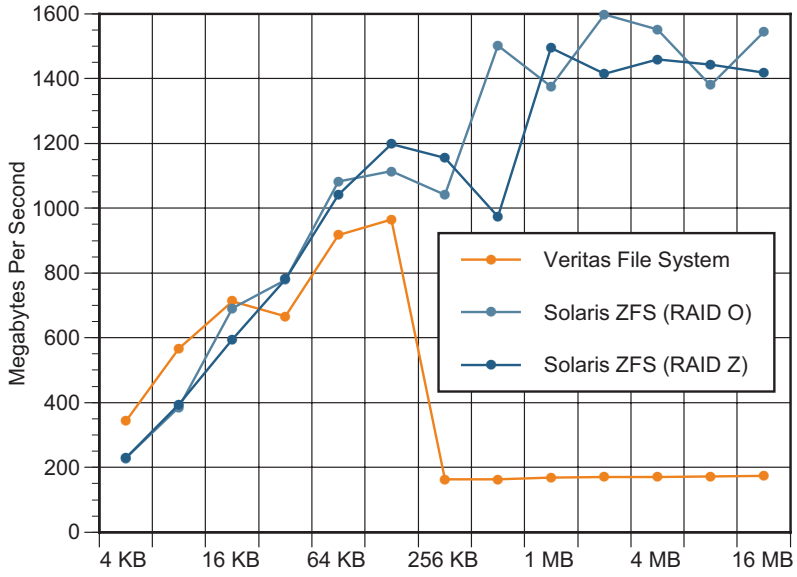

Figure 3-29. IOzone rewrite test results

IOzone Read

The IOzone read test measures the performance of reading an existing file. Figure 3-30 shows the IOzone read results obtained during testing efforts.

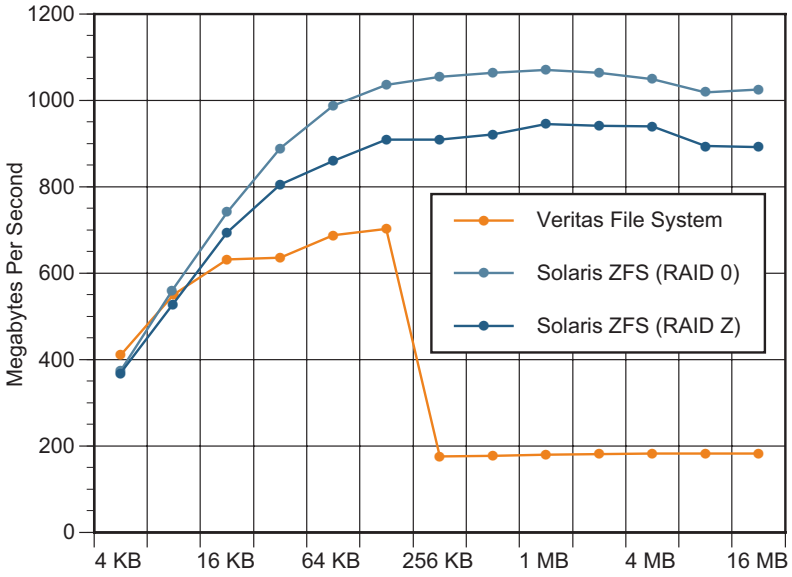

Figure 3-30. IOzone read test results

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 3-31 shows the results of the IOzone re-read test.

