Windows 2000 Disk IO Performance

Leonard Chung,
Jim Gray,
Bruce Worthington,
Robert Horst

June 2000

Technical Report
MS-TR-2000-55

Microsoft Research
Advanced Technology Division
Microsoft Corporation
One Microsoft Way

Redmond, WA. 98052

Windows 2000 Disk IO Performance

Leonard Chung, UC Berkeley, Leonard@ssl.berkeley.edu
Jim Gray, Bruce Worthington, Microsoft, {Gray, BWorth}@Microsoft.com
Robert Horst, 3Ware Inc., Bob.Horst@3ware.com
28 April 2000

Abstract

This paper is an empirical study of the random and sequential I/O performance of Windows 2000™ using the NT File System. It continues the work done by Riedel, et. al. in their 1997 paper exploring sequential IO performance under Windows NT 4.0™. This paper explores the performance and overhead of today's Intel Pentium III™ based memory and IO subsystems, including the processor bus, the PCI bus, the disk controllers, the SCSI and IDE buses, and the disk media. It also examines the performance available from IDE RAID, a relatively new technology built from inexpensive IDE disks. Network IO performance is briefly covered. We describe how to achieve good throughput, and discuss the major bottlenecks. While Riedel's model of software performance remains largely unchanged, increases in hardware performance have made the 32bit, 33MHz PCI bus the dominant bottleneck reaching saturation with three disks.

Table of Contents

1 Overview

Much has changed in the three years since Riedel's 1997 study of sequential I/O performance [Riedel]. Disk capacities have increased; today's biggest hard drives are over four times larger than the largest drives available then. Single disk sequential I/O throughput has also improved with 10,000 RPM drives. Processors have increased in speed and number. SMP is now available for the desktop. Memory and system buses have improved. SCSI has also improved over the last two years with Ultra160 SCSI promising a four-fold improvement in adapter to disk bandwidth. With today's technology, we were able to achieve almost 24 MBps of read throughput on a single disk. Write throughput on a single disk peaked out at 22.5 MBps. This represents a 2.5 to 3 times imp rovement over the throughput Riedel measured. The disks used for our study were representative of drives commonly available at this time but not necessarily the highest performance disks on the market. The fastest drives now exceed 30 MBps in sequential transfers.

The software landscape has also changed, as Windows 2000 is replacing Windows NT4. Windows 2000 introduced *dmio*, a new volume manager, to replace *ftdisk*. Despite the fact that *dmio* and *ftdisk* are very different in implementation, our IO performance measurements show only minor performance differences between them. Processor overhead for *dmio* is somewhat higher so a processor-bound application might see a slightly different picture.

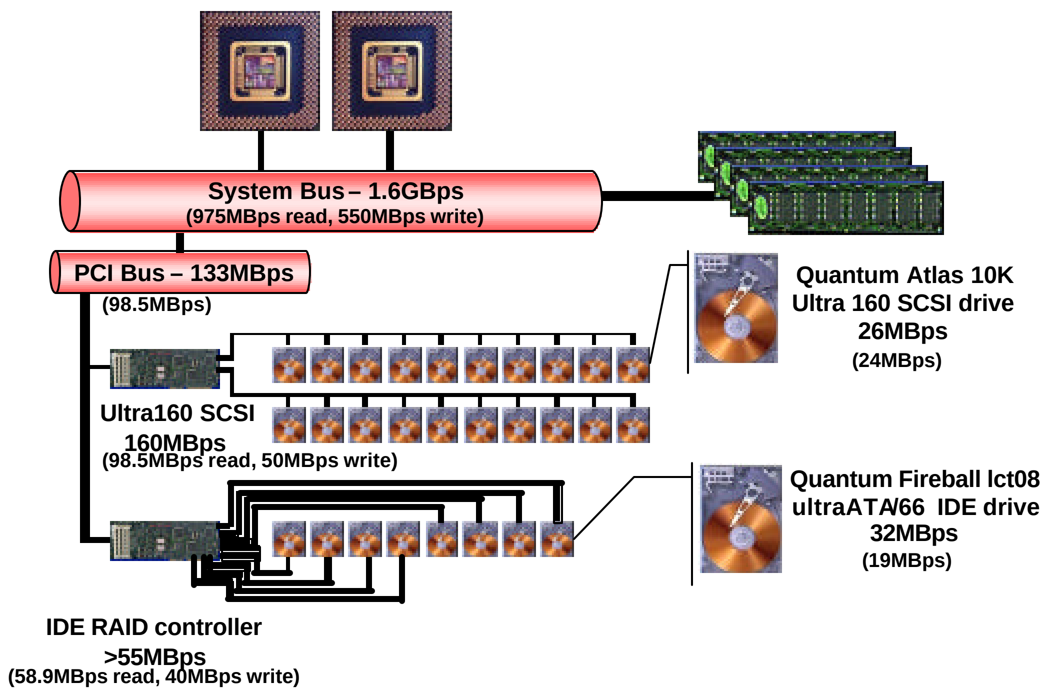


Figure 1 – Peak Advertised Performance (PAP) vs. Real Application Performance (RAP). The graphic above shows the various hardware components involved with disk IO: the processors, memory/system bus, PCI bus, SCSI and IDE buses, and individual disks. For each component, the PAP is shown in bold while the RAP we measured is shown below in parentheses.

We first measured single disk sequential I/O performance on NT4SP6 and Windows 2000 Advanced Server. We then compared our results to those of the original sequential I/O study of NT4SP3 by Riedel, et. al.

Performance of all three operating systems was similar. Windows NT4SP6 and Windows 2000 Advanced Server have very similar performance, and show almost identical performance to NT4SP3 except:

- The overhead for large buffered read and write requests was substantially higher on NT4SP3.
- Small (2KB and 4KB) requests no longer show the 33% to 66% decrease in throughput seen in NT4SP3.
- The buffered IO throughput "dip" of NT4SP3 above 64KB is corrected.

Sequential I/O performance under NT4SP6 shows a few improvements compared to NT4SP3. Differences were incremental rather than radical: the models of NT performance are still valid. Win2K compared to NT4 similarly shows incremental improvement in I/O performance. Basic volumes and the new dynamic volumes have similar sequential I/O performance.

With the sequential IO throughput of disks and controllers increasing, the bottleneck has shifted to the one thing that has not improved much: the PCI bus. Our modern workstation was capable of 98.5MBps across its PCI bus. When compared to the 72MBps Riedel was able to achieve, our workstation's PCI bus is only 37% faster while its disks are 300% faster. What this means is while two years ago it would take nine disks spread over three adapters to saturate a PCI bus, today three to four disks on one adapter can saturate a PCI bus. The multiple 64-bit 66 MHz PCI buses found on high-end servers, and the future Infiniband™ IO interfaces will likely change this, but today the PCI bus is a bottleneck for sequential IO on low-end servers.

Of course, most applications do random rather than sequential IO. If applications are doing random 8KB IOs against modern disks, then each disk can deliver about 1MBps, and so a modern controller can manage many (16 or more) disks, and a single PCI bus can carry the load of 64 randomly accessed disks. Faster PCI technologies such as PCI-X and the 64bit / 66MHz flavors of PCI are still premium products.

Along with the advances in SCSI drive technology, IDE drives are beginning to grow beyond the desktop market into the workstation and server markets, bringing with them considerably lower prices. The ANSI standard ATA (Advanced Technology Attachment) interface, more commonly called Integrated Drive Electronics (IDE), was first created to provide cheap hard drives for the PC user. As the Intel x86 architecture has become more popular, price conscious consumers have purchased IDE drives rather than pay a premium for drives with more expensive interfaces such as SCSI. Today, IDE drives have evolved to include DMA and 66MHz (500Mbps) connections and still hold a wide margin over SCSI drives in terms of units shipped. With their higher volume, IDE prices benefit from economies of scale. At present, a terabyte of IDE drives costs $6,500 while a terabyte of SCSI drives costs $16,000. When optimizing for cost, IDE drives are hard to beat.

Of course, cost is only one of the factors in purchasing decisions. Another is undoubtedly performance. Since IDE was designed as an inexpensive and simple interface, it lacks many SCSI features like tagged command queuing, multiple commands per channel, power sequencing, hot-swap, and reserve-release. The common perception of IDE is that it should only be used when performance isn't critical. Conventional wisdom says that SCSI is the choice for applications that want high performance or high integrity. As such, most desktops and portables use IDE while most workstations and servers pay the SCSI price premium. This is despite the fact that most drive manufacturers today use the same drive mechanism across both their IDE and SCSI lines – the only difference is the drive controller.

It is possible to mitigate some of IDE's performance penalties by using a host bus adapter card that makes the IDE drives appear to be SCSI drives. Indeed, Promise Technology? and 3ware? are two companies that sell such cards. These cards add between $38 and $64 to the price of each disk. These controller cards are typically less expensive than corresponding SCSI controller cards, so they potentially provide a double advantage – SCSI functionality at about half the price.

Among other things, this report compares IDE and SCSI performance using micro-benchmarks. We used a 3ware 3W-5400 IDE RAID card to allow us to connect four IDE drives to our test machine. In summary, we found that individual IDE drive performance was very good. In our comparison of an SCSI Quantum Atlas 10K 10,000 RPM drive to an IDE Quantum lcs08 5400 RPM drive, the IDE drive proved to be 20% slower on sequential loads and at most 44% slower in random loads. But the SCSI drive was more than 250% more expensive than IDE drive. Even with the lower random performance per drive, buying two IDE drives would be cheaper, and a mirrored pair would give both fault-tolerance and read IO performance superior to a single SCSI drive. The IDE price/performance advantage gets even better for sequential workloads. For the same price as SCSI, IDE delivers almost double the sequential throughput of a single SCSI disk.

However, SCSI features like tagged command queuing, more than two disks per string, long cable lengths, and hot swap don't currently exist for native IDE – although 3ware promises hot-swap in their next model (3W-6000)

The report also examined the performance of file IO between a client and a file server using the CIFS/SMB protocol, either as a mapped drive or via UNC names. These studies showed that clients can get about 40 MBps reading and half that writing. But, there are several strange aspects of remote file IO. Unfortunately, there was not time to explore the details of this problem.

In summary:

Yesterday's IO Performance:
- Smaller, slower disks
- SCSI bus saturation can be reached by a small number of disks
- PCI bus saturation requires many disks.

Today's IO Performance:
- Disks are four times bigger and three times faster.
- SCSI buses now have a higher advertised bandwidth than PCI so now…
- For sequential loads, 32bit 33MHz PCI is now the major bottleneck. PCI bus saturation can be achieved with only three disks, a configuration not uncommon on workstation machines.
- Random loads aren't affected as they don't come anywhere near PCI bus saturation.

IDE drives:
- Best price/performance ratio.
- Lack many features like command queuing, long cables, power sequencing…
- Some features such as command queuing and hot swap, can be done by the controller card.

SCSI drives:
- Roughly 2.5x more expensive than IDE.
- Have the best performance: 1.7x better on random, and up to 1.2x better on sequential when comparing 10,000 RPM SCSI drives to 5,400 RPM IDE drives.
- Have features lacking in IDE, such as being able to connect 15 devices to one string and being able to hot swap devices, fiber channel connection, tagged command queuing, power sequencing, ….
- SCSI RAID is a more mature technology and widely supported by hardware vendors.

