VERITAS Software Corporation

VERITAS File System Performance White Paper

VERITAS Software Corporation · 1600 Plymouth Street · Mountain View, CA 94043 · 1.800.258.8649 in the USA · 415.335.8000 · FAX 415.335.8050 · E-mail: vx-sales@veritas.com · World Wide Web: http://www.veritas.com/

Table of Contents

Chapter

1

VERITAS Software Corporation

VERITAS File System Performance - White Paper

Introduction

The VERITAS storage management product line has been designed in response to the needs of commercial computing environments. These are the systems that are now supporting mission critical applications, and playing a major role in corporate computing services. Typical environments include online transaction processing systems, both inter as well as intra-networked database servers, and high performance file services. VERITAS specializes in systems storage management technology which encompasses a product line that offers high performance, availability, data integrity, and integrated, on-line administration. VERITAS provides three complementary products: VERITAS FirstWatch, VERITAS Volume Manager, and the VERITAS File System.

VERITAS FirstWatch is a system and application failure monitoring and management system that provides high-availability for mission-critical applications. FirstWatch dramatically increases server application availability through the use of duplicate cluster monitoring processes on each node in a FirstWatch system pair. These monitoring processes communicate with each other over dedicated, duplicated heartbeat links.

VERITAS Volume Manager is a virtual disk management system providing features such as mirroring, striping, disk spanning, hot relocation and I/O analysis. The VERITAS Visual Administrator exists as a graphical interface to VERITAS Volume Manager offering visual representation and management of the virtual disk subsystem including drag and drop features and simple or complex device creation.

The VERITAS File System is a high availability, high performance, commercial grade file system providing features such as transaction based journaling, fast recovery, extent-based allocation, and on-line administrative operations such as backup, resizing and defragmentation of the file system.

This report describes the performance mechanics of the 2.3 version of the VERITAS File System. This report will provide a discussion of the key performance components in the VERITAS File System. This report will also present a series of benchmark tests comparing the throughput and CPU utilization of different VERITAS File System component technologies, as well as provide some tests comparing the VERITAS File System (VxFS) with the Solaris UNIX File System (UFS).

VERITAS Software developed a series of benchmark tests for the express purpose of testing performance throughput of the installable file system, software component of the

UNIX OS. Since there are a number of components involved in computer file system, it was deemed necessary to develop a methodology, and later a program, to allow the configuration and running of a number of different I/O streams, in an effort to understand the role that file systems play in the overall file I/O model.

The testing included assembling a hardware test platform that would give the testers the ability to induce hardware bottlenecks at some specific points in order to see the overall effect on file system throughput. For the sake of this testing the performance hardware bottleneck areas focused on are:

- CPU - For most tests the desire was to not create a CPU bottleneck, but rather allow enough system CPU cycles, and system RAM, to exist for testing so that the throughput emphasis could be analyzed elsewhere.

- I/O Controller - The controller bus utilized in the tests was a Fast/Wide SCSI bus with the theoretical throughput of 20mb/sec. In some tests the controller was saturated in order to determine the overall effect on file system I/O performance.

- Disk - The disks utilized in the tests were all Fast Wide Differential SCSI hard drives with an aggregate throughput of approximately 6.5mb/sec. In other tests the disks were saturated in order to determine their effect on file system I/O performance.

Benchmark Testing platform

The complete breakdown of the hardware testing platform used for all benchmark tests is as follows:

- Hardware:
  - SUN Microsystems Ultra 4000 equipped with:
    - 4 Ultra SPARC Processors running at 167.5mhz
    - 256 MB system memory
    - 3 Andataco RSE-254S3w Storage Subsystems
      - each RSE-254S3W contained 8 Seagate ST3255OWD drives
      - each RSE-254S3W contained two fast/wide SCSI busses
    - 6 SUN 1062A F/W/D SBus controllers were used
- Software:
    - Solaris 2.5.1
    - VERITAS File System version 2.3
    - VERITAS Volume Manager version 2.3
    - VERITAS Software *vxbench* benchmark program
- Test Specifications:
    - All measurements were made with VERITAS Volume Manager RAID-0 array - volumes
    - These volumes were configured with 64 KB stripe units and the file system was aligned automatically by the combination of the VERITAS File System and Volume Manager, to stripe unit boundaries

Benchmark Program - VERITAS vxbench

VERITAS engineering developed a benchmark program specifically to allow a user to create a variety of I/O environments. Using this type of tool would allow the tester the ability to perform a wide variety of performance measurements on the installable file system component of the UNIX OS. The program developed is called *vxbench* and some of the features include the ability to utilize:

- All VxFS file system options
- All UFS file system options
- raw I/O, UFS, or VxFS partitions
- Multiple block sizes
- Multiple file sizes
- Random and sequential I/O

All of the test results described in this report were derived using vxbench.

As you will notice from the description below vxbench can perform I/O in utilizing multiple I/O streams. One multiple stream mode is to perform I/O to a single file using multiple threads, indicative of a database type application workload. Another multiple stream mode is to perform I/O to multiple files, via multiple threads, indicative of a multiple user server environment. The following is the list of vxbench options:

usage: vxbench `-w workload [options]` *`filename ...`*

Valid options are:

-h print more detailed help message
-P use processes for users, threads for multithreaded I/O (default)
-p use processes for users and for multithreaded I/O
-t use threads for users and for multithreaded I/O
-m lock I/O buffers in memory
-s for multiuser tests only print summary results
-v for multithreaded tests print per-thread results
-k print throughput in KB/sec (default)
-M print throughput in MB/sec

-w workload selects a type of I/O workload
valid workloads are:

read	sequential read of the test files
write	sequential write of the test files
rand_read	random read of the test files
rand_write	random write of the test files
rand_mixed	mix of random reads and writes
mmap_read	use mmap to read the test files
mmap_write	use mmap to overwrite the test files

-i subopts specify sub options describing test
valid sub options are:
nrep=*n* repeat the I/O loop in the test n times

	nthreads=n	number of threads accessing each file
	iosize=n	size of each I/O
	fsync	do an fsync on the file after writing it
	remove	remove each file after the test
	iocount=n	number of I/Os
	reserveonly	reserve space for the file but don't do I/O
	maxfilesize	maximum offset in KB for random I/O tests
	randseed	seed value for random number generator
	truncup	set an initial file size for random I/O
	rdpct=n	set read percentage of job mix for mixed tests
-o	opentype	specify flags for opening the file

valid opentypes are:

append	use appending writes
sync	set the O_SYNC flag for synchronous file I/O
trunc	truncate the test files on open

-c cacheopts specify VxFS caching advisories

valid cache options are:

direct	use direct I/O to bypass the kernel cache
dsync	use data synchronous I/O
noreuse	set the VX_NOREUSE cache advisory
random	set the VX_RANDOM cache advisory
seq	set the VX_SEQ cache advisory

-e extsize specify a fixed extent size

-r reservation specify space reservation

-f flags specify flags for reservation and fixed extents

valid flags are:

align	require aligned extents
chgsize	set the file size to the reservation size
contig	require contiguous allocation
noextend	don't allow writes to extend the file
noreserve	allocate space but don't set file reservation
trim	trim reservation to file size on last close

Specifying multiple filenames will run tests in parallel to each file, thus simulating multiple simultaneous users. If multiple threads are also specified, then each simulated user will run multiple threads so the total number of I/O threads will be 'users * nthreads'.

An example usage of vxbench would be as follows. If you wanted to measure I/O throughput of sequentially writing a 1024 MB file in 8 KB blocks, you would invoke vxbench as follows:

```
./vxbench -w write - i iosize=8k,iocount=128k /dev/vx/dsk/perfvol1
```

There is also a built-in help file that can be invoked by:

```
./vxbench -h.
```

Benchmarks used in this Report

Each series of benchmarks in this report includes configuration and parameter information that was used during the series of tests. In some cases there was an effort to test the file system software components with default settings, in other cases certain changes were made based upon the testing modes used. As mentioned previously, in some instances the hardware configuration was limited based upon the testing mode utilized. Specific information is available with each series of test results.

All benchmark tests run in this report were done using the vxbench program. Also except where noted, all benchmark testing involved testing sequential I/O. Finally, anyone wishing to use vxbench for performance testing of their file systems may obtain the program from VERITAS Software for no charge.

CPU Measurements

A note regarding the CPU measurements reported in this paper. The vxbench program measures two types of CPU times:

1. time spent in the operating system (system time)

2. time spent in the application (user time)

The way in which the measurements were reported in these tests was that both times were combined to come up with a single measurement of CPU impact. If the application time was reported as 10.5 seconds and the system time was 189.5 seconds, the final measurements would be reported as 200 CPU seconds. CPU utilization is, strictly speaking, this time divided by the elapsed time (which is not reported). The reason for using CPU seconds is to compare the relative CPU seconds per file system option when transferring the same amount of data.

Chapter
2

VERITAS File System

In response to the growing file system needs of commercial computing environments VERITAS Software developed their own installable file system initially for the UNIX environment. The VERITAS File System is a modern file system which is semantically similar to UFS, but which has been redesigned to support server-class file storage requirements. It adds a method (called journaling or intent-logging) to increase reliability, and uses more efficient, extent-based allocation policies as well as layout and caching modifications to more closely meet the I/O requirements of commercial applications and databases.

Journaling

The VERITAS File System employs a variation on the general file system logging or journaling technique by employing a circular *intent log*. All file system structure changes, or metadata changes, are written to this intent log in a synchronous manner. The file system will then periodically flush these changes out to their actual disk blocks. This increases performance by allowing all metadata writes to be written out to the permanent disk blocks, in an ordered manner out of the intent log.

Because the journal is written synchronously, it may also be used to accelerate small (less than or equal to 8KB) synchronous write requests, such as those used for database logging. Writes of this class may be written to the journal, a localized sequential block store, before they are moved to their places in the larger file system; this can reduce head movement and decrease the latency of database writes.

By using this intent log, the VERITAS File System can recover from system downtime in a fraction of the time. When the VERITAS File System is restarted in the same scenario, the system simply scans the intent log, noting which file system changes had completed and which had not, and proceed accordingly. In some cases, the VERITAS File System can roll forward changes to the metadata structures, because the changes were saved in the intent log. This adds availability and integrity to the overall file system.

Performance Components

The VERITAS File System has been developed with many of the latest industry file system performance improvements in mind. These improvements can be divided into the following feature categories:

- Extent Based Allocation

Unlike traditional UNIX file systems, which assign space to files one block at a time, the VERITAS File System allocates blocks in contiguous segments called extents. Extent sizes are chosen based on the I/O pattern of the file, or may be explicitly selected to suit the application. Extent-based allocation can accelerate sequential I/O by reducing seek and rotation time requirements for access, and by enabling drivers to pass larger requests to disks.

- Cache Policies

The UNIX operating system supplies a standard asynchronous mode for writing to files, in which data is written through a write-back page cache in system memory, to accelerate read and write access. It also calls for a synchronous mode, which writes through the cache immediately, flushing all structures to disk. Both of these methods require data to be copied between user process buffer space to kernel buffers before being written to the disk, and copied back out when read.

However, if the behavior of all processes that use a file is well-known, the reliability requirements of synchronous I/O may be met using techniques which offer much higher performance, often increasing file access to about the speed of raw disk access. The VERITAS File System provides two types of cache policies which enable these types of performance improvements.

The first method is called Direct I/O, and using this method the VERITAS File System does not copy data between user and kernel buffers; instead, it performs file I/O directly into and out of user buffers. This optimization, coupled with very large extents, allows file accesses to operate at raw-disk speed. Direct I/O may be enabled via a program interface, via a mount option, or with the 2.3 version of the VERITAS File System, Direct I/O can be invoked automatically based upon the I/O block size. This feature is known as Discovered Direct I/O.

The second cache policy available with the 2.3 version of the VERITAS File System is the Quick I/O for Database. While Direct I/O improves many types of large I/O performance, the single writer lock policy of the UNIX OS creates a performance bottleneck for some types of file system writes. Database application writes are particularly affected by this. Included in the VERITAS ServerSuite Database Edition 1.0, the Quick I/O for Database bypasses the single writer lock policy in the UNIX OS by representing files to applications as character devices. This allows database applications suited to utilizing raw partitions, the ability to operate like they are using a raw partition, on a file system. This combines the manageability of file systems with the performance of raw partitions.

Chapter
3

Extent Based Allocation

The VERITAS File System uses the much more efficient extent based allocation that improves the way in which large files are handled. Rather than linking indirect blocks of addresses, the VERITAS File System uses extent addresses which list a starting block address and a size. The disk blocks allocated for a file are stored in contiguous extents starting at the starting block address, and extending contiguously the number of blocks denoted by the size number.

Because a single pointer addresses more than one block, an extent-based file system requires fewer pointers and less indirection to access data in large files. UFS, with its 12 direct pointers, can only directly address up to 96KB of data (using 8KB blocks) without requiring at least one extra block of pointers and an indirect access. The VERITAS File System, with its 10 pointers to extents of arbitrary size, can address files of any supported size directly and efficiently.