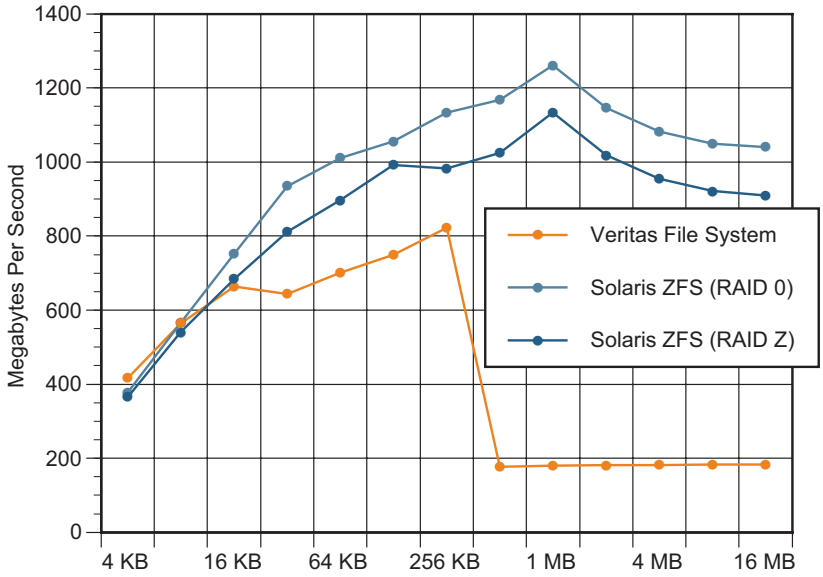

Figure 3-31. IOzone re-read test results

IOzone Record Rewrite

The IOzone record rewrite test measures the performance of writing and re-writing a section of a file. Figure 3-32 illustrates the results obtained during testing efforts.

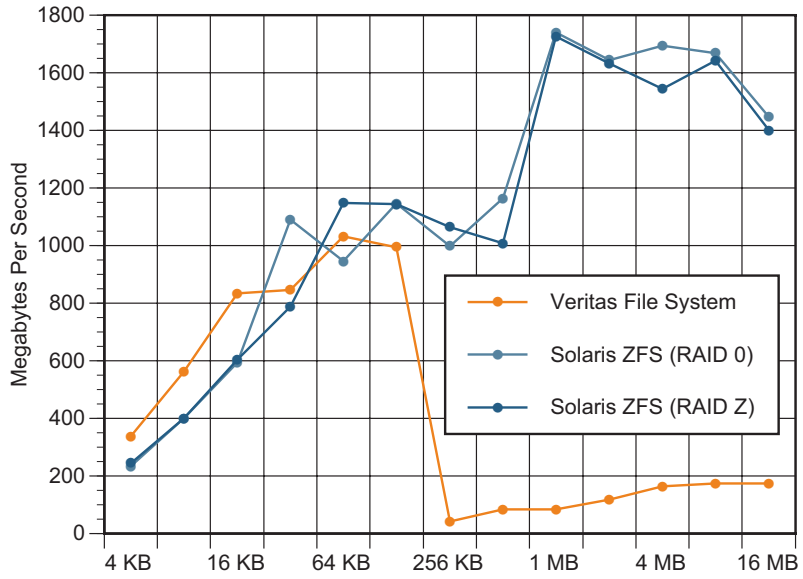

Figure 3-32. IOzone record rewrite test results

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. Figure 3-33 illustrates the results obtained during testing efforts.

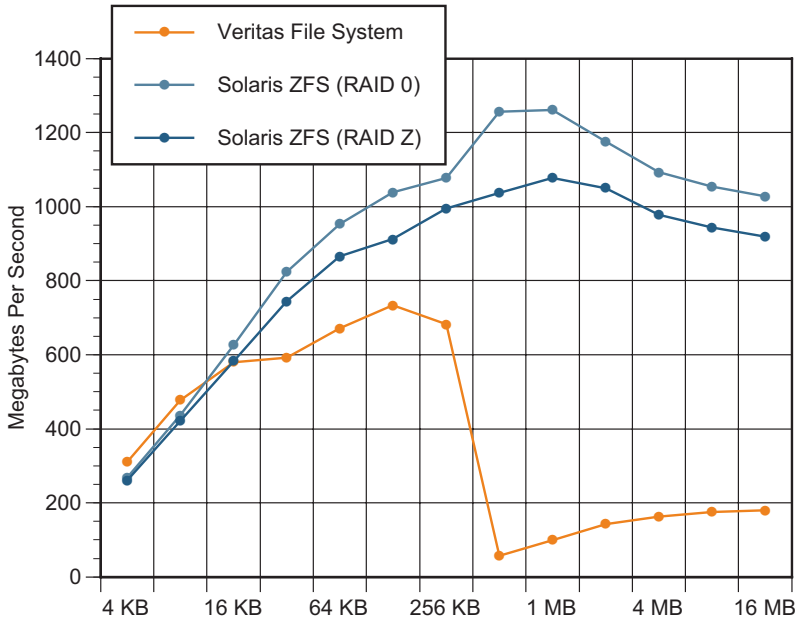

Figure 3-33. IOzone random read test results

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 3-34 depicts the results of the IOzone random write test.