2 Introduction

We first sought to measure and compare the I/O performance on a workstation similar to that of Riedel's running NT4SP6 and then Windows 2000 to compare performance between the two operating systems. We called these the old-old and old-new tests: signifying that we were testing old hardware and the old OS, and old hardware with the new OS. We then tested a modern workstation and the Windows 2000 operating system. These tests were called the new-new tests. The software and hardware test naming matrix is shown in Table 1.

The first old-old and old-new measurements were conducted on hardware similar to that used in the original study. The processor is faster, but the disks and controller that were the main bottlenecks in the original report, remain the same. We measured both Win2K and NT4SP6. Our objective for these tests was twofold: first, to compare the results of the original study by Riedel, et. al. with the most recent version of NT4 on comparable hardware to see if there have been any changes in performance. Second, to explore the differences and similarities between Windows NT4SP6 and Win2K.

The new-new measurements were taken on a Dell Precision 420 test machine with the latest processors, memory, and SCSI disks (When we started in January 2000): Dual Intel Pentium III processors running at 733 MHz, Rambus memory, an Ultra160 SCSI adapter, and four 10K RPM SCSI drives. Each of these has an advertised bandwidth of 1.6GBps, 160MBps, and 18 to 26MBps respectively. We also added a 3ware 3W-5400 IDE RAID card, along with four Quantum Fireball lct08 5400 RPM drives with an advertised internal throughput of 32MBps each.[1] PAP (Peak Advertised Performance) however is often is quite different than RAP (Real Application Performance). We wanted to explore what kind RAP today's hardware is actually able to achieve, along with how to achieve good performance with minimal effort.

	Software	**Hardware**
Old	Windows NT4SP6	333 MHz Pentium II 4 GB 7200 RPM Ultra-Wide SCSI drives (RAP: 9MBps per drive)
New	Windows 2000	2 x 733 MHz Pentium III 18GB SCSI 10,000 RPM Ultra160 SCSI drives (RAP: 24MBps per drive) 27GB 5,400 RPM UltraATA/66 IDE drives (RAP: 19MBps per drive)

Table 1 – The experiments.

To allow price comparisons, here are the prices we paid for the various components.

Dell Precision 420	$3,750
Dual Channel SCSI controller	$235
Quantum Atlas 10K Ultra160 SCSI 18 GB disks	$534
3ware 3W-5400 IDE RAID adapter	$255
Quantum Fireball lct08 ATA/66 26GB disk	$209

Table 2 – Prices of hardware components

[1] Vendors typically quote internal and external data rates, which are burst rates. These can't necessarily be directly correlated to sustained throughput, however the internal data rate is generally the number closest to being an actual advertised throughput.

2.1 Hardware Configuration

Unless otherwise noted, all of the old-old and new-old tests were run on the following hardware. Note that the Riedel study used a 200 MHz Pentium II.

Table 4 – "Old" machine hardware configuration.

Host	*Gateway E-5000*						
	Processor: 333 MHz Pentium II RAM: 64-bit wide, 66 MHz memory interconnect 1 x 128 66 MHz SDRAM Bus: 32-bit wide, 33 MHz PCI Host bus adapter: Adaptec 2940UW Ultra-Wide SCSI adapter IDE controller: 2 independent PCI bus mastering interfaces						
Disk	**Name**	**Interface**	**Capacity**	**RPM**	**Seek Time**	**Transfer Rate**	**Cache Size**
	Seagate Barracuda 4LP Ultra-Wide (ST34371W)	SCSI-2 Ultra-Wide ASA II	4.3 GB	7200	Avg 4.2ms range 1-17	**External** 40 MBps **Internal** 10 MBps to 15 MBps	512 KB
Software	Old: Microsoft Windows NT 4.0 SP6 using the NT file system						
	New: Microsoft Windows 2000 Advanced Server using the NT file system						

Unless otherwise noted, all of the new-new tests were run on the following:

Table 5 – "New" machine hardware configuration.

Host	*Dell Precision 420*						
	Processor: 2 x 733 MHz Intel Pentium III RAM: 64-bit wide, 133 MHz memory interconnect 2 x 128 ECC PC800 RDRAM Bus: 32-bit wide, 33 MHz PCI Host bus adapter: Adaptec AIC-7899 Ultra160/m SCSI adapter 3ware 3W-5400 IDE RAID adapter IDE controller: 2 integrated bus mastering interfaces						
Disk	**Name**	**Interface**	**Capacity**	**RPM**	**Seek Time**	**Transfer Rate**	**Cache Size**
	Four Quantum Atlas 10K (QM318200TN-LW)	Ultra160 Wide LVD	18.2 GB	10,000	Avg 5.0ms	External 160 MBps Internal 18 to 26 MBps	2 MB
	Four Quantum Fireball lct08	Ultra ATA/66	26.0 GB	5,400	Avg 9.5ms	External 66.6 MBps Internal 32 MBps	512 KB
Software	Microsoft Windows 2000 Workstation NT file system. SQLIO for basic volume striping experiments, Windows 2000 *dmio* RAID for dynamic volume striping experiments. The 3ware controller's hardware RAID was used for striped and mirrored dynamic volume IDE experiments.						

3 Device Internals Performance

In this section, we examine the performance of some of the internal subsystems in the new-new Dell Precision 420 test machine.

3.1 System Memory Bus Throughput

System memory bandwidth was measured using *memspeed*. *memspeed* is covered in detail below, in the Testing Methodology section. The results are shown in Figure 1 and Table 6. Rambus RAM is advertised as being capable of a throughput of 1,600MBps. [Rambus] However, on our test machine we were only able to achieve 975MBps on reads and 550MBps on writes. This represents 61% and 34% of the PAP respectively. Compared to what we measured on previous Intel systems, this represents a huge 5x advance in read bandwidth and 3x advance in write bandwidth.

3.2 SCSI and PCI bus Throughput

The RAP for our 32bit, 33MHz PCI bus was 98.5MBps, a full 74% of the PAP of 133MBps when 1MB requests were sent to the controller cache. This is 37% more throughput than Riedel was able to achieve on his machine. PCI chipsets have clearly improved. When smaller 64KB requests were used, the RAP was 83.6MBps.

Ultra160 SCSI advertises itself as a 160MBps bus. However, even under ideal conditions, our Ultra160's PAP is unachievable. The standard 32bit, 33MHz PCI bus found in PCs only has a PAP of 133MBps. This limits Ultra160 adapters to 133MBps at most. In practice, our PCI bus never actually achieves 100% of the PAP so our Ultra160 adapter was limited by its PCI bus interface to 98.5MBps. Even so, its RAP was a respectable 62% of the PAP.

3.3 IDE controller and Throughput[2]

The 3ware 3W-5400 is a PCI card that supports four IDE/ATA drives. The newer 3W-5800 card supports up to 8 drives. Each IDE drive is set to master and given its own string. The 3ware card and its driver software presents the IDE drives to the Windows or Unix host as SCSI drives. The drives can be presented to the system as either just a bunch of disks (JBOD), or as large logical disks through RAID0 and RAID1. Since the 3ware card offloads much of the IO processing from the CPU, the processor overhead was similar to that seen on SCSI adapters.

All of our IDE measurements were taken with WCE as the 3W-5400 enables the drive write cache automatically.[3] Unlike most SCSI controllers, the current 3ware card only allows WCE to be disabled on a per-request basis. At the time of our tests, it did not allow WCE to be disabled globally.

It is possible to measure the PCI and controller throughput by reading from the controller cache, rather than going to the disk media. We measured the peak throughput using the tool *DiskCache* program described in the *Testing Methodology* section. By reading directly from disk cache from four Fireball IDE drives, we were able to achieve 58.9 MBps from the 3ware card using 64KB requests. This is the card's PCI limit and larger request sizes had no effect. Other experiments with SCSI controllers delivered as much as 84MBps. The first generation of 3ware cards

---

[2] We had initially intended to test the Promise FastTrak66 IDE RAID card in addition to the 3ware card, but were unable to do so. Although the FastTrak66 functioned correctly as a basic UltraATA/66 IDE controller, we were unable to use the RAID features offered by the card on Windows 2000. We were unable to do even basic operations such as formatting of RAID volumes within Win2K. Promise's technical support told us Windows2000 is not yet supported. They are currently working on these problems, but the fixes were not yet available.

[3] Write Cache Enable(WCE) is a drive parameter that allows the drive to acknowledge completion of a write without actually writing the bits to disk. Instead, the bits are held within the drive's cache, allowing it to schedule, coalesce, and otherwise optimize the write requests to improve throughput. WCE improves write performance at the cost of reliability: if power to the drive is lost, then the contents of the controller cache are lost and the bits are never written to disk. This can lead to data corruption. However, software can force individual requests to be written to the media for operations that could otherwise cause data corruption after loss of power.

were limited by their PCI implementation. Second generation 3W-6000 series cards, which were not available in time for this study, have a higher PCI throughput. Tests performed on a 3W-6800 show a peak read throughput of 101 MBps and peak write throughput of 85 MBps read throughput when accessing four Fireball IDE drives. For random workloads, the bandwidth demand of 4 drives is less than 5MBps, so the PCI bus speed is not an issue.

	Peak Advertised Performance	**Real Application Performance**
Memory bus	1,600 MBps	975 MBps read 550 MBps write 327 MBps copy
PCI bus	133 MBps	98.5 MBps
SCSI bus (Ultra160)	160 MBps	98.5 MBps @ 1 MB read req. 83.6 MBps @ 64KB read req. 50 MBps write
3ware 3W-5400	Greater than 55MBps	58.9 MBps @ 64KB read 40 MBps write
3Ware 3W-6800* (Alpha test unit)	100+ MBps	101 MBps @ 256KB read 85 MBps @ 256KB write
Hard disk (Atlas 10K)	18MBps to 26MBps	17 MBps to 24 MBps
Hard disk (Fireball lct08)	Up to 32 MBps	12.4 MBps to 19 MBps

Table 6 – PAP vs. RAP across multiple buses. The SCSI bus's throughput was hampered by the adapter's PCI bus interface, as the Ultra160 SCSI's advertised PAP is higher than that of the PCI bus. The first generation 3ware card, despite being able to interface with four drives, has a limited throughput of 58.9MBps, or approximately the combined throughput of three of our Fireball IDE drives reading sequentially. However, in each of these cases, the half-power point was achieved.

3.4 Symmetric Multiprocessing (SMP)

All of our tests were run on a dual processor system. We found SMP had little effect on our throughput measurements. Since the measurements are already IO bound, increasing the number of processor cycles available does little to improve performance. We tested the single CPU case by booting Win2K using /NUMPROC=1 as an argument. This causes Win2K to function in a uniprocessor mode, ignoring the second processor. Overhead increased slightly when two CPUs were used as there was now some additional overhead incurred by synchronization code. In addition, Windows 2000 only allows one outstanding deferred procedure call (DPC) per SCSI adapter. We only had one adapter. For each of these reasons, we weren't able to take advantage of the additional parallelism offered by the additional processor.