What this translates to, is that when a large file is accessed in the VERITAS File System, the blocks needed can usually be found with no indirection, or directly. This direct addressing ability of the VERITAS File System dramatically increases the performance when the file system handles large files.

The VERITAS File System also provides interfaces for explicitly managing the layout of extents. Using a programmer interface or a command, one can choose a fixed extent size for a file, require that its blocks all be stored contiguously, and reserve space for its future growth. This allows for optimal performance for applications which manage large files, such as voice and image data.

The following Buffered File System Benchmark tests are intended to show the performance advantages of extent based allocation. What the test results should indicate is that as the file I/O size increases the VERITAS File System maintains throughput by benefit of using this allocation method.

Buffered File System Benchmark

The Buffered File System Benchmark tests the standard file system performance of the VERITAS File System (VxFS) against that of the Solaris UNIX File System (UFS). These tests were run using the hardware platform mentioned in Chapter 1, along with the vxbench testing program. The reads performed in this test are standard UNIX buffer cache reads and the file system used was installed as brand new, with little or no fragmentation.

Buffered File System Reads

File Size KB	UFS KB/sec	VxFS KB/sec	UFS CPU sec	VxFS CPU sec
1	82	51	0	0
2	166	104	0	0
4	334	147	0	0
8	617	299	0	0
16	1315	400	0	0
24	767	630	0	0
32	1023	938	0	0
40	1280	1057	0	0
48	1286	1217	0	0
56	1416	1492	0	0
64	1392	1712	0	0
72	1554	1934	0	0
80	1728	1734	0	0
88	1900	1890	0	0
96	2074	2096	0	0
104	1881	2280	0	0
112	1859	2545	0	0
120	1928	2657	0	0
128	1629	2836	0	0
160	2025	3478	0	0
192	2408	4024	0	0
224	2797	4668	0	0
256	2709	5367	0	0
320	3345	4979	0	0
384	4118	5654	0	0
448	4856	5630	0	0
512	5446	6183	0	0
1024	8179	9670	0	0
2048	10991	13688	0.1	0.1
4096	11581	18347	0.1	0.1
8192	11588	22073	0.2	0.2
16384	11451	24475	0.4	0.4
32768	12101	25463	0.7	0.7
65536	12241	26380	1.6	1.5
131072	12303	26204	2.9	3
262144	12470	26671	6.2	6.5
523072	12364	26896	13.5	15.1
1048576	12244	27094	26.1	30.1
2097144	12244	26625	52.1	62.5

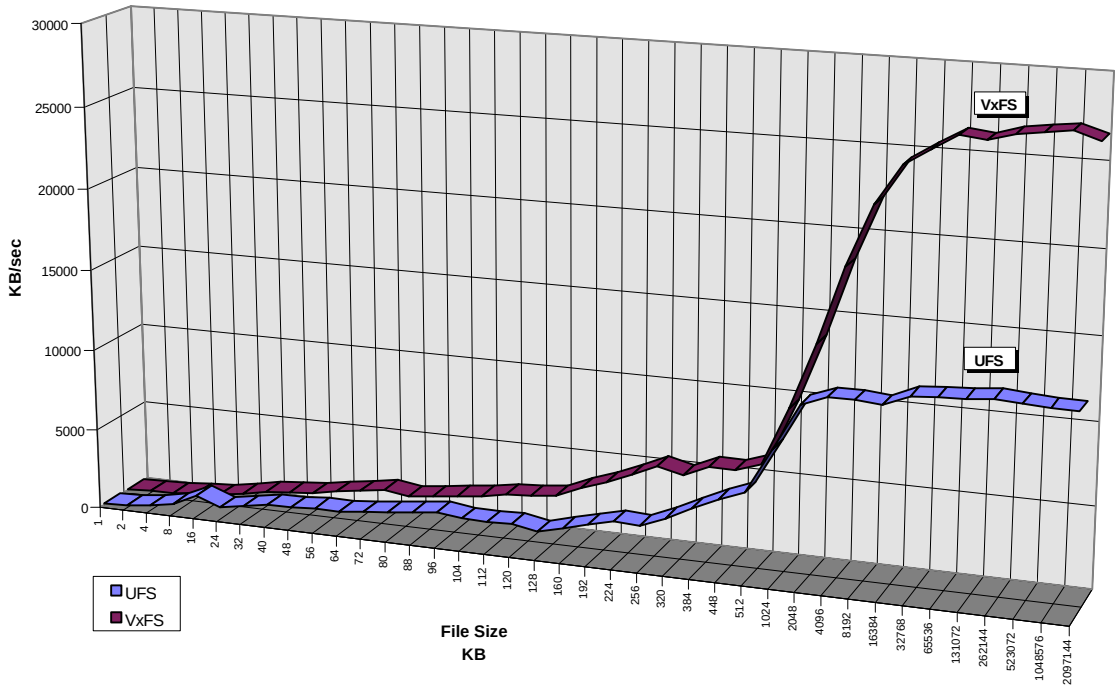
Buffered File System Writes

File Size KB	UFS KB/sec	VxFS KB/sec	UFS CPU sec	VxFS CPU sec
1	81	165	0	0
2	162	220	0	0
4	344	657	0	0
8	650	358	0	0
16	1386	716	0	0
24	2089	1036	0	0
32	2638	1317	0	0
40	3101	1458	0	0
48	2287	2185	0	0
56	2869	2262	0	0
64	3244	2716	0	0
72	3659	3003	0	0
80	4084	3437	0	0
88	4211	3974	0	0
96	4939	3969	0	0
104	2960	4935	0	0
112	3195	4908	0	0
120	3478	5395	0	0
128	3445	5693	0	0
160	4480	7306	0	0
192	5246	9065	0	0
224	6067	9603	0	0
256	6613	11217	0	0
320	8016	12222	0	0
384	9818	15621	0	0
448	8787	17642	0	0
512	8321	17578	0	0
1024	10197	27358	0	0
2048	12219	31196	0.1	0
4096	14956	39098	0.1	0.1
8192	15290	41263	0.2	0.2
16384	15729	39104	0.4	0.3
32768	16008	44400	0.8	0.6
65536	15720	50220	1.6	1.3
131072	15546	48538	3.5	2.5
262144	15920	43100	7.2	5.5
523072	15733	38389	15.7	11.3
1048576	15467	26642	30.9	24.5
2097144	15326	23141	62.4	52.5

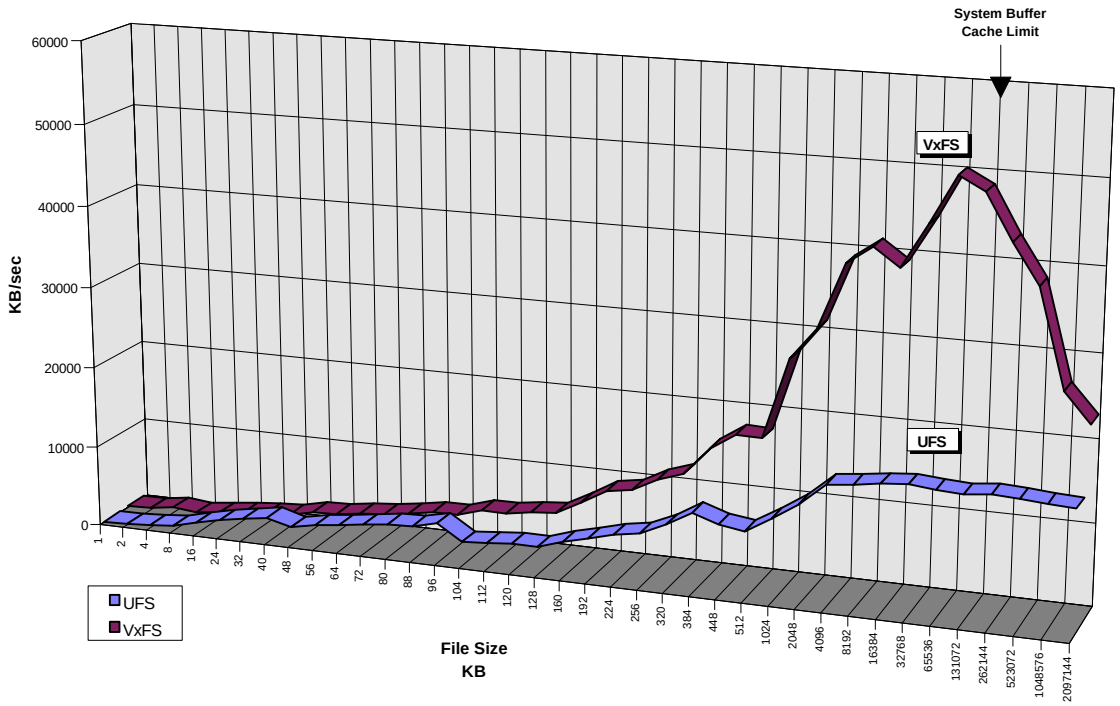
These buffered tests were done using the default UFS and VxFS parameters. The hardware disk configuration used was a single SCSI controller connected to 4 SCSI drives, creating a single 8 GB RAID-0 volume.

Buffered File System Tests - Graphs

Buffered File System Read Tests



Buffered File System Write Tests



As the results indicate, both UFS and VxFS standard buffered read throughput begins to accelerate at the 512KB size range, however the increase for VxFS is much larger than UFS peaking at almost 26 MB/sec whereas UFS almost reaches the 12 MB/sec range. Read CPU utilization indicates a very similar curve between the two file systems.

The buffered write results provide a similar picture on the right side of the table. These results show the same similar increase in throughput at the 512KB size, however the increase in VxFS throughput climbs to almost 50 MB/sec while UFS manages almost 16 MB/sec.

Note that during the buffered write tests, the VERITAS File System, by default, reached it's maximum limit of one half the system RAM for use as a buffer cache. Since our test platform system had 256 MB of memory, by default the VERITAS File System will limit itself to not using more than one half, or in this case 128 MB, of the installed memory for system buffers. If you were to add memory to this system, this would increase this ceiling, and increase the performance throughput measurements.

Buffered Aged File System Benchmark

This next series of buffered tests, as in the previous series, uses the buffered mode of the VERITAS File System, only. The next chapter will provide test results for the VxFS Discovered Direct I/O, and Direct I/O technologies.

This second series of buffered tests involves measuring the impact that external fragmentation has on file system performance. The way in which this was accomplished was to first write a file to a new file system and then perform a read on the file. Three different block sizes were used for performing these read tests.

Next multiple files were simultaneously written to a new file system, creating fragmented file allocations, and then those same files were read back in combinations while the system read throughput was measured.

These buffered aged file system tests were done using the default UFS and VxFS parameters. The file size used in all iterations was 256 MB. The hardware disk configuration used was 4 SCSI controllers connected to 8 SCSI drives, creating a single 16 GB RAID-0 volume.

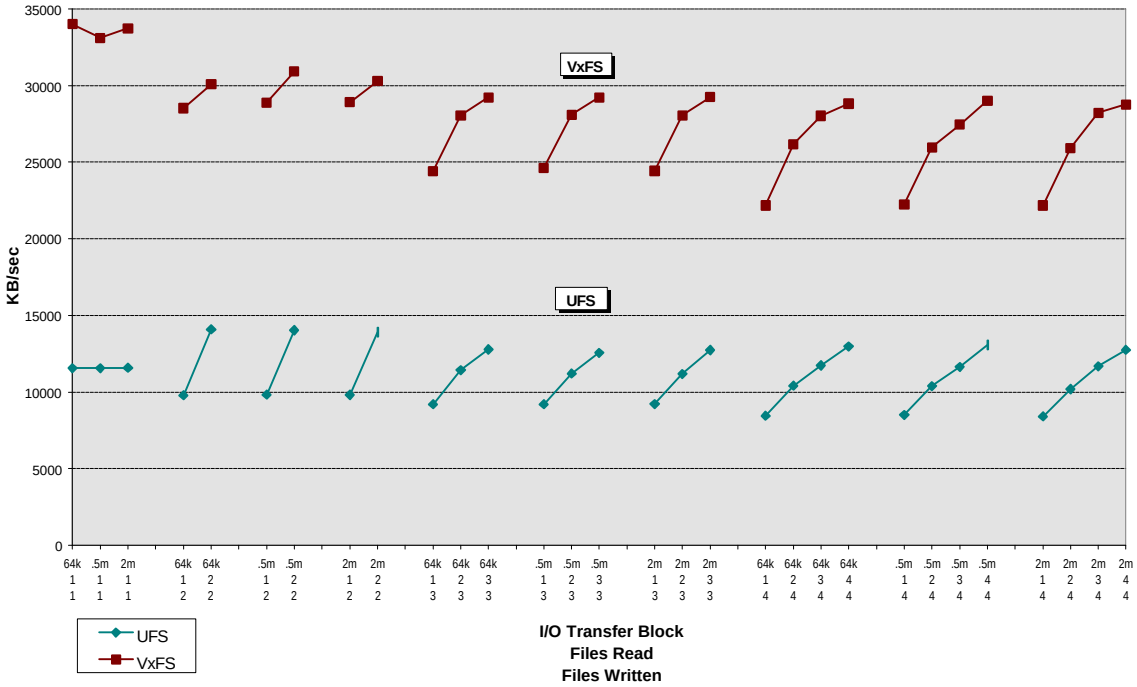
Buffered Aged File System Reads UFS vs. VxFS