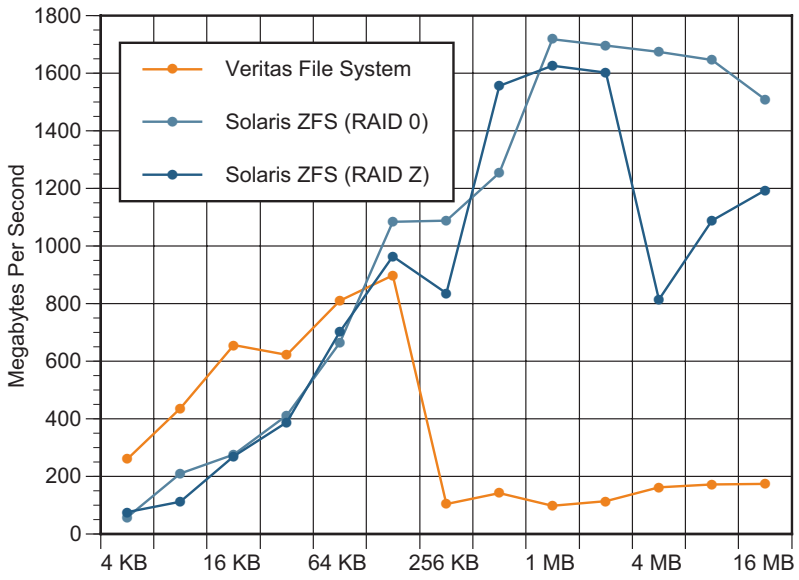

Figure 3-34. IOzone random write test results

IOzone Backward Read

The IOzone backward 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 3-35 shows the results obtained during testing efforts.

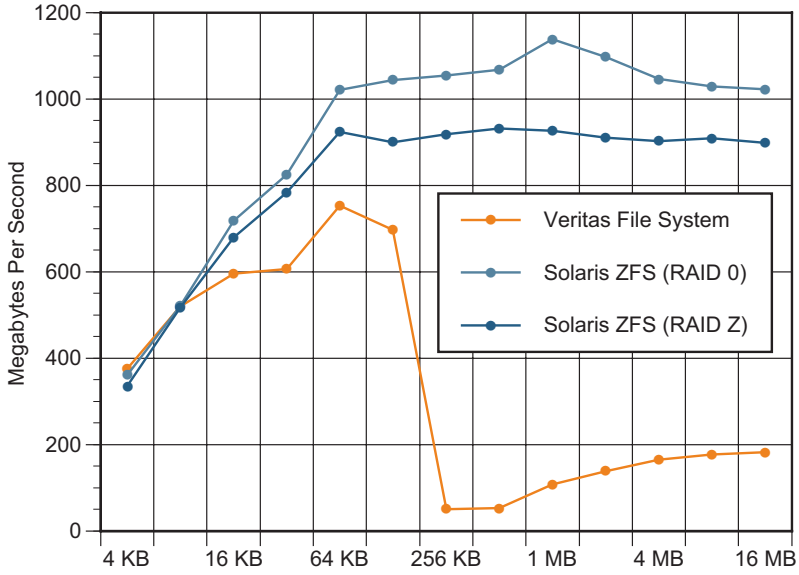

Figure 3-35. IOzone backwards read test results

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 3-36 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 pattern 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.