3.5 *DMA vs. PIO*

IDE controllers have two protocols to transfer data to and from the disk to main memory: direct memory access (DMA) and programmed input/output (PIO). The main difference between them is that while PIO requires the processor to poll the drive repeatedly and places the burden of copying the data on the processor, DMA allows the processor to just initiate a transfer and places the burden of copying the data on the controller. While the controller is copying the bits, the processor can perform other tasks. PIO is 85 times more expensive than DMA – that's 8,500%!

DMA and PIO can be selected through the Device Manager in Win2K. We connected one IDE drive to an IDE channel on the motherboard and measured the affects of DMA and PIO. Table 7 shows the dramatic performance penalty of PIO. Not only did throughput decrease by 75% for unbuffered 64KB reads, but also processor utilization jumped to 50% across both processors. The first step to good IDE performance is to ensure that DMA is enabled. Since the 3ware IDE RAID card masquerades as a SCSI adapter, DMA and PIO selection was not necessary.

	DMA	**PIO**
64KB unbuffered read throughput	19 MBps	4.3 MBps
Average CPU utilization	3% of two 733Mhz processors ~ 2 cpu clocks per byte	50% of two 733Mhz processors ~ 170 cpu clocks per byte

Table 7– DMA vs. PIO – 64KB unbuffered reads. DMA delivers greater throughput with less CPU utilization. DMA needs to be enabled for good performance. Since IDE RAID cards masquerade as SCSI controllers, they don't require the user to specify DMA or PIO.

4 Testing Methodology

This section examines how these measurements were done along with issues we encountered.

4.1 Throughput measurement

SQLIO served as the primary test program for our measurements. *SQLIO* is a workload generation and measurement tool that allows the user to specify for how long a file should be read or written and various parameters about how the IO should be done. Along with measuring throughputs, *SQLIO* outputs system and kernel timing information. *SQLIO* is used by Microsoft's NT Performance Group for their tests. *sqlio2* is a complete rewrite of *SQLIO.* Among its features are the ability to plug in IO "personalities" which are modules specifying different ways of doing disk IO, a cleaner code base, and more flexible test parameter specification and result output.

In order to ensure the uniformity of results, each disk was first formatted before test files were copied to the disk in the same order. This made certain that our tests were always run from the same outer tracks on each disk, reducing the problem of media banding and maximizing test results.

We also scrubbed the file system cache between buffered runs in order to ensure our requests were going to disk instead of cache. Unless otherwise noted, *SQLIO* was used to generate all of the results in this paper except for buffered tests. *SQLIO* was found to underreport the throughput of buffered reads during testing so *cacheflush* was used for all buffered measurements. *SQLIO* was modified to generate a trace file that is easily imported to an Excel spreadsheet or relational database. The format of the trace file is a comma-separated list where each line has the format:

Field	**Description**
MB/s	I/O throughput in MB/s
IO/s	I/Os per second
Operation	Specifies reading or writing (read/write)
BlockSize	Request size in KB
OutstandingIOs	Depth of asynchronous I/Os (0 = Synchronous)
Buffering	0 and Y - File system and controller caching S –File system caching only (FILE_FLAG_WRITE_THROUGH) H – Controller caching only (FILE_FLAG_NO_BUFFERING) N – No caching (FILE_FLAG_WRITE_THROUGH | FILE_FLAG_NO_BUFFERING)
NumThreads	Number of threads
Time	User specified time to run
FileSize	Size of the file being read/written in KB
Clock(s)	Actual amount of time the test took. Used for calibration purposes.
User(ms)	Total user mode processor time as reported by GetProcessTimes()
Kernel(ms)	Total kernel mode processor time as reported by GetProcessTimes()
Time Stamp	Time and date the test was run
Comment	String containing a user specified comment. (no commas allowed)

Table 8 – The format of the SQLIO trace file, a comma separated values (CSV) file.

cacheflush is a simple program which does a buffered read of a file and then quits. It measures throughput and processor consumption, and outputs a tracefile in the same format as *SQLIO.* In earlier tests, we often saw erratic measurements that would alternate between throughputs higher than the maximum throughput of a drive to throughputs below. This was because each test was somewhat dependent upon those before it. The results were thrown off since the system was sometimes able to retrieve part of the file from cache. Our solution to the problem was to fill up the cache using buffered reads from a scratch file. As *cacheflush* read in the scratch file, the FS buffer was cleaned of any lingering data from the previous test. After *cacheflush* was run before each buffered test, the erratic measurements disappeared.

Tests were automated using batch files generated by a program named *cmdgen*. *cmdgen* takes as its input the environmental variable *sqlio_note* which contains comments about the current test target. It then outputs a batch file containing multiple calls to *SQLIO* varying the arguments along with appending information about buffering and synchronism.

We measured the system memory bus using *memspeed*. *memspeed* has eight different tests by which it measures the memory bus bandwidth. 10MB is read, written, or copied for each test. In order to calibrate the instruction overhead, before each test a null test was run where no actual reads or writes took place. This instruction overhead is then subtracted from each subsequent test's time. Reads and writes are measured using two methods: the first method read and wrote 64 bits at a time using doubles. In the read case, the 10MBs are read from memory into a register double. In the write case, the register double is repeatedly written to main memory. The second method used the file system calls WriteFile() and ReadFile() to read and write 64KB at a time. A cached file was created using FILE_FLAG_DELETE_ON_CLOSE and FILE_ATTRIBUTE_TEMPORARY. In the read case, bits were read from this cached file into a 64KB buffer. In the write case, bits were written from the buffer into the cached file. The last test used memcpy() to copy data between two 10MB buffers. On multiprocessor machines, the double read/write tests and the memcpy() test were rerun with multiple threads. The number of threads was equal to the number of CPUs.

memspeed was also used to measure the SCSI, IDE, and PCI bus saturation points by reading a small amount of data on a disk repeatedly using two-deep requests. On the first read, the drive must access the disk in order to fulfill the request. This data is then placed in the drive cache. On subsequent requests, the drive can fulfill the request straight from its cache without going to disk. Tests were run on 1 to 4 disks. This allowed us to measure the burst throughput of single and multiple drives, along with quantifying throughput limitations of the SCSI, IDE, and PCI buses.

The programs we used for workload generation and measurement are available online, along with the raw test results, at http://research.microsoft.com/BARC/sequential_IO/

4.2 Overhead Measurement

The Win32 API has a function called GetProcessTimes() which extracts timing information from a process. The user time and kernel time consumed by a process are among the information it returns. We originally used GetProcessTimes() as our only overhead measurement. However, it became clear that GetProcessTimes() often understated the actual processor cost that was being incurred, because GetProcessTimes() only returns the processor cost which was *charged* to that process. Measuring the cost per IO is complicated by the fact that not all of the processor time used is charged to processes. During certain situations, such as interrupt time, processor cycles aren't charged to any process. Those cycles are unaccounted for by GetProcessTimes(). Measurements taken with GetProcessTimes() are the lower bound of the range of actual CPU overhead.

To get the upper bound, we measured system idle time. We measured idle time by using a soaker: a thread or set of threads running at idle priority that are set in a busy loop. They try to consume all available idle cycles, effectively becoming the idle process. By first measuring the soaker's processor time while the system is idle, then measuring the soaker's processor time while running a test, and finally calculating the delta, we can subtract out the background processor overhead and find the processor time being consumed by IO generation and the previously unaccounted processor time associated with the IO requests. Since the soaker looks only at total system idle time rather than processor consumption by individual processes, the soaker does not return results for a single process alone, but may include processor time used by other applications. The overhead reported by the soaker is the upper bound on CPU overhead. Unless otherwise noted, all of the overhead graphs shown are taken using a soaker.

At small request sizes, the upper bound can show a significant difference in the measured overhead from the lower bound. However, at larger request sizes these upper and lower bounds generally converge.

4.3 Zoned Disk and Variable Media Rates

Modern disks are zoned: tracks have a variable number of sectors and therefore a variable size rather than a fixed size. Each track's size is a function of its circumference. Since tracks on the outer edge of the media have a larger circumference than tracks on the inner edge, outer tracks have more bits traveling under the disk head per second than inner tracks due to the disk's constant angular velocity. The Ziff Davis WinBench 99 was used to generate Figure 2 that shows a throughput map of the disk as the head is moved from the outer to the inner tracks [ZD]. The stair stepping is due to banding on the media. Each band has the same number of sectors per track. Within each band, the throughput remains the same. From band to band however, the sectors per track changes, resulting in a sudden change in throughput. Figure 2 also shows that there is considerable variance among read speeds ranging ? 10%.

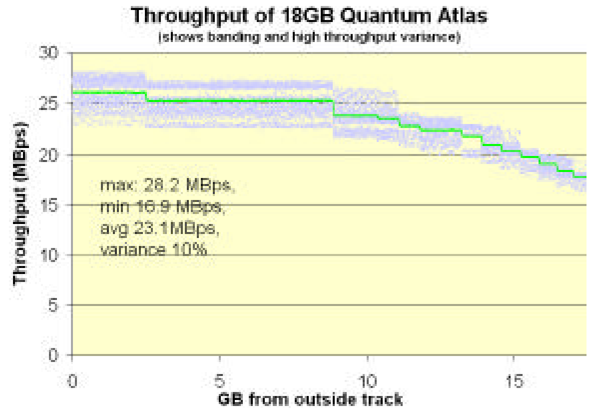


Figure 2 – Read throughput variation across the Quantum Atlas 10K surface. The read throughput on the inner band of the media is significantly less than the outer band. The inner band on the Atlas 10K has 35% less throughput than on the outer band. Measurements taken on the Fireball lct08 showed its inner band had 40% less throughput than on its outer

4.4 File Pre-allocation

For security reasons, Windows NT4 and Windows 2000 force writes to become synchronous if writing to a new file, writing to a newly extended area of a file, or writing beyond the end of a file. This is because C2 security requires a value of zero to be returned when reading data that has not been previously written. In order to prevent users from allocating storage and then reading its previous contents, like NT4, Windows 2000 must balance the need for performance with the need for security.

As Figure 3 shows, file extension severely impacts asynchronous unbuffered write performance -- unless writes are appended to the end of the file or the writes are to a pre-allocated file, write throughput decreases by 50%. Buffering can help reduce the impact of file extension, but for applications concerned with write throughput, files should be pre-allocated or writes should be to the end of the file. The figure shows that programs sequentially writing a file (writing at EOF) get the full disk speed. But, if a program first writes the last block and then sequentially writes the file, the write proceeds at half the speed because Windows2000 first zeros the file, and then writes it. When growing a file, it is important to write it in sequential order. Otherwise Windows writes each block twice: once to zero the "gap" and once to write the actual data when the need write finally is issued.