Files Written	Files Read	I/O Transfer Block Size KB	UFS Buffered KB/sec	VxFS Buffered KB/sec	UFS Buffered CPU sec	VxFS Buffered CPU sec
1	**1**	**64**	11558	34000	6.32	6.29
1	**1**	**512**	11558	33100	6.03	6.27
1	**1**	**2048**	11584	33717	6.11	6.3
2	**1**	**64**	9783	28524	6.51	6.87
2	**2**	**64**	14095	30074	15.27	15.36
2	**1**	**512**	9823	28875	6.82	6.4
2	**2**	**512**	14037	30911	14.54	15.13
2	**1**	**2048**	9821	28905	6.16	6.23
2	**2**	**2048**	13908	30288	14.39	15.6
3	**1**	**64**	9211	24415	6.73	6.33
3	**2**	**64**	11432	28053	15.72	15.38
3	**3**	**64**	12787	29224	22.72	24.51
3	**1**	**512**	9207	24619	6.22	5.99
3	**2**	**512**	11211	28090	14.52	14.8
3	**3**	**512**	12564	29213	23.21	23.53
3	**1**	**2048**	9228	24438	6.01	6.41
3	**2**	**2048**	11184	28049	14.96	15.5
3	**3**	**2048**	12744	29236	23.56	24.12
4	**1**	**64**	8461	22174	6.84	6.32
4	**2**	**64**	10411	26163	15.18	15.52
4	**3**	**64**	11741	28022	24.23	24.03
4	**4**	**64**	12983	28807	33.31	32.4
4	**1**	**512**	8504	22246	7.17	6.18
4	**2**	**512**	10391	25969	14.49	14.35
4	**3**	**512**	11631	27457	23	23.46
4	**4**	**512**	13073	29013	31.12	32.57
4	**1**	**2048**	8415	22175	7.32	6.07
4	**2**	**2048**	10195	25914	14.8	14.75
4	**3**	**2048**	11694	28214	23.46	23.32
4	**4**	**2048**	12759	28766	31.24	31.55

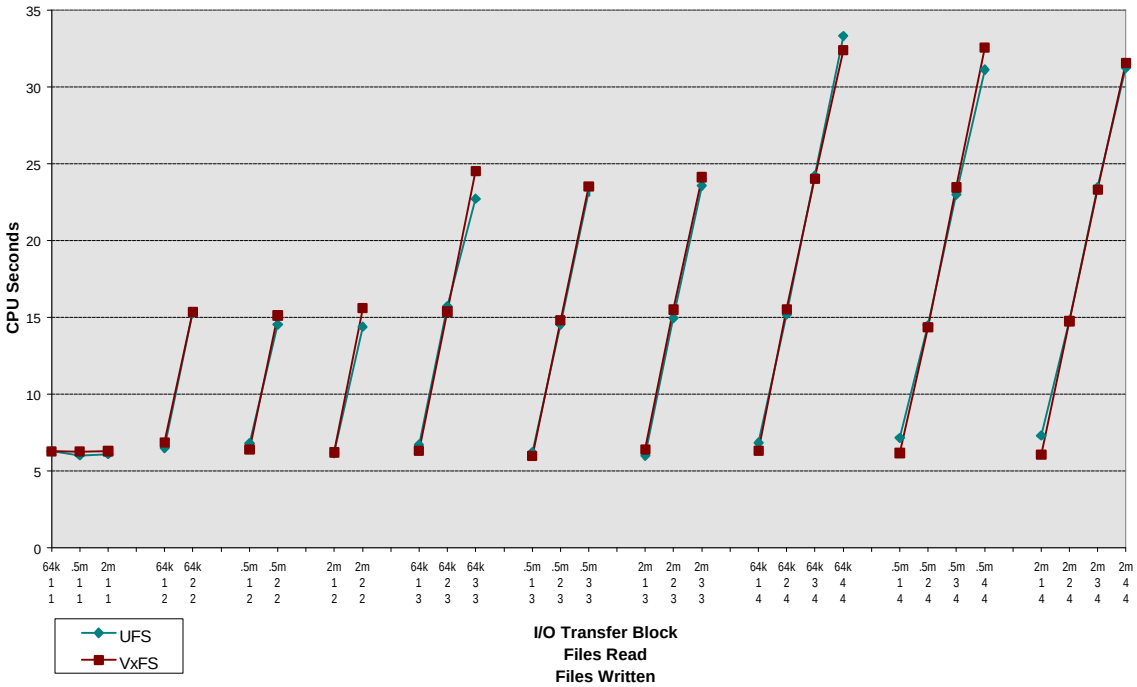
Results indicate that the fragmentation effect reduces the throughput for both UFS and VxFS. However, the results indicate that VxFS begins and maintains a higher read throughput rate and is less affected by the fragmentation. The CPU time curves are almost identical for both VxFS and UFS. (In the next chapter will illustrate the VxFS Discovered Direct I/O technology which results in much lower CPU utilization, in addition to boosting large block I/O performance.)

The following graphs look at these test results:

Buffered Aged File System Read Tests



Buffered Aged File System Read Tests



Chapter
4

Cache Policies

The UNIX operating system supplies a standard asynchronous mode for writing to files, in which data is written through a write-back page cache in system memory, to accelerate read and write access. It also calls for a synchronous mode, which writes through the cache immediately, flushing all structures to disk. Both of these methods require data to be copied between user process buffer space to kernel buffers before being written to the disk, and copied back out when read.

As mentioned previously, the VERITAS File System provides two types of cache policies which enable the File System to circumvent the standard UNIX write-back page cache in system memory. The first cache policy is a feature called Direct I/O. VERITAS implemented this feature in their file system and it provides a mechanism for bypassing the UNIX system buffer cache while retaining the on disk structure of a file system. This optimization, coupled with very large extents, allows file accesses to operate at raw-disk speed. Direct I/O may be enabled via a program interface, via a mount option, or with the 2.3 version of the VERITAS File System, Direct I/O can be invoked automatically based upon the I/O size. This feature is known as Discovered Direct I/O.

The second cache policy available with the 2.3 version of the VERITAS File System is the Quick I/O for Database. While Direct I/O improves many types of large I/O performance, the single writer lock policy of the UNIX OS creates a performance bottlenecks for some types of file system writes. Database application writes are particularly affected by this. Included in the VERITAS ServerSuite Database Edition 1.0, the Quick I/O for Database bypasses the single writer lock policy in the UNIX OS by representing files to applications as character devices. This allows database applications suited to utilizing raw partitions, the ability to operate like they are using a raw partition, on a file system.

The most important requirement for implementing these two VERITAS File System cache policies is that all I/O requests must meet certain alignment criteria. This criteria is usually determined by the disk device driver, the disk controller, and the system memory management hardware and software. First the file offset must be aligned on a sector boundary. Next the transfer size must be a multiple of the disk sector size. Finally depending on the underlying driver, the application buffer may need to be aligned on a sector or page boundary, and subpage length requests should not cross page boundaries.

The method for guaranteeing this requirement, as well as generally improving performance for RAID level volumes, is by utilizing a technique called file system alignment.

File System Alignment

While RAID technology increases performance in some implementations, tuning RAID systems for proper file system alignment can increase performance for most striped RAID configurations. In most commercial implementations this involves using RAID-1, RAID-5 and RAID-0+1 configurations.

The technique behind file system alignment involves setting the layout of the file system across the drive array in such a manner that the workload is distributed as equally as possible. In order to accomplish this there must be a determination as to what sections of the file system to distribute, and there must be a method for aligning these sections.

This can be accomplished with most modern file systems that use data grouping techniques. The beginning of a UNIX UFS cylinder group contains the metadata blocks, and these blocks tend to be centers of disk activity. Aligning the UFS file system so that the cylinder groups begin on different drives in the array will align the file system for this method. Using this technique allows the separate drives in the array to perform the highest amount of simultaneous accesses.

The way in which this can be accomplished is by the setting of the RAID stripe unit size. This is the size of disk space, on each disk, that is accessed in one pass. The combined total stripe size of all the disks is known as the RAID stripe width. Setting the stripe size to 512KB on a 3 column (disk) RAID-0 array, would result in a stripe width of 1.5MB (512x3).

Since the cylinder group size in UNIX is typically 4MB, setting the stripe unit size to 512K for a 3 column array as described, would mean that the beginning of each subsequent cylinder group begins on a different drive in the array.

The VERITAS File System's cylinder groups, called allocation units (AU), do not contain similar metadata blocks at the beginning of the AU. Inodes are allocated dynamically within the data blocks of each AU. What is more important in terms of file system alignment for this file system is keeping the AUs allocated on even disk boundaries. This provides increased performance throughout the file system, as well as allow Direct I/O technologies to be utilized. What this necessitates is padding the AUs so that they begin and end on even disk boundaries.

> In the 2.3 version of the VERITAS File System this is done automatically if the disks are being managed by the 2.3 version of the VERITAS Volume Manager.

Direct I/O

VERITAS Software has developed a cache policy called Direct I/O in their file system and it provides a mechanism for bypassing the UNIX system buffer cache while retaining the on disk structure of a file system. The way in which Direct I/O works involves the way the system buffer cache is handled by the UNIX OS. In the UNIX operating system, once the type independent file system, or VFS, is handed a I/O request, the type dependent file system scans the system buffer cache, and verifies whether or not the requested block is in memory. If it is not in memory the type dependent file system manages the I/O processes that eventually puts the requested block into the cache.

Since it is the type dependent file system that manages this process, the VERITAS File System uses this to bypass the UNIX system buffer cache. Once the VERITAS File System returns with the requested block, instead of copying the contents to a system buffer page, it instead copies the block into the application's buffer space. Thereby reducing the time and CPU workload imposed by the system buffer cache.

In order to ensure that Direct I/O mode is always enabled safely, all Direct I/O requests must meet certain alignment criteria. This criteria is usually determined by the disk device driver, the disk controller, and the system memory management hardware and software. First the file offset must be aligned on a sector boundary. Next the transfer size must be a multiple of the disk sector size. Finally depending on the underlying driver, the application buffer may need to be aligned on a sector or page boundary, and sub-page length requests should not cross page boundaries.

Direct I/O requests which do not meet these page alignment requirements, or which might conflict with mapped I/O requests to the same file, are performed as data-synchronous I/O. This optimization, coupled with very large extents, allows file accesses to operate at near raw-disk speed.

Discovered Direct I/O

Prior to the 2.3 version of the VERITAS File System invoking Direct I/O was possible via one of two ways. The first way was via a programmatic interface, an application developer could enable their application to specifically invoke Direct I/O using the VxFS system calls. The second method allowed a system administrator to specifically mount a file system whereby all I/O performed on that file system was done via Direct I/O.

Since the benefit of Direct I/O is evident at larger I/O sizes, VERITAS Software implemented a feature in their 2.3 version of the VERITAS File System, which allows Direct I/O to be invoked automatically once the file system I/O reaches a specified size. This feature is known as Discovered Direct I/O and is controlled via the *vxtunefs* command. The *discovered_direct_iosz* (Discovered Direct I/O size) parameter, which defaults to 256 KB, is the parameter that controls whether or not the file system should handle an I/O transaction with either buffered or Direct I/O. Once the file system I/O is greater than 256K in size, the file system will automatically handle the I/O as Direct I/O.

The next series of tests were performed in order to compare Direct I/O and Discovered Direct I/O against the performance of a UNIX raw partition. These next two series of tests were run with bottlenecks created at two specific locations. The first series of tests were run in order to induce a bottleneck at the channel controller level. These were done by utilizing a single SCSI channel connected to 4 SCSI drives. The second series of tests were run in order to induce a bottleneck at the disk level. These were done by utilizing four SCSI channels connected to 8 SCSI drives (2 each).

Controller Limited Benchmark - 4 way Stripe

The controller limited benchmarks compare the performance throughput differences between VxFS buffered, Direct I/O, Discovered Direct I/O and UNIX raw I/O. These tests were performed on the benchmark platform described in Chapter 1, limiting the hardware to a single SCSI controller, connected to 4 disks in a striped RAID-0 array.