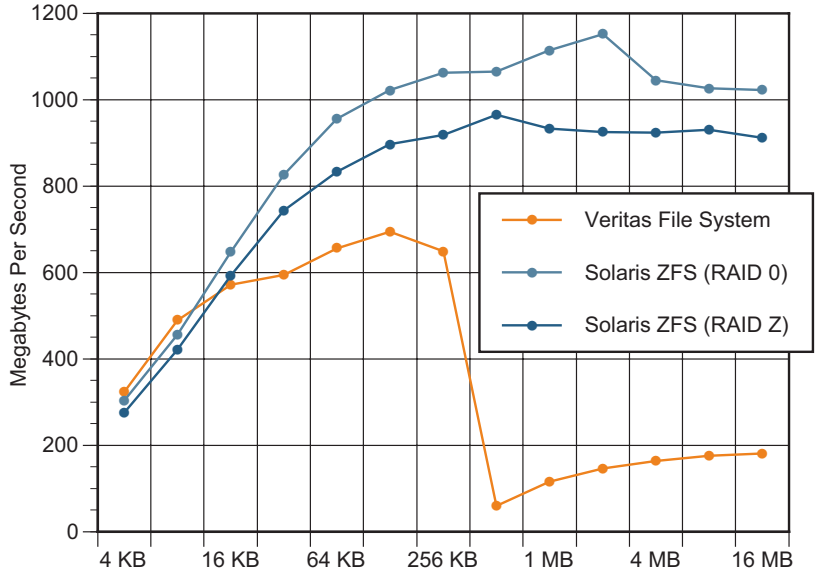

Figure 3-36. IOzone strided read test results

IOzone fwrite

The IOzone fwrite test 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 3-37 shows the test results obtained. Note the test writes a new file so metadata overhead is included in the measurement.

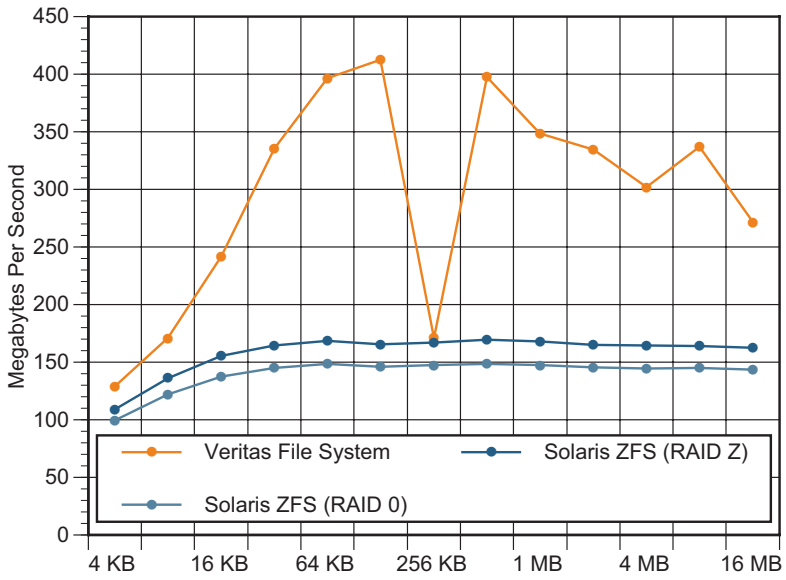

Figure 3-37. IOzone fwrite test results

IOzone Re-fwrite

The IOzone re-fwrite test performs repetitive rewrites to portions of an existing file using the `fwrite()` interface. Figure 3-38 illustrates the results obtained.

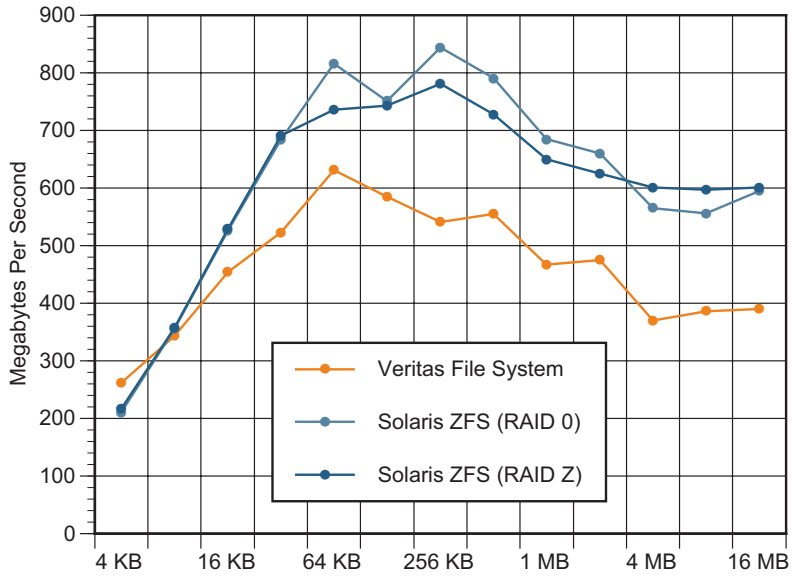

Figure 3-38. IOzone re-fwrite test results

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 3-39 shows the results obtained.