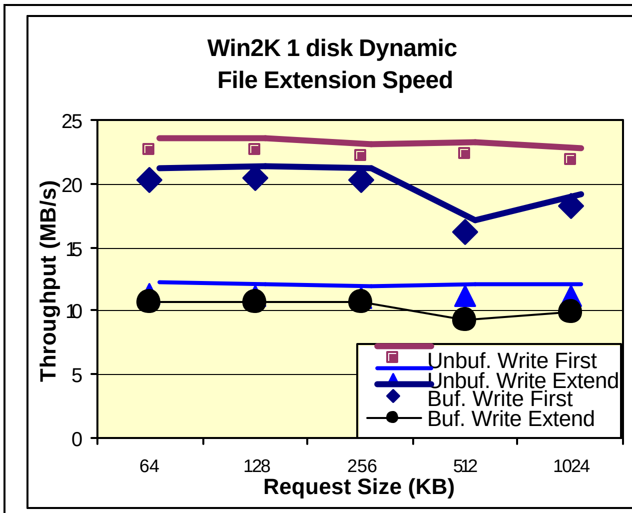


Figure 3 – File extension. The file system must balance the security verses performance. In order to prevent an application from allocating space on disk and then reading its previous contents, the file system serializes write requests. As the graph above shows, this leads to poor write performance on file extension unless one writes only at the end of the file. Applications concerned with good write performance should pre-allocate their files, or only append to them.

5 *NT4SP6 vs. NT4SP3 Sequential IO Performance*

We will first examine the sequential IO performance on the old-old and old-new configurations. We compared NT4 Service Pack 6 to the original NT4 Service Pack 3 used by Riedel. The graphs in Figure 4 tell the story for buffered IO. Things are mostly unchanged except that the performance bug discovered by Riedel was fixed by NT4SP6. In NT4SP3 buffered read requests of 128KB and above had half the sequential IO bandwidth (see left graph in Figure 4). The right graph in Figure 4 indicates this problem is has disappeared. As is expected, WCE writes show much better performance than writes without WCE.

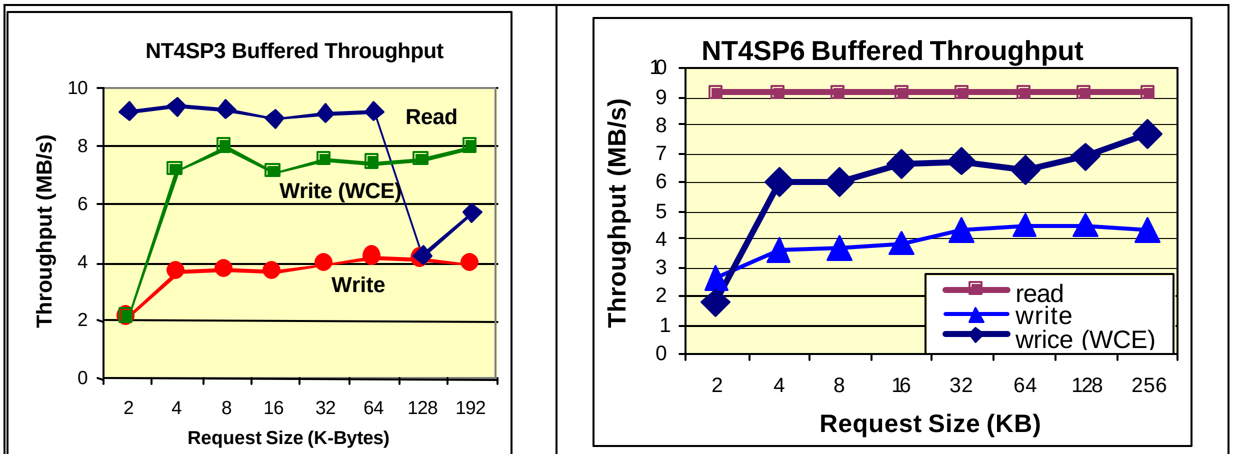


Figure 4 – Synchronous Buffered throughput: NT4SP6 vs. NT4SP3. The dip in the left graph is due to a performance bug in NT that caused decreased throughput at 128KB read requests and above. Buffered reads at small request sizes of 2KB and 4KB run at the full disk speed under NT4SP6.

Figure 5 shows buffered overhead in NT4SP3 and NT4SP6. Riedel's results on the left were taken using GetProcessTimes() while our graph on the right was taken using a soaker. The left graph therefore may be under-reporting processor overhead while the right graph may be over-reporting. Measurements using both GetProcessTimes() and a soaker showed lower overheads for buffered writes at request sizes above 64KB. Riedel's results show both the overhead of buffered reads and writes per MB remains constant for requests above 8KB, with a slight increase for writes for requests above 64KB. Writes in both NT4SP3 and NT4SP6 cost more than reads. In order to compensate for the faster processor in our 333 MHz machine compared to Riedel's 200 MHz machine, we have scaled the NT4SP6 graph so that the relative heights on the two graphs correspond to the same number of instructions executed.

Under NT4SP6, the fixed cost is 20?s per request and marginal cost is 16?s per KB on a 333 MHz Pentium III.

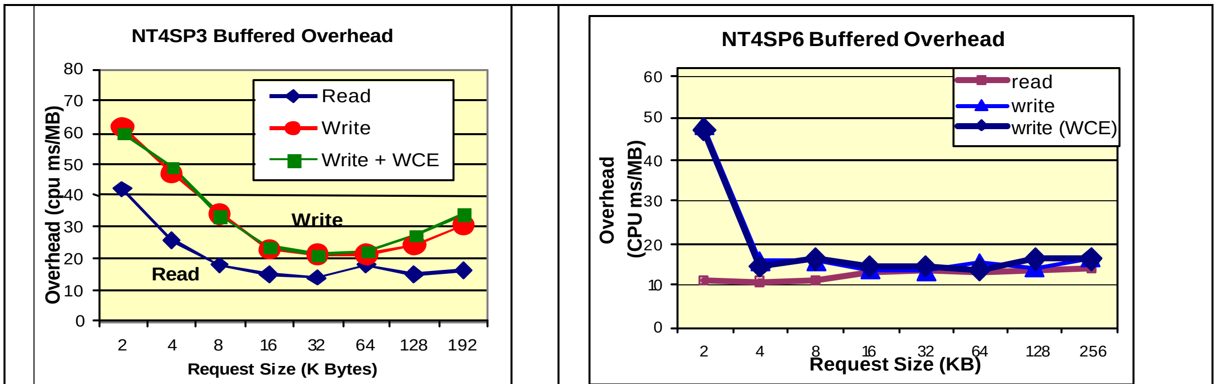


Figure 5 – Synchronous Buffered overhead: NT4SP6 vs. NT4SP3. The right graph shows a lower overhead for large request sizes and read requests of all sizes. The right graph's vertical axis has been scaled to compensate for differences in processor speed between the two machines used for each study. Note that the graph on the left was measured using GetProcessTimes() and may be under-reporting the overhead.

Figure 6 shows the significant improvement in throughput for 2KB unbuffered reads and writes under WCE. Using SP3, the previous study measured a drop of about 4MBps in throughput for 2KB unbuffered write requests with WCE. Under SP6, there is no longer a penalty for 2KB unbuffered writes. With WCE, 2KB writes were able to reach the disk throughput maximum.

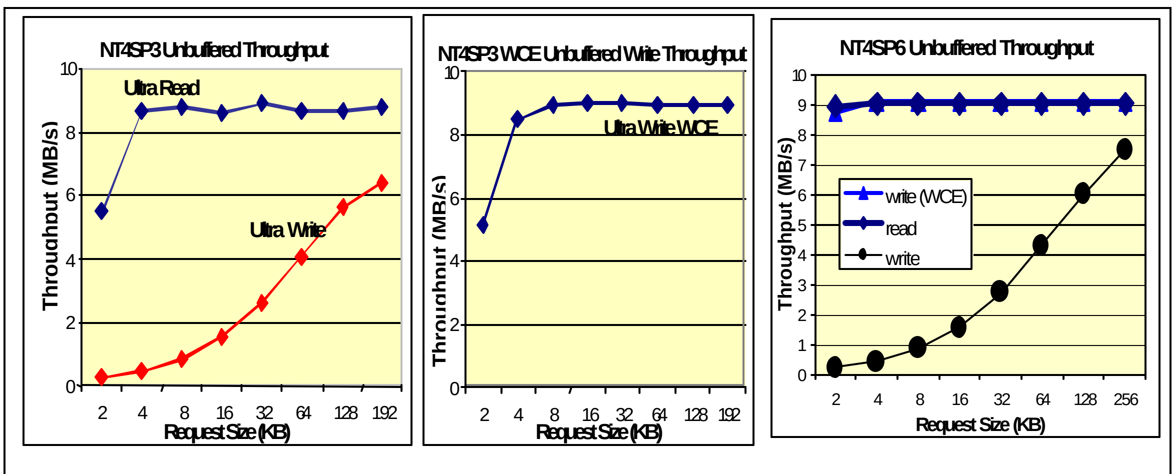


Figure 6 – Synchronous unbuffered throughput: NT4SP6 vs. NT4SP3. The right graph compared to the left and middle graphs of the original SP3 measurements shows SP6's improved read and WCE write throughput. Unbuffered overhead between NT4SP6 and NT4SP3 was the same. Note that due to a plateau in unbuffered writes, the unbuffered write tests on NT4SP6 were run on ST34572 drives rather than the ST34371.

Figure 7 compares the asynchronous read and write throughput of SP3 and SP6. Asynchronous reads and writes under SP6 have the same improvements at small request sizes as synchronous unbuffered requests in Figure 6. The new measurements are on disks that are slightly faster (9MBps vs. 7MBps), so the reader should focus more on the relative shapes of the curves.

Unbuffered read throughput at small request sizes improved dramatically as shown by the upper two graphs. Unbuffered write tests run on a ST34572 drive followed the trend shown by the lower right graph: increasing the write depth up to a depth of four leads to higher disk throughput due to increased parallelism. The bottom right graph showing unbuffered writes on NT4SP6 shows a lower throughput than the bottom left graph of NT4SP3 at 2KB requests. This is likely due to the fact that different drives were under test between the two experiments. As with the previous graphs, the reader should focus on the relative shapes of the curves.

Depths beyond four showed no additional throughput gains. Writes of three deep and beyond reach the same performance as WCE at 32KB requests.