This purposely introduced a performance bottleneck at the controller level. Theoretically a Fast / Wide SCSI channel can produce 20 MB/sec of streaming data. Due to SCSI bus

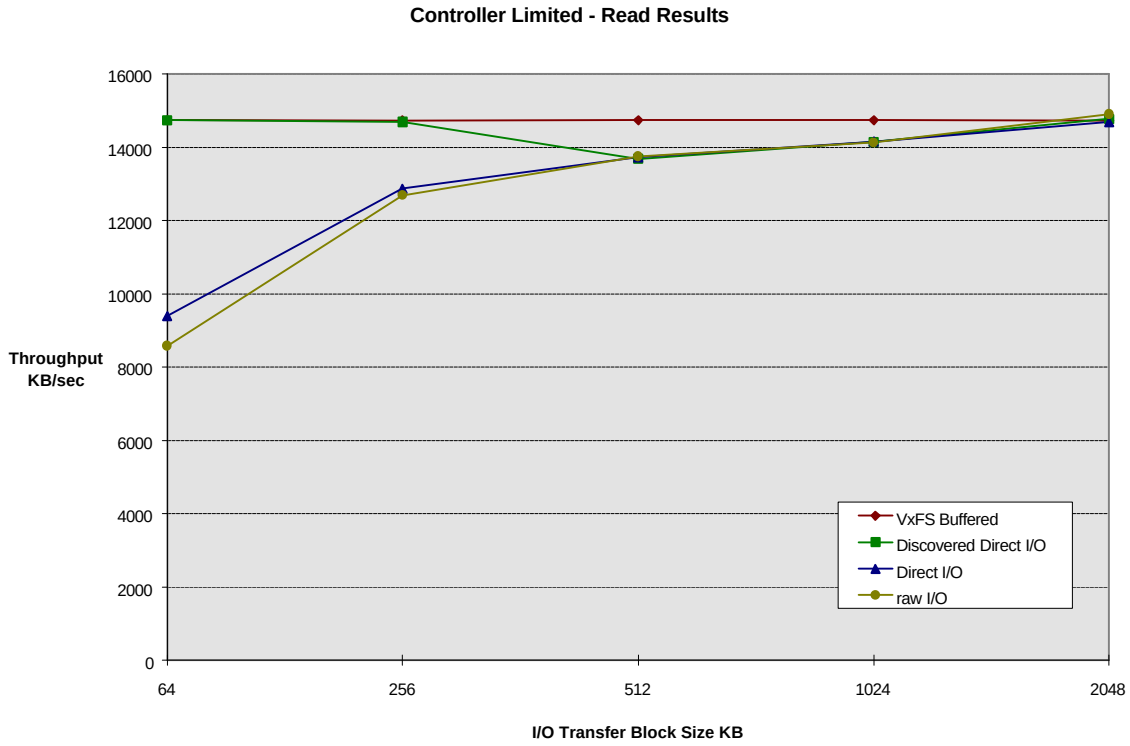
technology the real limit approaches 16 MB/sec throughput. Since each SCSI disk in the test platform can produce approximately 6.5 MB/sec, with four disks per controller this creates a bottleneck at the controller level. This was done in order to illustrate the performance differences between the tested technologies in this limited throughput environment.

Controller Limited Read Tests - VxFS Buffered / Discovered Direct / Direct / UNIX raw I/O

I/O Transfer Block Size KB	VxFS Buffered KB/sec	Discovered Direct I/O KB/sec	Direct I/O KB/sec	raw I/O KB/sec	VxFS Buffered CPU sec	Discovered Direct I/O CPU sec	Direct I/O CPU sec	raw I/O CPU sec
64	14732	14733	9392	8582	6.67	6.03	1.26	1.15
256	14723	14687	12867	12683	6.28	6.24	0.76	0.79
512	14732	13684	13722	13745	6.6	0.7	0.85	0.75
1024	14730	14140	14149	14124	6.29	0.49	0.45	0.71
2048	14722	14770	14680	14898	5.91	0.72	0.7	0.74

These controller limited tests were done using the default UFS and VxFS parameters. The file size used in all iterations was 256 MB. The hardware disk configuration used was a single SCSI controller connected to 4 SCSI drives, creating a single 8 GB RAID-0 volume.

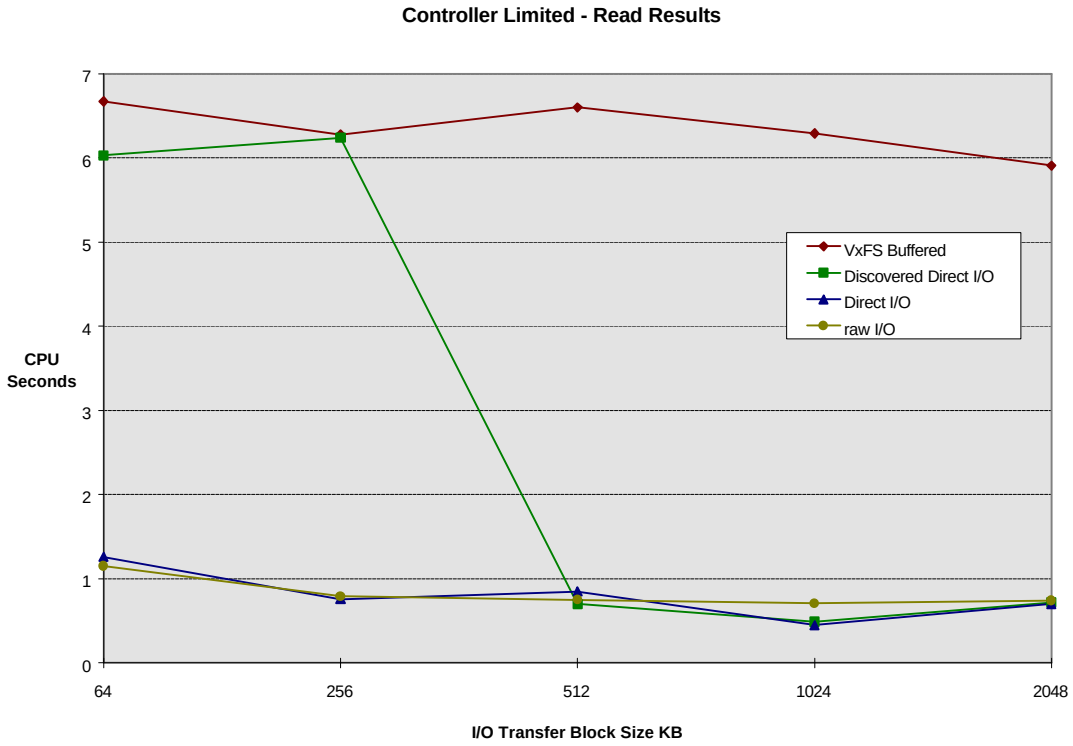
Controller Limited Read Test Graphs - VxFS Buffered / Discovered Direct / Direct / UNIX raw I/O



These file system read results show the benefit of Discovered Direct I/O. While the I/O size remains below the discovered direct I/O size of 256K, the file system performs standard buffered I/O. Once above that size, both Discovered Direct I/O and Direct I/O throughput performance climb along the same curve of raw I/O.

Note that the raw I/O final throughput of nearly 16 MB/sec is almost the maximum available throughput given the controller limited testing. This was done in order to illustrate that in terms of its scalability the VERITAS File System can provide throughput that is very close to the actual hardware limits. This model of providing the highest realized throughput for each installed system, scales as you install the VERITAS File System on larger and faster platforms.

The second interesting result is the fact that the CPU utilization is high while using standard buffered I/O in the Discovered Direct I/O category, and then drops appreciably once Direct I/O is invoked once past the 256 KB block size. This demonstrates the potential for tremendous scalability which will be realized in testing later in this chapter.



Disk Limited Benchmark - 8 way Stripe

The first series of disk limited benchmarks compare the performance throughput differences between Solaris UNIX raw I/O with the VERITAS File System Buffered I/O, Discovered Direct I/O and Direct I/O. These tests were performed on the benchmark platform described in Chapter 1, limiting the hardware to four SCSI controllers, each connected to 2 disks in a striped RAID-0 array.

This purposely introduced a performance bottleneck at the disk level. Theoretically a Fast / Wide SCSI channel can produce 20 MB/sec of streaming data. Due to SCSI bus technology the real limit approaches 16 MB/sec throughput. Since each SCSI disk in the test platform can produce approximately 6.5 MB/sec, with two disks per controller this creates a bottleneck at the disk level.

Disk Limited Read Tests - VxFS Buffered / Discovered Direct I/O / Direct I/O / raw I/O

I/O Transfer Block Size KB	VxFS Buffered KB/sec	Discovered Direct I/O KB/sec	Direct I/O KB/sec	raw I/O KB/sec	VxFS Buffered CPU sec	Discovered Direct I/O CPU sec	Direct I/O CPU sec	raw I/O CPU sec
64	35896	34935	10595	8490	5.85	5.91	1.3	1.13
256	36091	35799	20238	20217	5.72	5.67	0.69	0.7
512	38342	30868	33996	29491	5.76	0.73	0.68	0.74
1024	38530	45136	45893	45686	5.77	0.65	0.64	0.66
2048	34265	47851	44927	49444	5.71	0.7	0.73	0.62

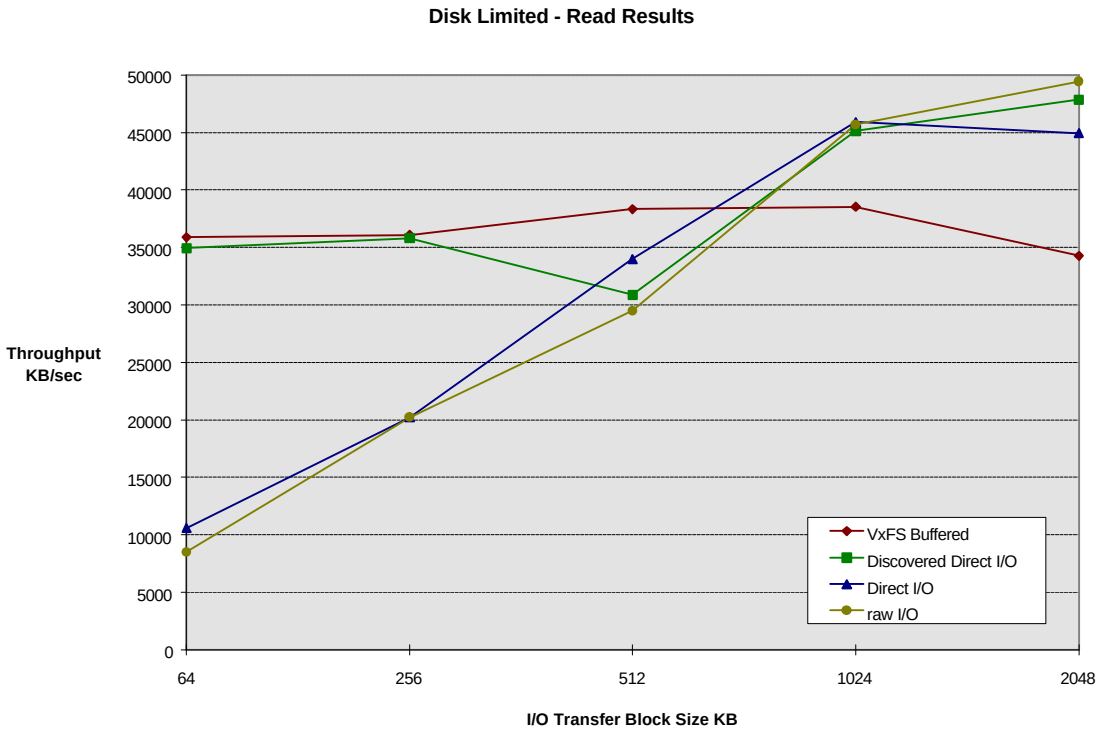
These set of disk limited tests were done using the default UFS and VxFS parameters. The file size used in all iterations was 256 MB. The hardware disk configuration used was 4 SCSI controllers connected to 8 SCSI drives, creating a single 16 GB RAID-0 volume.

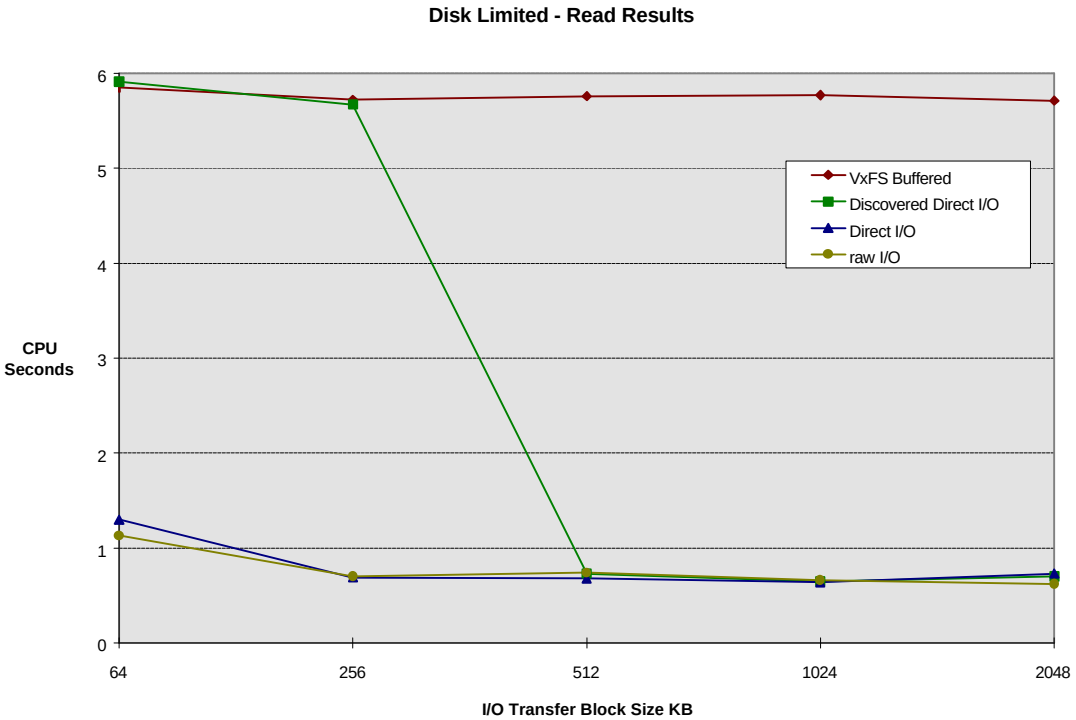
This set of tests compares the file system read performance throughputs of VxFS Buffered, Discovered Direct I/O, Direct I/O and Solaris UNIX raw I/O. Here again we see the combined advantages of VxFS buffered I/O and Direct I/O performance. As soon as Discovered Direct I/O invokes the Direct I/O mode, the performance throughputs between the three technologies is very similar. We also see the same drop in CPU utilization once Discovered Direct I/O invokes the Direct I/O technology.