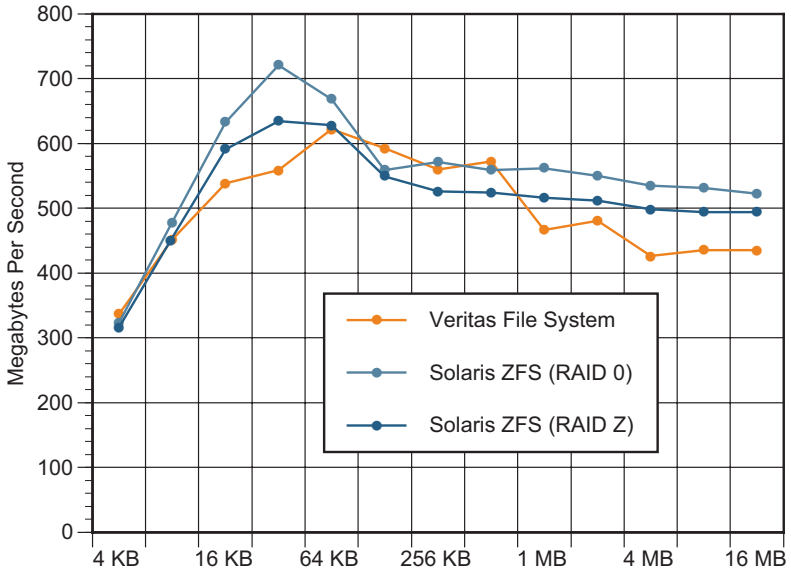

Figure 3-39. IOzone fread test results

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.

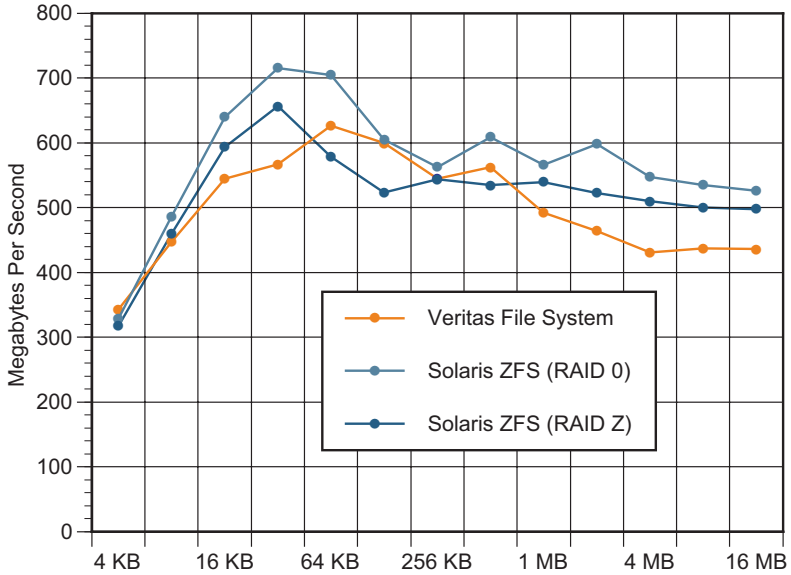

Figure 3-40. IOzone re-fread test results

Chapter 4
Summary

This white paper details the performance characteristics of Solaris ZFS and the Veritas Storage Foundation through Filebench and IOzone benchmark testing. For the conditions tested, Solaris ZFS can outperform the combination of the Veritas Volume Manager and Veritas File System for many workloads. The following points should be taken into consideration:

- The tests were performed on a Sun Fire system incorporating powerful 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 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 databases, e-mail, Web server and software development, Solaris ZFS performs better in a side by side comparison with the Veritas Storage Foundation. 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 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
Solaris Operating System	sun.com/solaris
Solaris ZFS	sun.com/solaris
Solaris ZFS Administration Guide	docs.sun.com (Document Number: 817-2271-2)

More information on the Veritas Storage Foundation and related topics can be found in the following:

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

Veritas Volume Manager 4.1 Administrator's Guide Solaris, March 2005, N13110F.

Veritas File System 4.1 Administrator's Guide Solaris, March 2005, N13073F.

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

SunWIN #505690 Lit #STWP12872-0 6/07