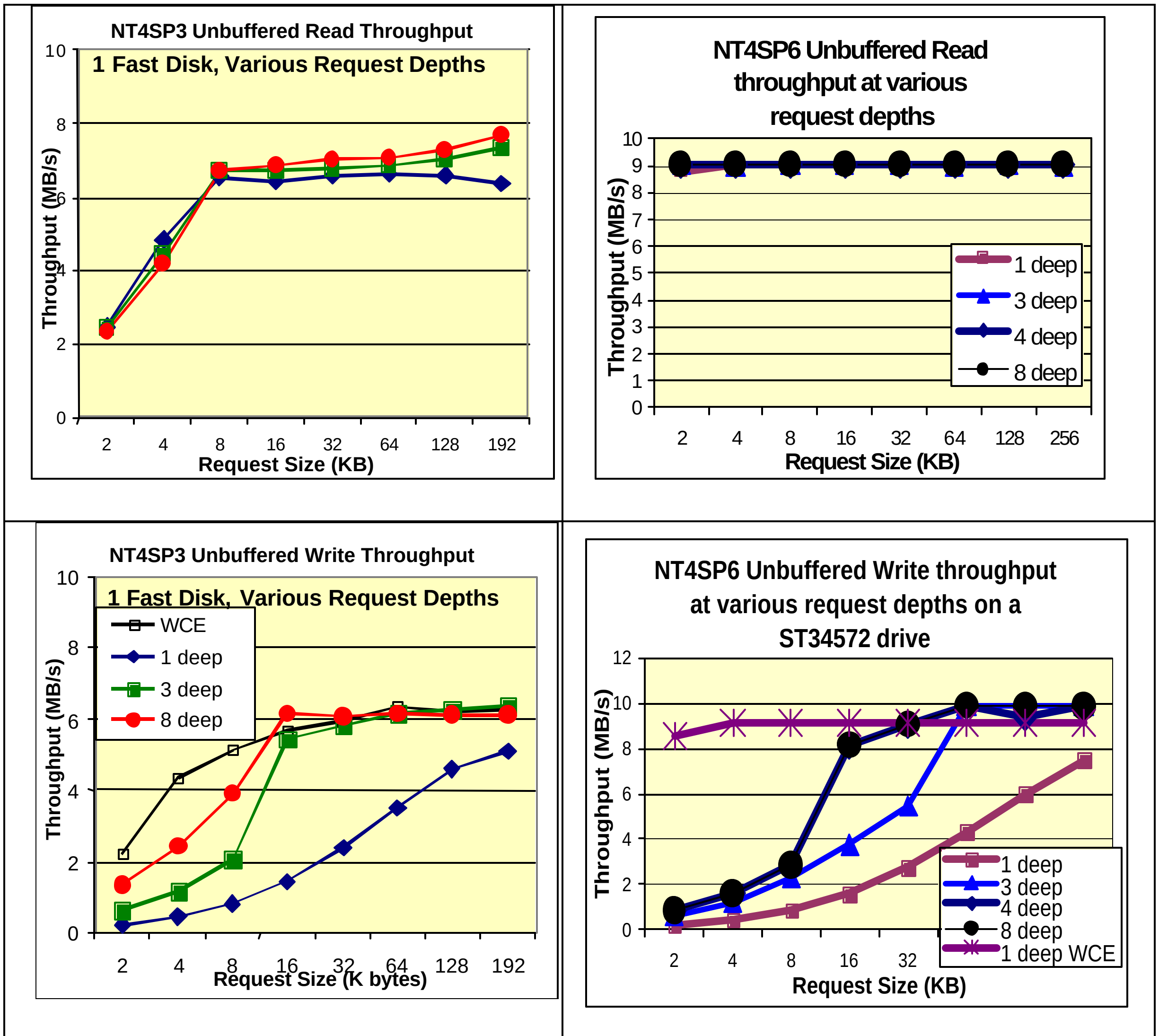


Figure 7 – Asynchronous throughput using no file system buffering: NT4SP6 vs. NT4SP3. Read and WCE write throughput for small requests has improved in SP6 as 2KB and 4KB requests now generate the same throughput as larger request sizes. The right graphs show a higher peak throughput as they were using faster SCSI drives. Unbuffered writes were measured on a ST34572 drive instead of our ST34371. The lower unbuffered write throughput we were able to achieve on NT4SP6 is likely due to differences in the drives under test, rather than due to differences between NT4SP3 and NT4SP6.

6 *Windows 2000 vs. NT4*

Now we contrast Windows 2000 on “old” hardware with NT4 on “old” hardware. As Figure 8 indicates, peak sequential I/O performance under Windows 2000 is virtually identical to that of NT4SP6. Read and write performance, and their associated overheads, are the same as NT4SP6. This holds true with both basic and dynamic volumes. Just as in NT4SP6, the buffered read performance bug for requests above 64KB is gone.

Windows 2000 NTFS unbuffered write throughput is comparable to NT4SP6. Like NT4SP6, increased write depth increases throughput up to four deep writes. Beyond that, the device is saturated, and additional outstanding requests do not increase throughput.

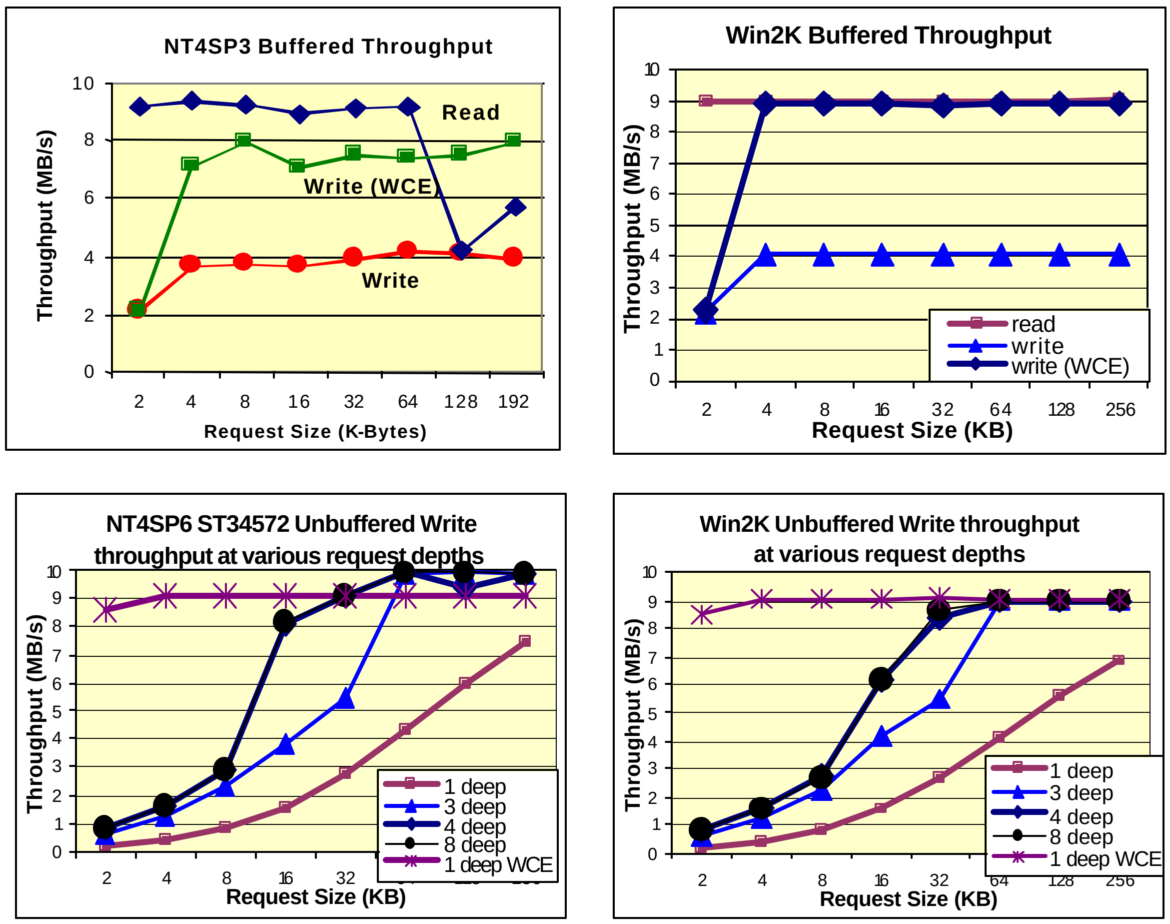


Figure 8 – Windows 2000 Advanced Server buffered and unbuffered throughput on a Basic volume. These graphs are show the sequential I/O performance attained using basic volumes under Win2K. Win2K is shown as having an almost identical performance to NT4SP6. Differences in buffered write performance are due to drive banding. Buffered reads under Win2K don’t show a dip at 128KB and above as the performance bug which caused the dip in NT4SP3 buffered reads doesn’t exist in Win2K.

Two different things are being shown in Figure 9. The first is, unlike NT4SP3, small 2KB and 4KB unbuffered asynchronous read requests in Windows 2000 no longer incur the 33% to 66% performance hit. WCE writes showed similar results. The second is the significant performance degradation seen with requests deeper than one-deep. This is likely an artifact of the drive or driver, rather than Windows 2000, as measurements taken with different drives on different SCSI adapters don't show any such degradation in performance.

The new dynamic volumes, introduced with Win2K, have the same performance curves as basic volumes. Throughputs and overheads remain the same except for an 8% drop in throughput for 2KB unbuffered requests. Figure 10 shows CPU overhead for a dynamic volume is slightly higher than that of a basic volume due to *dmio*'s longer code path lengths. The fixed cost for basic volumes is 50?s per read request and 56?s per write request, while the marginal cost is .46?s per KB read and .51?s per KB written. The fixed cost for dynamic volumes is 58?s per read request and 54?s per write request, while the marginal cost is .43?s per KB read and .50?s per KB written.

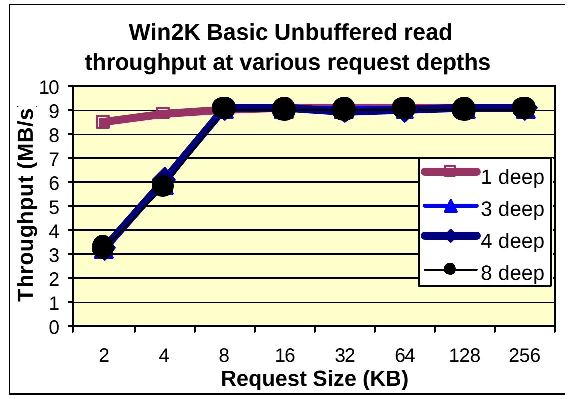


Figure 9 – Windows 2000 unbuffered asynchronous reads. Read request depth beyond a depth of one shows a significant performance hit of 66%. Writes show similar behavior. This is likely due to the drive firmware, and not Win2K. Our Win2K tests using different drives did not show a performance penalty. Unlike NT4SP3, at one deep requests, both reads and writes receive no performance hit.

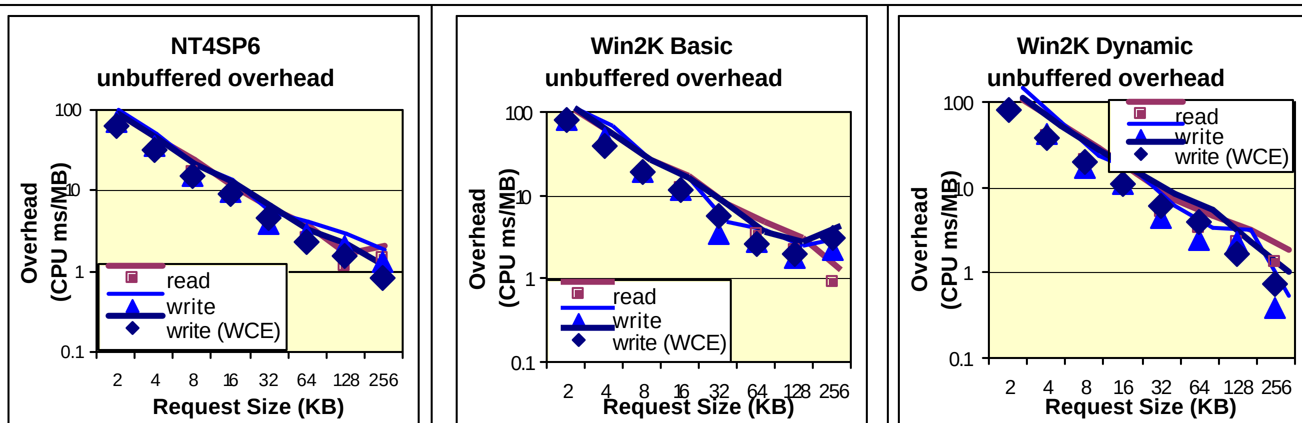


Figure 10 – Synchronous unbuffered overhead comparison of NT4SP3, NT4SP6, and Win2K. Although basic and dynamic volumes under Windows 2000 both attain similar throughput performance, dynamic volumes have a higher processor cost.