Again in this series of tests, note that the raw I/O final throughput of nearly 52 MB/sec is the maximum available throughput given the disk limited testing (6.5 MB/sec times 8 drives). This again illustrates that in terms of its scalability the VERITAS File System can provide throughput that is very close to the actual hardware limits. This also illustrates that standard buffered technology reaches bottlenecks very quickly when pushed.

This model of providing the highest realized throughput for each installed system, scales as you install the VERITAS File System on larger and faster platforms.

Disk Limited Read Test Graphs- VxFS Buffered / Discovered Direct I/O / Direct I/O / raw I/O





The next series of disk limited benchmarks compare the performance throughput differences between Solaris UNIX raw I/O and UFS buffered I/O with the VERITAS File System Discovered Direct I/O and Direct I/O. Additionally this entire series of disk limited tests were done utilizing a specific mode of the vxbench program. This mode is defined here as a multiple application mode in which multiple threads perform their respective file I/O, to their own unique file. This is an application workload that is similar to a standard server environment.

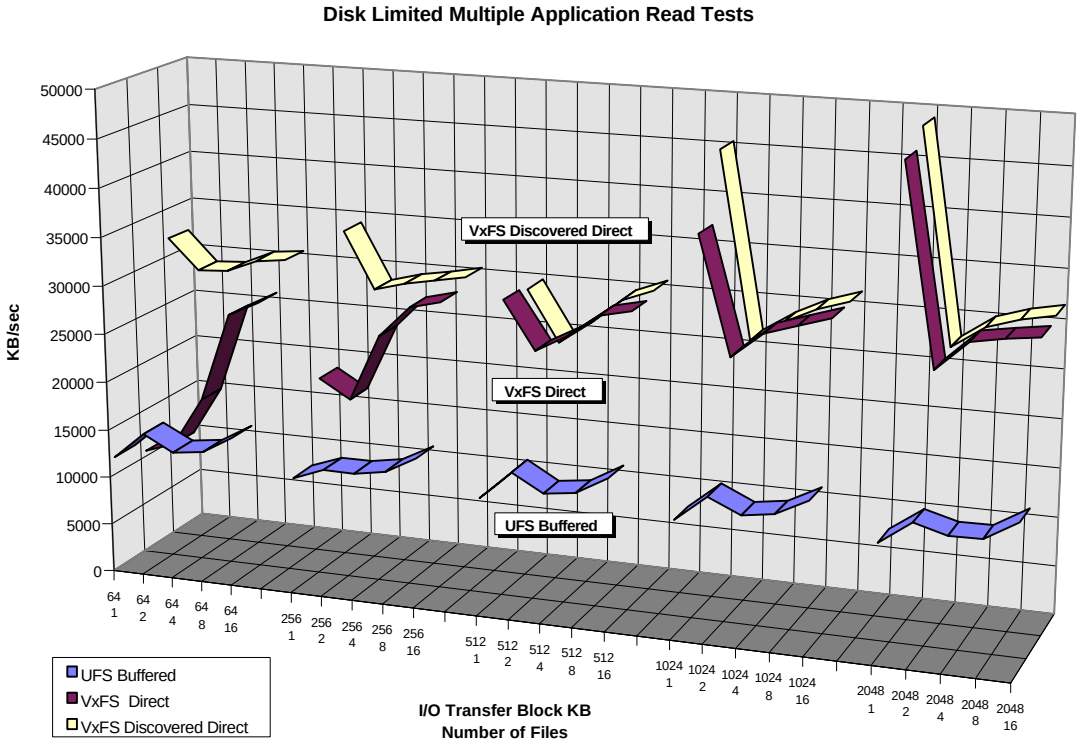
These tests were done using the default UFS and default VxFS parameters with one change. The vxtunefs parameter of *max_direct_iosz* was set to 2 MB. The file size used in all iterations was 256 MB. The hardware disk configuration used was 4 SCSI controllers connected to 8 SCSI drives, creating a single 16 GB RAID-0 volume.

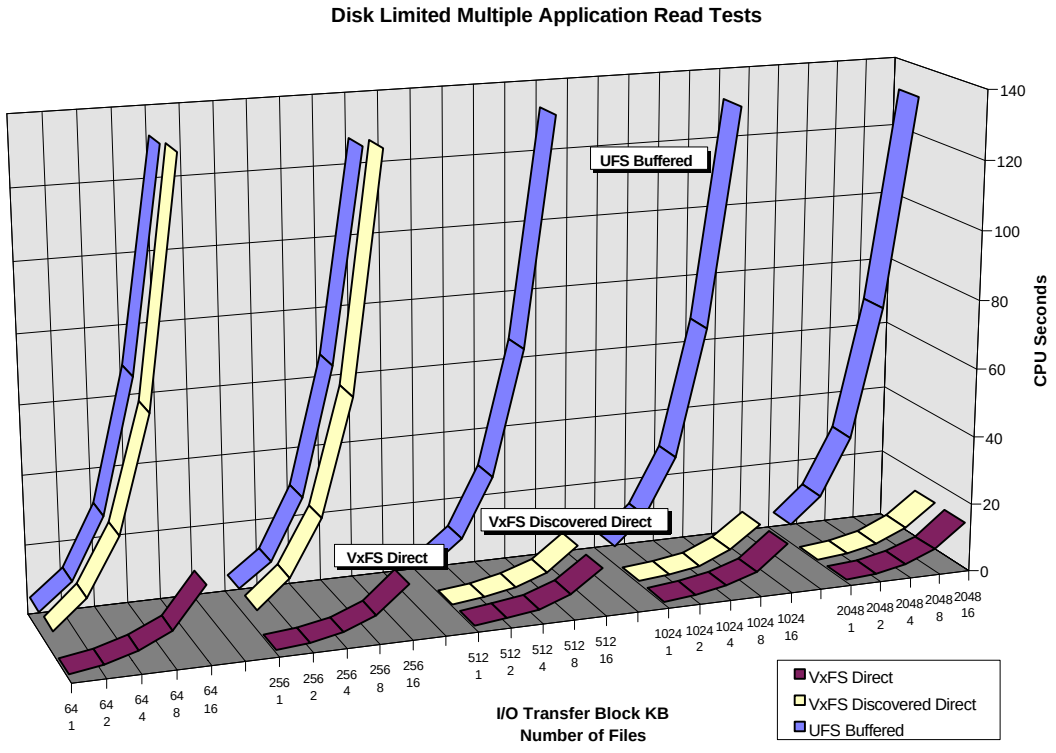
Disk Limited Multiple Application Read Tests

Number of Files	I/O Transfer Block Size KB	UFS Buffered KB/sec	VxFS Discovered Direct KB/sec	VxFS Direct KB/sec	UFS Buffered CPU sec	VxFS Discovered Direct CPU sec	VxFS Direct CPU sec
1	**64**	11818	31797	10615	5.8	6.6	1.3
2	**64**	14600	28665	11727	13.4	14.9	2.8
4	**64**	12992	28855	16797	31	31.2	5.5
8	**64**	13406	30179	26025	69.3	64.2	9.8
16	**64**	15244	30553	27639	132.4	133.6	21.4
1	**256**	11722	34146	20276	5.8	5.8	0.8
2	**256**	12895	28275	18421	12.6	13.9	1.7
4	**256**	12897	29193	25344	30.2	29.9	3.6
8	**256**	13427	29762	28598	67.2	63.2	7.2
16	**256**	15170	30385	29404	127.8	130.8	15
1	**512**	11771	29738	30181	5.6	0.6	0.8
2	**512**	14791	24510	25317	12.9	1.6	1.8
4	**512**	12952	26814	27085	30.1	3.4	3.1
8	**512**	13390	29435	29477	66.7	6.6	6.4
16	**512**	15228	30705	30212	132.9	13.4	12.9
1	**1024**	11781	45345	38373	5.2	0.8	0.8
2	**1024**	14482	26498	26531	13.5	1.6	1.7
4	**1024**	12997	28668	29143	30.1	3.1	3.3
8	**1024**	13461	30262	30191	67.8	7	6.2
16	**1024**	15195	31314	31264	131.7	13.5	13.5
1	**2048**	11795	48987	47010	5.5	0.6	0.7
2	**2048**	14152	27654	27128	13.3	1.4	1.3
4	**2048**	13209	29916	30138	30.3	3.4	3.1
8	**2048**	13309	30991	30737	68.9	7.3	6.5
16	**2048**	15266	31650	31195	131	13.8	13.5

These test results offer a good example of the combination performance of VxFS buffered and Direct I/O, in the Discovered Direct I/O feature. Note that while the UFS throughput results remain relatively flat, VxFS Discovered Direct I/O provides better initial throughput performance than Direct I/O, and then provides a similarly increasing curve as Direct I/O. The CPU time measurements again indicate the CPU resource differences between buffered file system activity, and Direct I/O file system activity.

Disk Limited Multiple Application Read Test Graphs





Tuning UFS File Systems

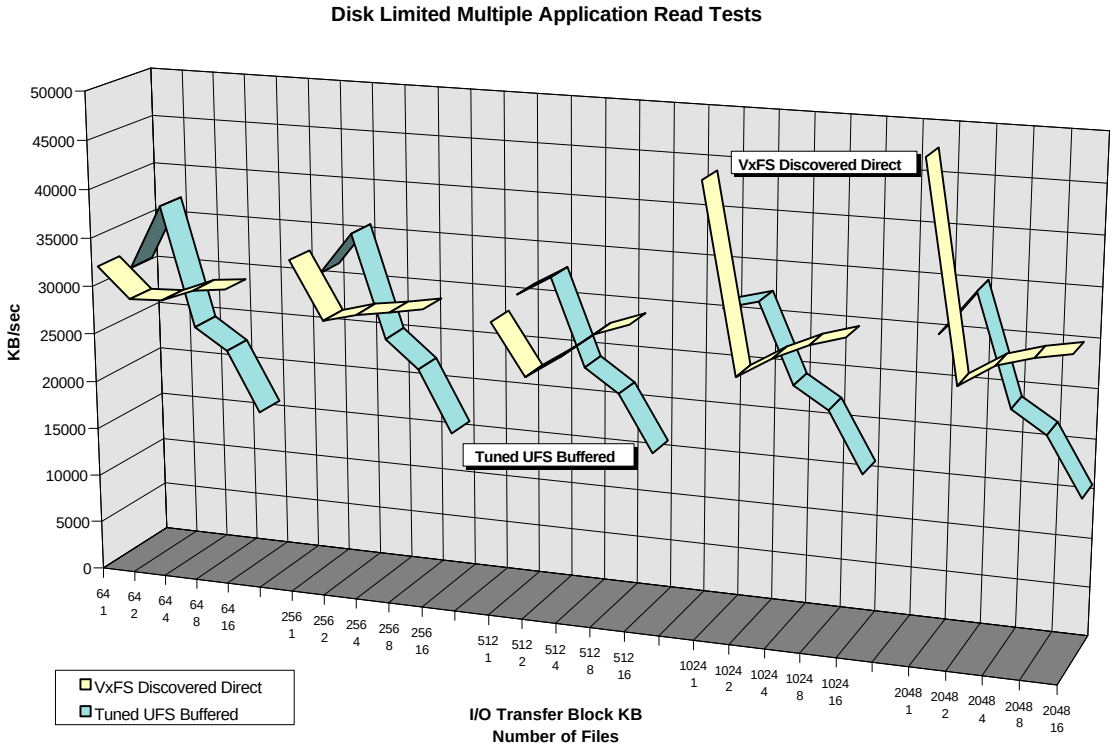
Most benchmark tests in this report were run with little changes to the default file system parameters. However there are a number of tunable settings for both UFS and VxFS that can be utilized to increase performance, for a given application workload. As mentioned previously in the section on UFS, the UFS cylinder groups can be aligned for improving performance of UFS throughput. UFS also contains some file system settings which can affect overall system throughput. The tuning information used in this series of tests is included in the Tuning UFS section following the graphs.

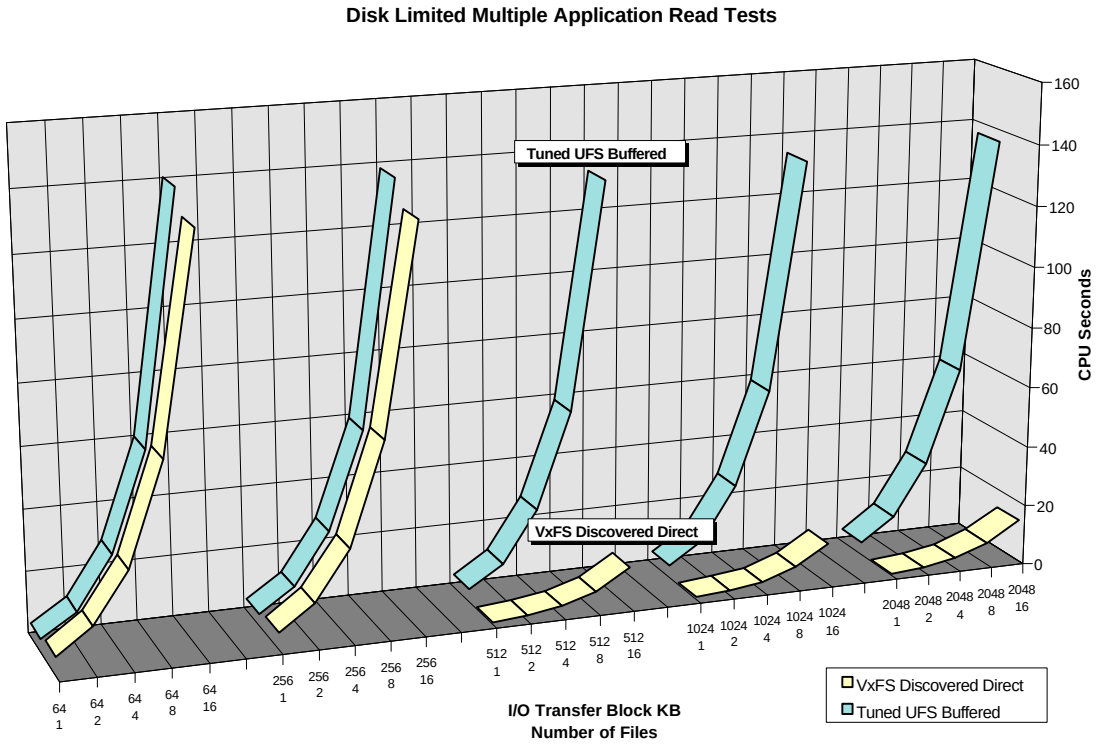
The next series of disk limited tests were done comparing the Solaris UFS buffered I/O tuned for large block I/O, and the VERITAS File System Discovered Direct I/O using a multiple application mode test. These tests were done using the tuned UFS parameters (explained below) and the default VxFS parameters with one change. The vxtunefs parameter of *max_direct_iosz* was set to 2 MB. The file size used in all iterations was 256 MB. The hardware disk configuration used was 4 SCSI controllers connected to 8 SCSI drives, creating a single 16 GB RAID-0 volume.

Disk Limited Multiple Application Read Tests - Tuned UFS Buffered / VxFS Discovered Direct

Number of Files	I/O Transfer Block Size KB	Tuned UFS Buffered KB/sec	VxFS Discovered Direct KB/sec	Tuned UFS Buffered CPU sec	VxFS Discovered Direct CPU sec
1	**64**	30074	31797	4.4	6.6
2	**64**	36802	28665	11.7	14.9
4	**64**	24339	28855	28.8	31.2
8	**64**	22091	30179	60.8	64.2
16	**64**	15856	30553	141.1	133.6
1	**256**	31230	34146	4.2	5.8
2	**256**	35576	28275	11.7	13.9
4	**256**	24958	29193	28.4	29.9
8	**256**	22106	29762	60	63.2
16	**256**	15744	30385	138.9	130.8
1	**512**	30845	29738	4.2	0.6
2	**512**	32872	24510	11.2	1.6
4	**512**	23949	26814	27.9	3.4
8	**512**	21595	29435	59.2	6.6
16	**512**	15846	30705	133.1	13.4
1	**1024**	31250	45345	4.1	0.8
2	**1024**	32212	26498	11.3	1.6
4	**1024**	24152	28668	28.6	3.1
8	**1024**	22009	30262	59.4	7
16	**1024**	15891	31314	134.3	13.5
1	**2048**	30548	48987	4.2	0.6
2	**2048**	35032	27654	11.9	1.4
4	**2048**	23831	29916	29	3.4
8	**2048**	21632	30991	60	7.3
16	**2048**	15746	31650	136	13.8

Disk Limited Multiple App. Read Test Graphs - Tuned UFS Buffered / VxFS Discovered Direct





Tuned UFS demonstrates a pretty consistent performance throughput while VxFS again demonstrates higher throughput during a majority of the testing. These file system read performance numbers indicate that proper tuning of Solaris UFS can increase overall throughput while reducing CPU utilization over the standard default Solaris UFS. However VxFS with Discovered Direct I/O, still provides better throughput with reduced CPU utilization for large file I/O.

Additionally, tuning UFS for larger file I/O does impact general purpose servers that continue to perform a mix of I/O. For example tuning UFS for large file I/O, with more than one application, can cause serious degradation in systems performance, due to the OS paging mechanism and excessive CPU utilization. A better alternative is VxFS with Discovered Direct I/O

Tuning UFS

There are a number of information resources available in the UNIX industry which describe tuning UFS for different application workloads. Among those, DBA Systems, Inc. of Melbourne, FL. has done a lot of work in the area of tuning UFS for performing large file I/O. A report completed by William Roeder of DBA Systems, Inc. outlines the basic steps involved in performing this tuning. The information presented there is repeated here only in summary form.

In tuning UFS one of the most important settings is the *maxcontig* parameter. By using this to alter the number of contiguous file blocks that UFS allocates for a file, the UFS system will actually operate more like an extent based file system. Other settings that can be used for tuning UFS for large block I/O are *fragsize*, *nbpi* and *nrpos*. Using this information the UFS file system created for this large file I/O testing was done with the following command:

```
>mkfs -F ufs -o nsect=80,ntrack=19,bsize=8192,fragsize=8192,
cgsize=16,free=10,rps=90,nbpi=32768,opt=t,apc=0,gap=0,nrpos=1,
maxcontig=1536 /dev/vol/rdsk/vol01 256880
```

Passing the GB/sec Barrier

Since the benchmark tests for the Direct I/O technology indicates that the scalability of the VERITAS File System could be very large, VERITAS Software recently teamed up with some other high performance computing companies to test this scalability. On November 15, 1996 teams from Sun Microsystems Computer Company; Fujitsu Limited (HPC Group); VERITAS Software Corporation; Maximum Strategy, Inc.; Genroco, Inc.; and Instrumental, Inc. gathered together to assemble the software and hardware necessary to provide the highest file system I/O possible with standard open-system components. The goal was to exceed 1 GB/sec file transfer I/O.

The hardware testing platform was the following:

- Server Hardware
  - 1 Sun UltraSPARC 6000 with
    - 8 167Mhz UltraSPARC CPU's
    - 1664 MB memory
    - 36 I/O card slots
- Storage Interface Hardware:

    - 36 - Genroco Ultra-Wide S-Bus SCSI host adapter cards
- Storage Hardware:
    - 4 Terabytes of RAID-5 disk arrays, provided by Maximum Strategy Gen5-S XL Disk Storage Servers. Each disk server was attached via multiple UltraSCSI channels to the UltraSCSI host adapters on the ULTRA Enterprise I/O boards.
    - Disk attachment was via the Ultra-Wide SCSI channels provided by the Genroco S-Bus to UltraSCSI host adapters

The software platform consisted of:

- Solaris 2.5.1
- VERITAS Volume Manager (VxVM 2.3)
- VERITAS File System (VxFS 2.3 Beta)
- VERITAS Software vxbench benchmark program
- Instrumental's Performance Manager (PerfStat)

The UltraSPARC 6000 was configured with 4 UltraSPARC CPU boards, each containing 2 CPUs and 832 MB of RAM, installed in 4 of its 16 slots. The remaining 12 slots were installed with the UltraSPARC I/O cards, each containing 3 S-Bus card slots. This provided a total of 36 S-Bus card slots into which the 36 Genroco Ultra-Wide S-Bus SCSI host adapter cards were installed.

Each of the Genroco SCSI adapters was attached to one of the six ports on the Maximum Strategy Gen5-S XL RAID-5 disk arrays. In this configuration each of the ports on the Gen5-S XLs appear to the VERITAS Volume Manager as a single drive, even though they actually consist of a RAID-5 array of disks. The VERITAS Volume Manager was then used to configure all 6 Gen5-S XLs, as a single RAID-0, 36 column array. Each column used a stripe width of 2 MB, presenting a full stripe size of 72 MB.

With the machine configured in this manner the testing was performed by starting 30 processes using vxbench, with each process performing I/O on one sixth of a full stripe, or 12 MB. These 30 processes would perform successive 12 MB I/O operations in parallel on a single 2 GB file in the first series of tests.

The first performance throughput numbers were measured using 30 Discovered Direct I/O threads performing reads on the same 2 GB file, multiple times. Using this method we demonstrated 960 MB/sec file system throughput. We used this same method to produce a multithreaded write test on a single large file with a throughput of 598 MB/sec.

At this point the testers determined that the throughput numbers while impressive, were wholly constrained by the raw disk speed. This determination was reached since Solaris raw I/O generated the same performance throughput as VxFS. As a result, with all I/O slots in the UltraSPARC 6000 filled, testers felt that the 1024 MB/sec barrier could be reached, by hooking up additional drive arrays to the server. In order to accomplish this several smaller disk arrays were attached to the SCSI host adapters built into the UltraSPARC I/O cards. Next multiple file I/O operations were run in parallel, by performing a single I/O operation on the combined Gen5-S XL large array, and by performing a single I/O operation for each of the smaller drive arrays. The final performance throughput measured was 1049 MB/sec while performing file system reads on multiple files.

An interesting side note. The highest performance throughput measured on a single Genroco Ultra SCSI controller was 27 MB/sec. Once the system was configured with 36 controllers in parallel, the performance throughput only decreased to 26.67 MB/sec per controller card. This demonstrates continued impressive scalability for VxFS.

Chapter

5

Quick I/O for Database

File Systems and Databases

As applications increase in size and type of workloads, some application vendors find that the standard file system design creates performance bottlenecks for their specific application workload. Database vendors in particular realize that due to the nature of their application workload, a standard file system, designed for general purpose workloads, actually introduces performance bottlenecks, simply because of the design.

As an answer to this, UNIX provides a method to completely remove the file system. This is commonly referred to as *raw I/O mode* or *raw I/O*. This mode removes the file system completely from between the application and storage devices. As a result, the application vendor must provide their own file system services within their application. Since performance is of critical importance to database servers, many installations have taken to using raw I/O for any database server installations.

UNIX raw I/O

There are three basic kinds of I/O in the UNIX OS. *Block devices*, like disks and tapes, *character devices*, like terminals and printers, and the *socket* interface used for network I/O. All of these I/O devices are insulated from the OS by device drivers. While character devices deal in streams of data traveling between applications and devices, block devices are noted for the fact that data travels between applications and devices in blocks. These block transfers are almost always buffered in the system buffer cache.

Almost every block device supports a character interface, and these are typically called *raw device interfaces*, or *raw I/O*. The difference with this interface in that none of the I/O traveling to or from the device is buffered in the system buffer cache. A limitation with this type of I/O is that depending on the device driver, the information transferred must be made in a specific block size needed by the device driver and device.

As a result, UNIX supports a raw I/O mode for applications wishing to handle the file system I/O themselves. This ability to bypass the UNIX system buffer cache allows applications to define and manage their own I/O buffer cache. Database applications benefit from this technology expressly for the reason that the standard UNIX system buffer cache policies operate in way that is inefficient for most database applications.

The standard UNIX system buffer cache policy is to remove pages from the buffer cache on a least recently used algorithm. This type of cache management seems to provide good performance for a broad range of applications. After examining the technology,

database applications have found that the performance of the database will increase if the system buffer cache employs a most frequently used caching policy.

Another database application bottleneck with file systems, is that the UNIX OS maintains a single writer / multiple reader access policy on each individual file block. This allows the OS to enforce a policy of system updates being guaranteed to be updated by a single user at a time. This would keep file blocks from being corrupted with multiple simultaneous writes. However database applications lock data updates at a much more granular level, sometimes going so far as to lock updates based upon fields in a database record. As a result, locking an entire file block for data contained in a single field slows down database updates. Bypassing the file system and using a raw I/O interface allows the database vendor to lock system writes in a manner most efficient for their application.