7 Windows 2000 SCSI I/O Performance

Having compared NT4SP6 with Win2K on the old hardware, we move on to Win2K on the new hardware (see Table 5): a dual 733MHz Pentium III processor machine with 10K RPM disks.

7.1 Windows 2000 SCSI Random I/O

We first look at random IO performance. Although this paper is primarily focused on sequential I/O, no discussion of the IO subsystem would be complete without some discussion of random I/O.

The Quantum Atlas 10K disks showed good unbuffered random IO performance. Both basic and dynamic volumes had the same results. When we increased the number of disks, the number of IOs per second increased linearly with additional drives.

We used SQLIO to randomly read 8KB blocks from files that each filled an 18GB Quantum SCSI disk. Figure 11 shows that as the request depth increases, the driver and the disk drive are able to optimize the arm movement in order to service the requests more efficiently. One and two-deep requests show the same number of IOs per second. This is because at one deep, the accesses are random. At two-deep, as the first request comes in and the drive begins to service it, the second request also arrives. Since the drive is already servicing the first request, and must commit to the second request before it is given the third request, the drive is unable to optimize the movements of its arms so it achieves only one deep performance. Beyond two-deep requests, the disk is able to optimize by permuting the order requests are serviced, and so the IO rate increases.

With FS buffering, random IO performance, like sequential IO performance, is independent of request depth because the file system is buffering the requests. The file system serializes multiple read requests into one-deep queue of requests for a volume, and the lazy-writer thread also has only one outstanding request. FS buffering improved read performance, but lowered write performance. With FS buffering, the number of IOs per second on reads improved by a factor of two compared to a one deep read. The disk is spinning at 167 revolutions per second, and the disk is performing 220 reads per second (see Figure 11). So, it seems that the reads are being sorted and achieving about 1.3 8KB IOs per revolution. Buffered random writes per second of about 110 IOps is comparable to the 1-deep unbuffered random write throughput (see figure 11). Figure 11 doesn't show the high the fact that each buffered random write is actually a 64KB random read and then an 8KB write. When a buffered write request is received, the cache manager memory maps a 256KB view into the file. It then pages in the 64KB frame continuing the changed 8KB, and modifies that 8KB of data. This means that for each buffered random write includes one or more 64KB reads. The right side of Figure 11 shows this 100% IO penalty.

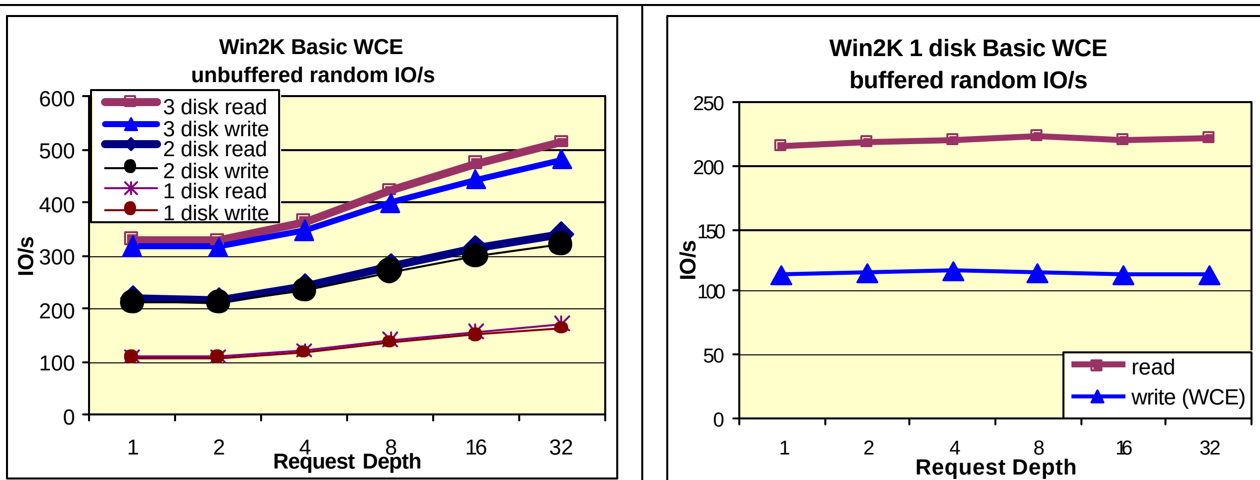


Figure 11 – Win2K single disk random IOs per second. The Quantum Atlas 10K drives showed good unbuffered random IO performance. When FS buffering is used, there is a significant 66% improvement in the read IO/s rate (compared to the one-deep rate), but the write IO/s suffers from the additional read. Win2K sorts requests to optimize disk arm movement. These measurements were all taken with WCE.

Figure 12 shows that processor cost per MB for 8KB buffered random requests is about 20ms for reads and 35ms for writes. For random writes, the additional cost of the additional read for each write is reflected by the write overhead being almost double that of the read overhead.

Unbuffered random requests cost much less than their buffered counterparts. Unbuffered requests averaged slightly over 9ms per MB on both writes and reads. This cost is similar for 2, 3, and 4 disk configurations.

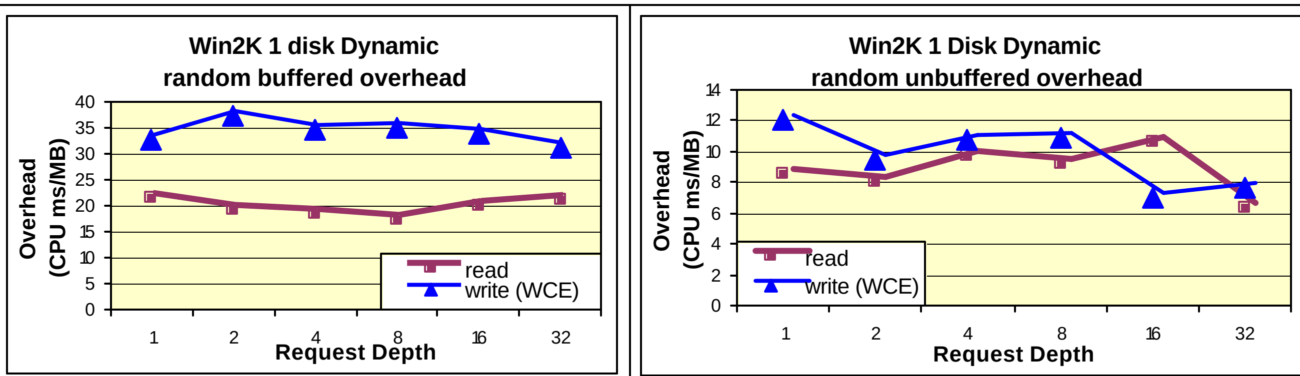


Figure 12 – Buffered and unbuffered random IO overhead. Random IO overhead for unbuffered requests remains low for random unbuffered requests. Buffered random requests are more than twice as expensive than unbuffered requests for reads, and over four times as expensive for writes. The additional read that each buffered random write generates causes the cost of doing random buffered writes.

7.2 Windows 2000 Out-of-the-Box Sequential Buffered SCSI Throughput

Next we measure the performance of Windows 2000 when sequentially reading and writing SCSI disks. Riedel's graph of NT4SP3, the left graph of Figure 13, is of measurements taken on a slower machine with slower disks. Compared with Riedel's machine, the "new" test machine's processors have more than seven times the number of available clock cycles between the two of them and the "new" drives are almost 2.5 times faster. We can learn much however from comparing Riedel's graph to the current measurements since although the vertical scale has changed, the trend lines remain the same. NT4SP3 shows a much greater disparity between WCE writes and reads.

Basic and dynamic volumes are shown in the middle and right graphs of Figure 13. They have almost identical throughputs. Both buffered reads and writes show good performance for all request sizes. Particularly good performance was achieved at small request sizes as FS buffering coalesced the requests into 64KB blocks. This allowed the small requests to run at disk speed. The only exception to this rule was the case of 2KB writes where the throughput was slightly less than 11.5MBps with WCE and slightly less than 7MBps without WCE. This is since, for each 2KB write, Win2K reads a 4KB page off disk and then merges the 2KB write with it. The 4KB read before each 2KB write more than doubles the overhead for each request. Even with this additional overhead, buffered 2KB writes still offered a throughput advantage over unbuffered 2KB writes. Increasing the depth of buffered requests resulted in minimal performance gains.

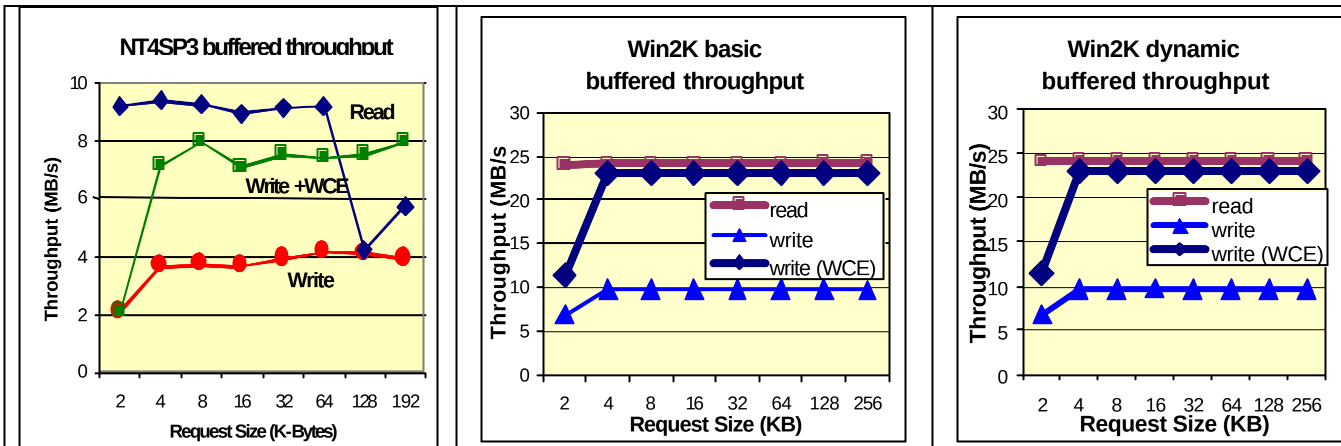


Figure 13 – Single disk buffered throughput. Basic and dynamic volumes show similar performance for buffered reads and writes. Buffered reads improve throughput for small read and write requests. Unlike unbuffered 4KB, and 8KB requests, buffered requests at those same sizes run at disk speed. 2KB write requests are the only exception to this: they require a 4KB read before each 2KB write. Buffered writes without WCE plateau at 10MBps. This is the same write plateau seen for unbuffered writes, however the plateau is earlier. This is because buffered writes smaller than 64KB are coalesced into 64KB blocks before they are written to disk. As we will show later, at 64KB requests and above, writes without WCE see a throughput ceiling of 10MBps due to the disk. The left graph's dip in read performance at requests sizes larger than 64KB is due to a performance bug. Win2K does not contain this bug which is why the Win2K graphs show no such dip in their read throughputs.