Using raw I/O allows a database vendor to employ an I/O system that is optimized to provide the best performance for their application. The largest problem that using raw I/O creates is the fact that raw I/O disks do not contain a file system. Therefore the data on the disks cannot be accessed using file system based tools, such as backup programs.

raw I/O Limitations

The largest single category of limitations that exist with raw I/O partitions are their management. Since the application manages all file system services, then any file system services such as backup, administration, and restoring, must be done within the application. Most of these tools treat the raw partition as one large image rather than separate files. System backups and restores must be done as a whole image, and performing maintenance on any one section of the raw system, can be very time consuming. As database servers grow in size the management problems associated with raw partitions increase.

Quick I/O for Database

The second new cache policy available with the 2.3 version of the VERITAS File System is the Quick I/O for Database. While Direct I/O improves many types of large I/O performance, the single writer lock policy of the UNIX OS creates a performance bottlenecks for some types of file system writes. Database application writes are particularly affected by this. Included in the VERITAS ServerSuite Database Edition 1.0, the Quick I/O for Database bypasses the single writer lock policy in the UNIX OS by representing files to applications as character devices. This allows database applications suited to utilizing raw partitions, the ability to operate like they are using a raw partition, on a file system. This combines the manageability of file systems with the performance of raw partitions.

The benefit of the Quick I/O for Database technology is demonstrated in the next series of benchmarks. It is important to understand that unlike Direct I/O, there are two limitations on implementing Quick I/O for Database. The first is the same as Direct I/O, the file system must be properly aligned. The second is that the file space which will be used by Quick I/O for Database must be pre-allocated which is typical for database applications.

Multiple Thread Random I/O Benchmarks

The following benchmark tests compare the multiple thread performance of VxFS Buffered I/O, Direct I/O and Quick I/O for Database with UNIX raw I/O throughput:

Multiple Thread Random I/O Read Tests

Threads	VxFS Buffered KB/sec	Direct I/O KB/sec	Quick I/O for DB KB/sec	raw I/O KB/sec	VxFS Buffered CPU/sec	Direct I/O CPU/sec	Quick I/O for DB CPU/sec	raw I/O CPU/sec
1	1748	1877	1898	1812	11.1	5	4.4	4.3
4	2858	3096	3122	2987	11.4	4.8	5.2	4.4
16	3798	4062	4145	4018	13.1	6.2	5.4	5.3

Multiple Thread Random I/O Write Tests

Threads	VxFS Buffered KB/sec	Direct I/O KB/sec	Quick I/O for DB KB/sec	raw I/O KB/sec	VxFS Buffered CPU/sec	Direct I/O CPU/sec	Quick I/O for DB CPU/sec	raw I/O CPU/sec
1	1314	1986	1983	1906	11.2	4.4	4.5	4.1
4	1430	2001	3001	2899	11.1	5	5	4.3
16	1401	1961	3887	3797	14.1	7.1	6	5.2

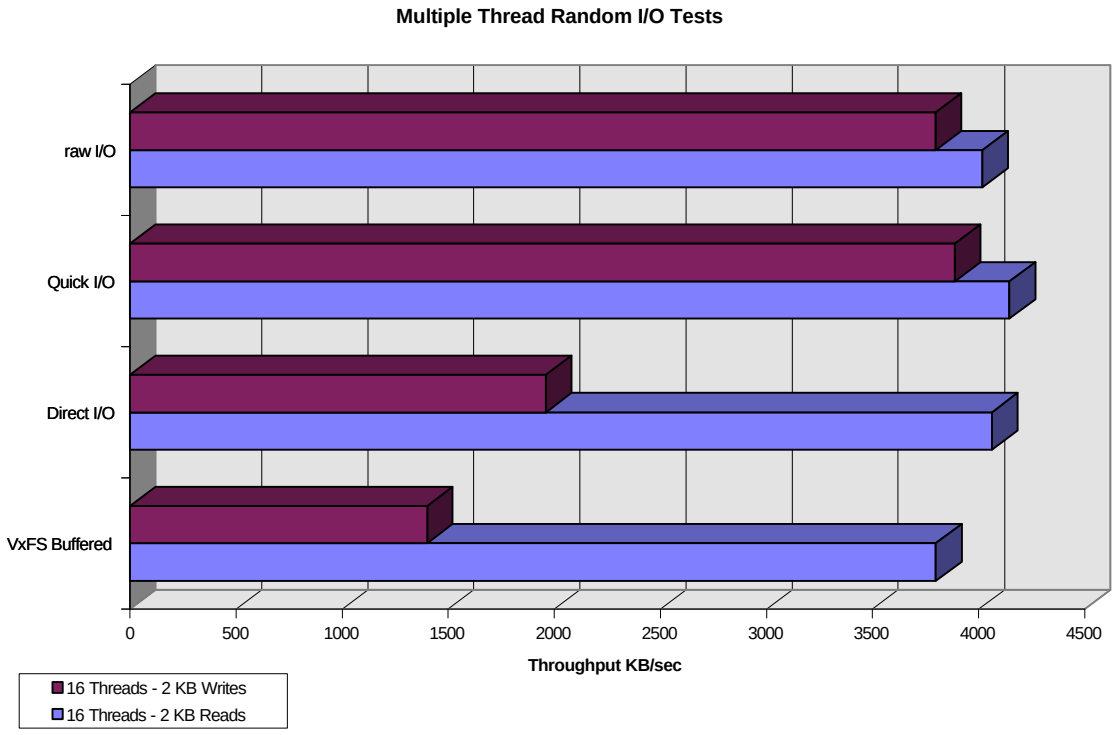
All of the multiple thread random tests were done using the default UFS and default VxFS parameters. The file size used in all iterations was 1 GB, the block size used in all iterations was 2 KB. The hardware disk configuration used was 4 SCSI controllers connected to 16 SCSI drives, creating four 8 GB RAID-0 volumes. Finally twenty 1 GB files were pre-allocated, 5 each to a single volume for performing these tests.

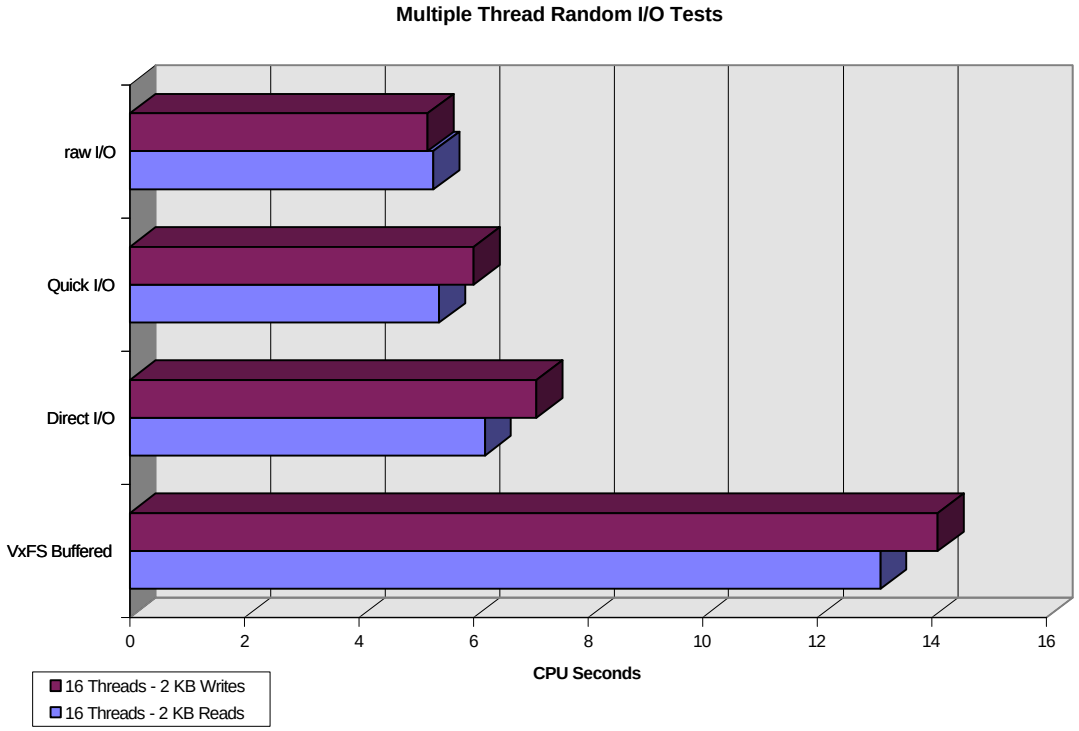
These benchmarks illustrate that for reads, all technologies provide very similar throughput. In some cases Quick I/O for Database actually provides slightly better throughput than the raw partition. This is likely due to some of the alignment features inherent in the combination of VERITAS Volume Manager and the VERITAS File System. The only big difference in these benchmarks is in the marked decrease in CPU utilization changing from buffered I/O to non-buffered I/O. Again Direct I/O and Quick I/O for Database perform on par with raw I/O.

However when switching to the write benchmark tests it becomes apparent how much of a performance cost the single writer lock policy in the UNIX OS incurs. It is important to note that these bottlenecks exist utilizing this type of I/O stream, due to the fact that I/O will queue up behind the UNIX locking.

Note that while buffered and Direct I/O reach bottlenecks in throughput at their respective levels, the Quick I/O for Database technology demonstrates impressive throughput while circumventing this system limitation. The following are the combination graphs of these results looking at the 16 thread tests:

Multiple Thread Random I/O Comparison Test Graphs





Chapter 6

Direct I/O vs. Quick I/O for Database

The final series of benchmarks focuses on the new cache policy technology for large block I/O, in the VERITAS File System. These are Direct I/O, and the Quick I/O for Database. Quick I/O for Database technology introduces impressive throughput as demonstrated in the previous multiple thread tests.

Direct I/O vs. Quick I/O for Database Benchmarks

Direct I/O vs. Quick I/O for Da tabase Multiple Thread Write Tests

Threads	I/O Transfer Block Size KB	Direct I/O KB/sec	Quick I/O for DB KB/sec	Direct I/O CPU sec	Quick I/O for DB CPU sec
1	**64**	6108	6644	0.91	1.27
4	**64**	6047	19736	2.16	2.42
8	**64**	6041	28764	2.16	2.41
16	**64**	6181	42104	2.18	2.82
32	**64**	6531	51368	2.52	3.03
1	**256**	16828	15829	0.74	0.8
4	**256**	16810	29284	1.09	1.52
8	**256**	16811	49128	0.99	2.3
16	**256**	16653	46539	1.06	2.36
32	**256**	15574	43378	0.65	2.24
1	**512**	28693	23847	0.64	0.69
4	**512**	24070	49373	1.06	1.1
8	**512**	25357	46217	0.9	1.43
16	**512**	24626	40537	0.71	1.97
32	**512**	22975	39549	0.99	2.63
1	**1024**	33813	31572	0.68	0.74
4	**1024**	32956	49435	0.77	1.11
8	**1024**	31658	46231	0.7	1.57
16	**1024**	30642	43517	0.9	2.22
32	**1024**	29958	38637	0.94	3.14
1	**2048**	36313	36355	0.63	0.71
4	**2048**	35959	45835	0.82	1.39
8	**2048**	35799	45854	0.81	1.82
16	**2048**	35305	40338	0.95	2.92
32	**2048**	33701	38255	1.26	4.4

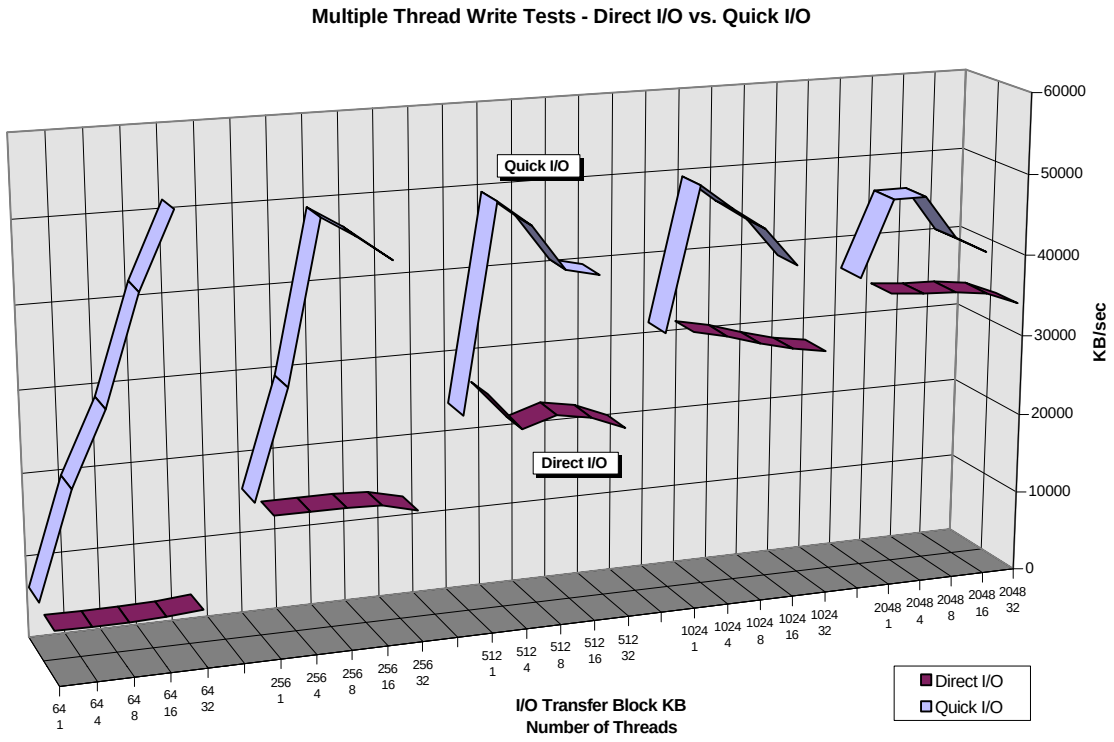
These benchmark tests continued the previous chapter testing by performing multiple thread, file system write testing, comparing Direct I/O with Quick I/O for Database for large block I/O, typical of imaging and other multimedia application workloads.

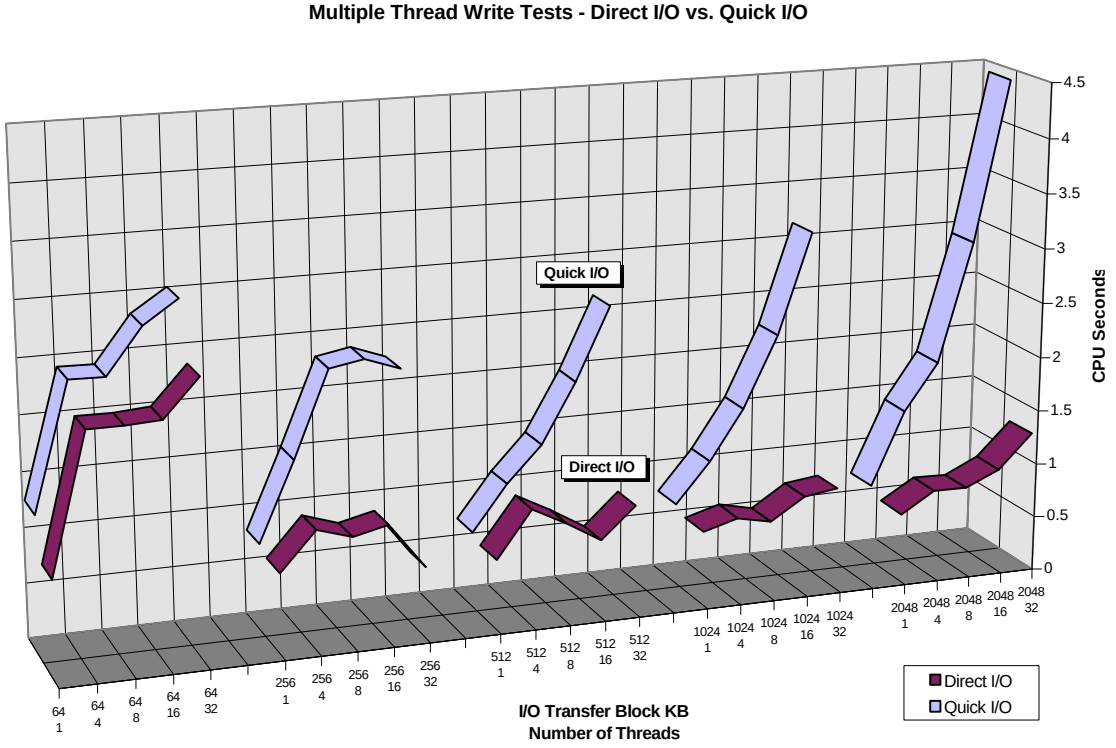
All of the final comparison tests were done using the default VxFS parameters. The file size used in all iterations was 256 MB. The hardware disk configuration used was 4 SCSI controllers connected to 8 SCSI drives, creating a single 16 GB RAID-0 volume.

Note that the performance curves begin to separate when multiple threads are used at the 4 thread level. Quick I/O for Database continues to demonstrate better throughput as the testing continues, while Direct I/O maintains a regular decrease in performance whenever the block size is smaller.

CPU utilization curves appear very similar only until the larger thread ranges are reached. There the Quick I/O for Database technology utilizes more CPU time, not for file system activity but rather this is mostly attributed to thread re-synchronization. The following are the graphs of these results:

Direct I/O vs. Quick I/O for Database Multiple Thread Writ e Test Graphs





These results demonstrate an interesting fact when it comes to Direct I/O. It is shown here that as the I/O transfer size increases, the Direct I/O throughput increases. This demonstrates that when the I/O requested gets larger the UNIX single writer lock policy affect is lessened. The concept behind this result is that when using low I/O sizes the disk subsystem is waiting for work to do based on the lock bottleneck. However as the I/O size increases, the disk subsystem has more work to do, and the bottleneck changes from the file system and balances with the disk subsystem.

This concept is further tested in the last series of benchmark tests. Here instead of performing multiple thread writes to a single file, the comparison testing of Direct I/O and Quick I/O for Database involves performing multiple writes to different files, simulating the activity of multiple applications. This is typical of the workload imposed in a multiuser server environment.

Direct I/O vs. Quick I/O for Database M ultiple Application Write Tests

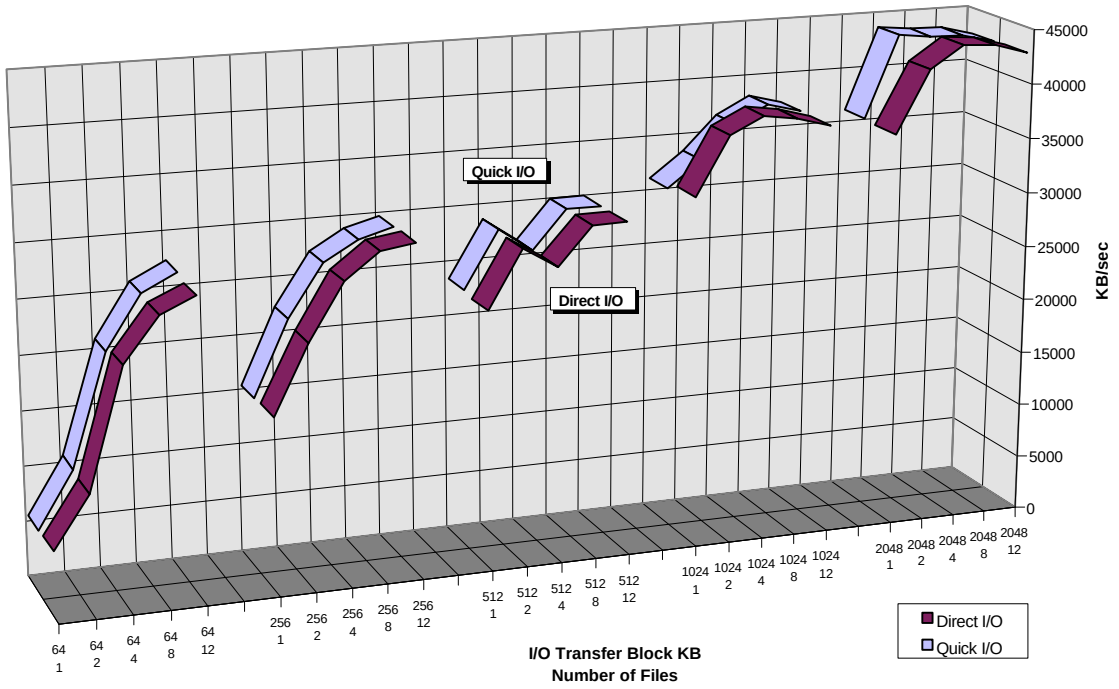
Files Written	I/O Transfer Block Size KB	Direct I/O KB/sec	Quick I/O for DB KB/sec	Direct I/O CPU sec	Quick I/O for DB CPU sec
1	**64**	6122	5875	0.89	1.15
2	**64**	10803	10989	2.31	2.23
4	**64**	21718	21176	4.93	5.02
8	**64**	25750	26027	10.29	10.39
12	**64**	27154	27581	15.58	15.38
1	**256**	15826	15681	1.06	1.07
2	**256**	22058	22534	1.53	1.58
4	**256**	27187	27289	3.11	3.49
8	**256**	29542	29084	6.8	6.94
12	**256**	30014	29944	10.31	10.38
1	**512**	23400	23683	0.71	0.68
2	**512**	28667	28878	1.34	1.53
4	**512**	26731	26716	2.75	2.93
8	**512**	30202	30199	6.26	5.92
12	**512**	30242	30223	9.66	9.62
1	**1024**	31935	31335	0.54	0.76
2	**1024**	37247	33647	1.27	1.14
4	**1024**	38730	36731	3.05	2.91
8	**1024**	38269	38260	5.93	5.49
12	**1024**	37413	37454	8.39	8.55
1	**2048**	36134	36313	0.61	0.68
2	**2048**	41792	43808	1.32	1.43
4	**2048**	43821	43474	2.76	2.89
8	**2048**	43575	43398	5.79	5.51
12	**2048**	42729	42627	8.65	8.32

This time the performance curves are almost identical. An interesting result considering the different technologies. This certainly demonstrates that for large I/O in a multiuser environment, utilizing Direct I/O technology, the UNIX single writer lock policy is less of an issue on overall system throughput.

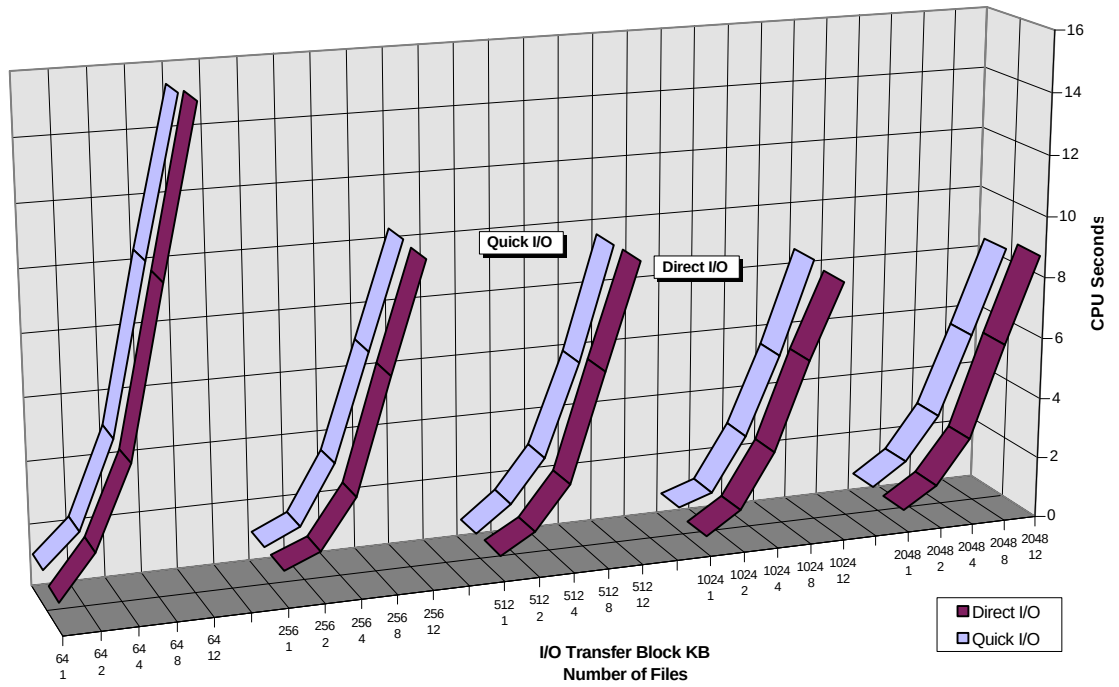
The following are the graphs of these results:

Direct I/O vs. Quick I/O for Database Multiple Application Write Test Graphs

Multiple Application Write Tests - Direct I/O vs. Quick I/O



Multiple Application Write Tests - Direct I/O vs. Quick I/O



Chapter 7

Conclusion

This performance report focused on the performance of the 2.3 version of the VERITAS File System. Included here has been a discussion of the key performance components in the VERITAS File System. This report also presented a series of benchmark tests comparing the throughput and CPU utilization of different VERITAS File System component technologies, as well as some tests comparing the VERITAS File System (VxFS) with the Solaris UNIX File System (UFS).

It is clear that the VERITAS File System presents technologies for improved performance at the smaller I/O sizes with features such as extent based allocation. The VERITAS File System also presents technologies for advanced performance with large I/O using the Direct I/O technology.

With the release of the 2.3 version of the VERITAS File system both buffered and Direct I/O performance can be combined in one file system with the Discovered Direct I/O feature. For database implementations the Quick I/O for Database provides throughput very close to that of database servers running on raw partitions.

This report outlined the performance advantages and system scalability of the VERITAS File System. Using the VERITAS File System and a Sun UltraSPARC server, VERITAS has been able to generate file system performance throughput in excess of 1 GB/sec.

In closing, VERITAS Software presents software technology that provides commercial class performance, availability, and manageability. This report presents a description of the very powerful performance component of VERITAS Software. Once the performance component is understood, it is important to realize that this performance comes not as a result of a standard UNIX file system, but comes coupled with the highly available, *journaled*, VERITAS File System.