Buffered request processor overheads in Figure 14 show similar overhead between basic and dynamic volumes with read and write requests showing similar overheads. Due to read-ahead and lazy writes, much of the actual request overhead was not being charged to our process. Instead, it was only being charged for the buffer copy. GetProcessTimes() therefore can significantly under-report the actual cost per request. 2KB write requests for both *dmio* and *ftdisk* can cost more than twice the processor time of larger requests. This is due to the extra read and merge that must take place for such small requests. Total processor cost equals fixed request cost plus the marginal cost per byte times the number of bytes per request. Since the overhead is the same per megabyte across all requests other than 2KB writes, this implies that the fixed overhead for each request is negligible while the marginal cost per byte dominates. On our machine, the fixed cost for both dynamic and basic volumes was 12?s. The marginal cost for dynamic volumes was 7?s per KB read and written while the marginal cost for basic volumes was 6?s per KB read and 7?s for each KB written.

In Windows 2000, writes still show a significant penalty for small 2KB requests, but the drop in overhead is much more drastic compared to NT4SP3 (see Figure 5). There is no significant penalty for small requests except for 2KB writes which must do a read before each write.

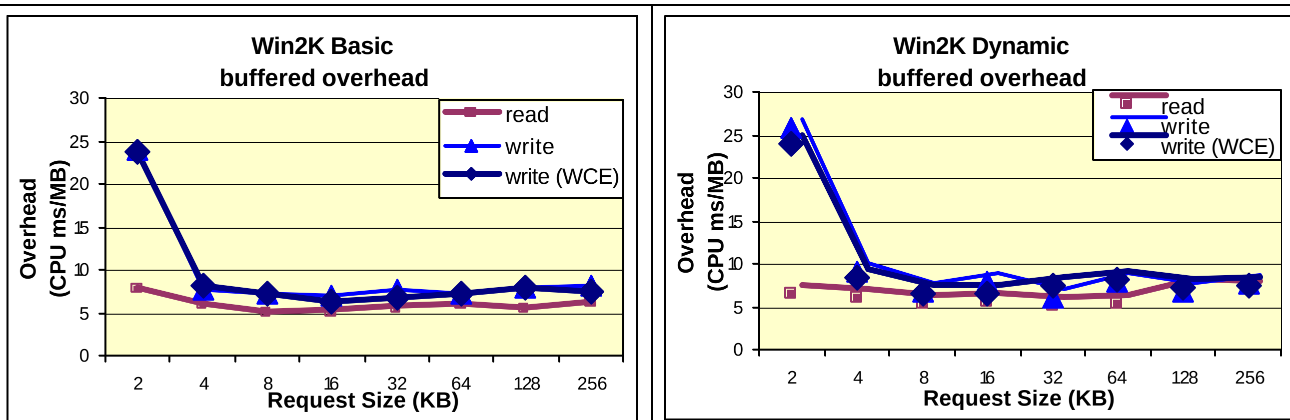


Figure 14 – Single disk buffered overhead. Buffered overheads for both basic and dynamic volumes show little difference. 2KB writes show a significantly higher overhead than reads. This is due to the additional 4KB read and merge that takes place before each 2KB write, effectively more than doubling the request overhead incurred.

The right graph implies that for dynamic volumes, a single processor saturates at 200MBps, and two processors saturate at 400MBps given 5ms of processor time per MB with 64KB read requests. For writes, one and two processors saturate at 125MBps and 250MBps respectively given 8ms per MB written with 64KB requests.

7.3 Windows 2000 Unbuffered Sequential SCSI Throughput

Figure 15 shows the almost identical performance between basic and dynamic volumes with WCE on one-deep unbuffered sequential requests. As synchronous requests are one deep from the standpoint of the disk, one-deep asynchronous and synchronous requests have the same results unless otherwise noted. We will show later in the paper that dynamic volumes have a slightly higher processor overhead than basic volumes. Despite this increased processor overhead, we did not see any effect on unbuffered throughputs on our test machine. This was due to the imbalance between the processor and the disks. On our unloaded machine, the processors are easily able to handle the increased workload of dynamic volumes. Regardless of whether basic or dynamic volumes are used, large requests with WCE combine to provide optimal performance.

A sizable 30% improvement is seen for 2KB reads on two-deep and greater read requests over one-deep requests. At one-deep read requests, the throughput is 17.7MBps, while at two-deep and above the throughput is just slightly under disk speed at 23MBps. This is due to additional pipelining: the amount of time that the disk sits idle is decreased. The computer works to try to ensure that one request is always ready for the disk. There was no improvement for increased write depth. The plateau on writes is an artifact of the disk drive and is explained later.

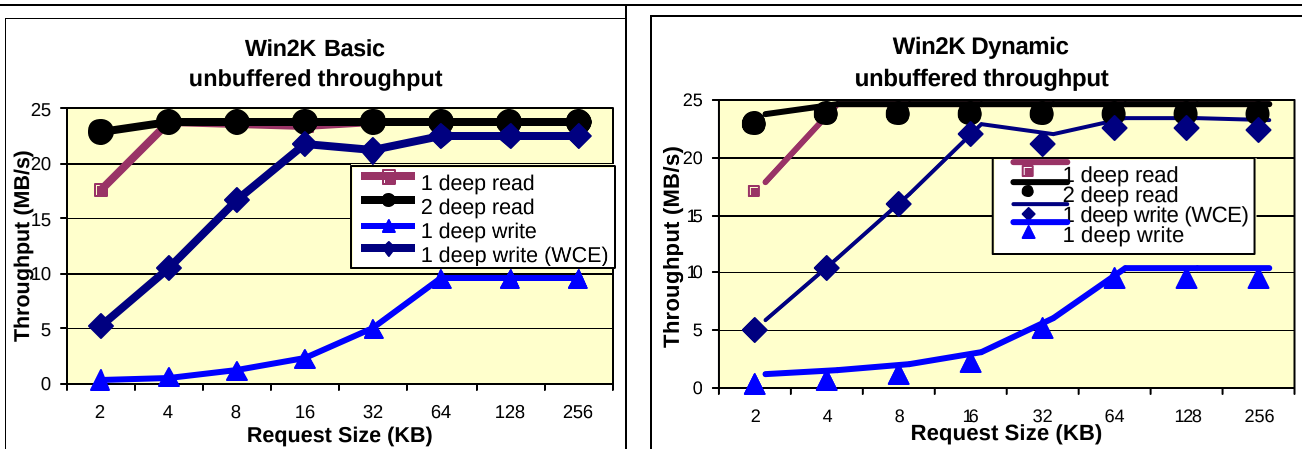


Figure 15 – Single disk unbuffered WCE throughputs for basic and dynamic volumes. Basic and dynamic volumes have similar unbuffered WCE throughput. For clarity, three, four, and eight deep reads have been omitted from these graphs as their results were identical to that of two deep. Similarly, two, three, four, and eight deep writes and WCE writes have also been omitted, as they were identical to the one deep results. Dynamic volumes have up to 25% more overhead than basic volumes at small request sizes. At 2KB requests, basic volumes show a 10% higher throughput when compared to dynamic volumes. Throughput for reads deeper than one-deep show improved throughput due to increased parallelism. Unbuffered writes show no improvement from increased request depth. The write plateau is a property of the disk, as explained later.

Small requests (2KB and 4KB) are expensive: 2KB requests show an overall 40% utilization across both processors in Task Manager. Small reads on dynamic volumes have more CPU consumption than small requests on basic volumes, as the fixed overhead per request is higher. Figure 16 shows as request size is increased, the difference in overhead between basic and dynamic volumes decreases. This implies that dynamic volumes have a higher fixed cost per request than basic volumes, but the incremental costs per byte thereafter are similar. Our measurements support this theory, as the fixed cost for reads on a basic volume was 50?s per request. There was a marginal cost of .48?s per KB. For dynamic volumes, the fixed cost was 58?s per request while the marginal cost was .43?s per KB. Dynamic volumes have a higher overhead per request, but as the fixed request overhead gets amortized over more bytes, the overheads between dynamic and basic volumes become similar. Still, buffered IO has twenty times more processor consumption than unbuffered IO.

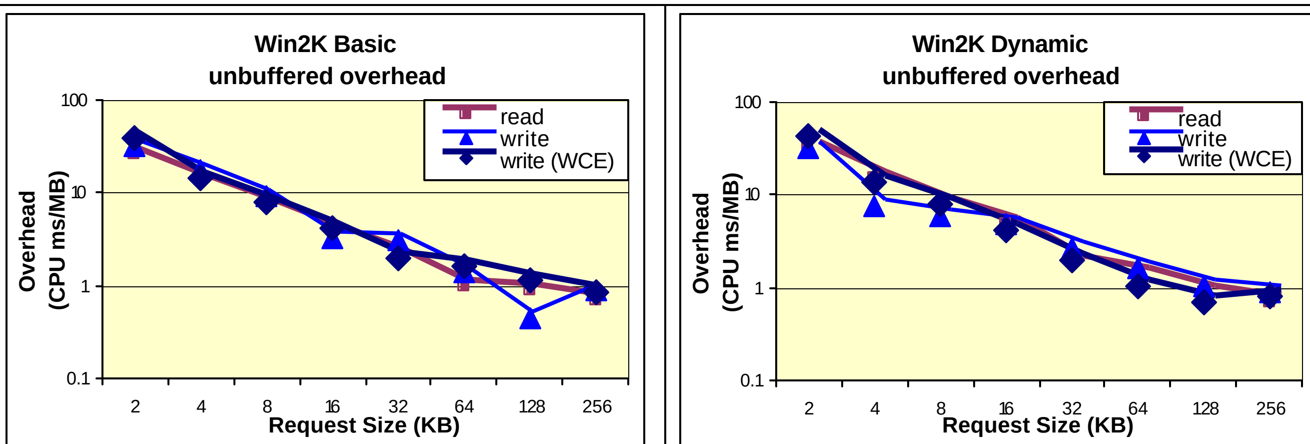


Figure 16 – Windows 2000 unbuffered overhead. The fixed overhead for requests on dynamic volumes is higher than that of basic volumes. At small request sizes, this overhead becomes more predominant causing dynamic volumes to be more expensive. This may be related to the increased throughput of basic volumes over dynamic volumes on small 2KB and 4KB requests.

7.4 *Windows 2000 Multiple SCSI Disk Performance*

In this section, we explore the effects of multiple disks and striping. Striping improves performance through creating a large logical volume out of smaller disks by interleaving fixed size blocks among the actual disks. We first created a two-disk RAID0 stripe set using the Win2K built-in software RAID support. In Windows 2000, the default block size is 64KB. Since blocks are interleaved, sequential requests are optimal for taking advantage of the additional parallelism afforded by multiple disks. In a two-disk configuration, the first request can go to the first drive, followed immediately by a second request going to the second drive. With two drives streaming data off disk at the same time, the potential throughput is doubled. As Figure 17 shows, adding an additional drive doubles throughput for requests larger than 4KB. Small requests don't benefit from additional drives without increased request depth as they don't allow the system to take advantage of the additional parallelism provided. This is due to the disks being interleaved in 64KB chunks. For a small request of 2KB for example, thirty-two requests must be issued before we reach the next chunk on the next drive. This effectively serializes access to the disks since, with 2KB requests, the system is spending most of its time accessing only one disk at a time.

Because Windows 2000 can only create dynamic volume stripe sets, and since basic and dynamic volumes show little difference in performance for simple and striped volumes, dynamic volumes were used for all the measurements below.

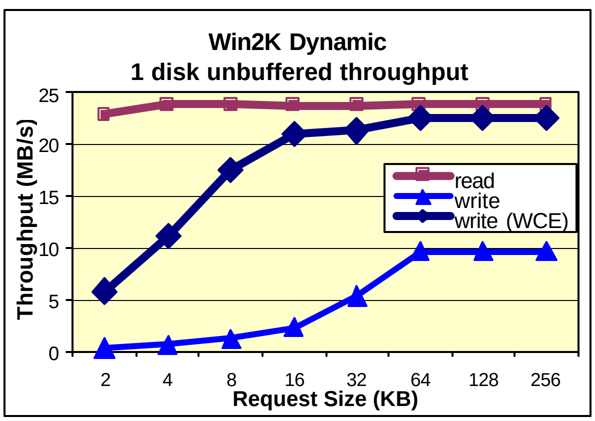


Figure 17 – One and two disk stripped unbuffered throughput. Two disks show a doubling of throughput over that of one disk with three deep requests. The drives are still the main bottleneck. The write plateau is an artifact of the Quantum Atlas 10K disk controller and the controllers of some other drives.

As more drives are added, limitations in other parts of the system become more critical. Each disk is capable of approximately 25MBps. With three disks, there should be 75MBps of bandwidth if disks were the limiting factor. At four disks, there should be 100MBps of bandwidth. However, the host bus adapter and the PCI bus soon become the limiting factor. Figure 18 shows three drives peaking at 73MBps and four drives at 87MBps. The controller could only handle 50MBps on writes which lead to marginal peak throughput gains of 2.5MBps and 2.8MBps for a third and fourth drive respectively. We tested PCI bandwidth by reading directly from disk caches, eliminating media speed as a factor. 87MBps was the maximum PCI bandwidth that we were able to achieve at 64KB requests. Due to this PCI limitation, adding more than three drives on our system gives little to no appreciable additional sequential IO read performance.

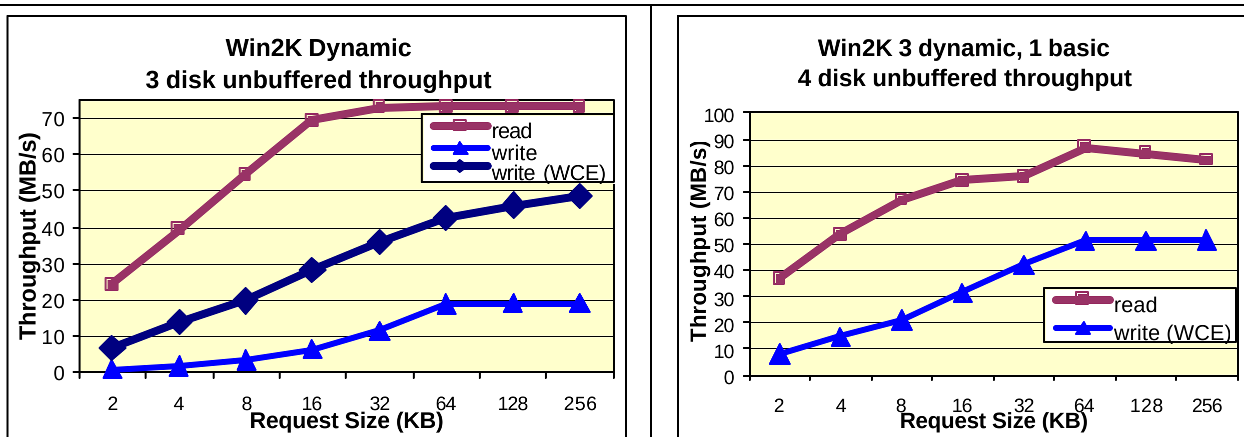


Figure 18 – Three and four disk striped unbuffered throughput. As more disks are added, additional performance gains are no longer a linear 25MBps per disk. A third disk adds an additional 20MBps while a fourth disk adds only 10MBps. The system is now becoming limited by the PCI bus limit of 87MBps and unable to take full advantage of the increased raw disk bandwidth. The 50MBps plateau for writes is likely due to the host bus adapter rather than the PCI bus.

Due to testing constraints, the four-disk configuration was tested on three dynamic volumes and one basic volume using SQLIO for striping. Despite the mixing of disks, since basic and dynamic disks have shown little to no differences in actual throughput, our comparison of striping done across dynamic and basic disks to striping on just dynamic disks is still valid.

Win2K's *dmio* striping shows very little overhead. Compared with processor overhead on a single disk, striping across multiple disks costs close to nothing in processor overhead. Even with three disks running almost at PCI bus saturation, the processor shows no significant additional load. The fixed and marginal costs incurred for three striped disks was virtually identical to that of the one disk configuration.

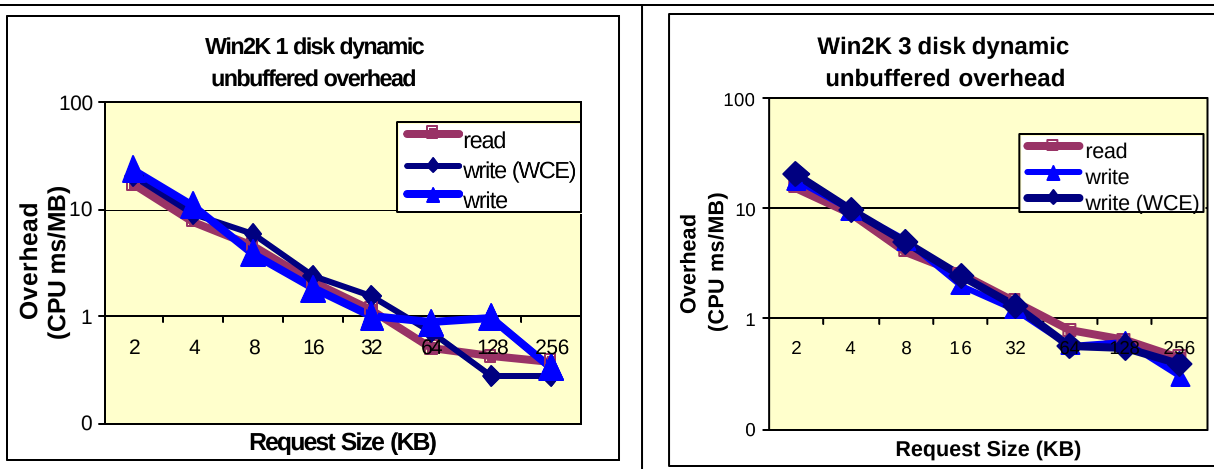


Figure 19 – One disk and three disk striped dynamic volume unbuffered overheads. The graphs above were measured using GetProcessTimes(). They show disk striping on dynamic volumes adds no additional overhead for unbuffered requests.

8 Windows 2000 IDE I/O Performance

In the introduction, we showed how IDE disks are substantially cheaper than SCSI disks per GB of capacity and per disk arm. In these next sections, we look at IDE performance to see what these inexpensive drives are capable of.

8.1 Windows 2000 IDE Random I/O

Unbuffered 8KB requests randomly spread across the entire drive delivered approximately 100 IOs per second using 32-deep requests. At 5400 RPM, this is 1.1 IOs per revolution, the same number we saw from tests on our 10K RPM SCSI drives. As the request depth increases from 1 to 32, the driver, controller, and the disk drive are able to optimize the arm movement in order to service the requests more efficiently. One and two-deep requests show 66 IOs per second for reads and 92 IOs per second for writes – 37% lower on reads and 12% lower on writes than that of a single SCSI Quantum Atlas 10K drive. This is because at one-deep, the accesses are random with an average service time of 15ms for reads. At two-deep, as soon as the first request comes in, the drive begins to service the request. The second request arrives after the drive has been committed to servicing the first request, eliminating the possibility of reordering. The net result is read performance equal to one deep performance. Beyond two-deep read requests, the controller is able to optimize the order in which requests are fulfilled, and so the read IO rate gradually increases. Writes are buffered by WCE, hence 1-deep requests in the file system are potentially very deep in the controller. The controller can sort the random write requests, and so get shorter average seek times.

Random IO performance is intimately linked to drive latency. Each time the heads seek to a new track, they must wait for the right sectors to travel under the drive head. As we increase the number of RPMs, the drive latency decreases so the drive waits for a shorter period of time before the desired sectors travel under the drive head. Considering that a 5400RPM drive rotates 46% slower than a 10K RPM drive, it is easy to see the effects of rotation rate. Similarly, the Quantum Atlas 10K had an advertised seek time of 5.0ms while the Quantum Fireball's advertised seek time is 9.5ms which is 48% slower. The Atlas 10K SCSI drive is about two and a half times more expensive than a Fireball IDE drive, so IDE turns out to be cheaper in terms of IOs per dollar. At $209, the Quantum Fireball costs $3.17 per KAPS (kilobyte access per second) while the Atlas 10K costs $534 or $5.09 per Kaps. The number of IOs per second scales linearly with the number of drives, so two and a half drives would give us 2.5*(66) IOs per second or 165 IOs per second. Compared with the 105 IOs per second achieved on an Atlas 10K, for the same amount of money, you get 57% more IOs per second than a single SCSI drive.

Drive	**Random 8KB IOs per second**	**Dollars per KAPS**
Quantum Atlas 10K (10,000 RPM)	105 read 105 write	$5.09
Quantum Fireball lct08 (5,400 RPM)	66 read 90 write	$3.17

Table 9 – SCSI and IDE random IO performance.

Indeed, when we tested a Maxtor 40GB DiamondMax™ drive that spins at 7200 RPM, we saw 107 random reads per second. This $250 drive comes closer to the random read performance of the 10K RPM SCSI drives. However, its write performance suffered with only 58 random writes per second. This is likely due to poor controller performance in the Maxtor hampering WCE write performance. In other tests we found that the Maxtor had poor performance on its outer 5GB as it was missing revolutions due to a slow controller. Performance improved on the inner bands as the media speed decreased and the controller was able to keep up.

With file system buffering, random IO performance, like sequential IO performance, is independent of request depth. For buffered reads and writes the file system serializes requests into one-deep requests. For buffered reads, the cache manager is apparently scheduling requests. With FS buffering, the number of IOs per second on reads doubled compared to a one deep unbuffered read. This is comparable to the gap on a single SCSI disk. The disk is spinning at 90 revolutions per second, and the disk is performing an average of 120 writes per second (see Figure 19). So, it seems that driver, controller, and drive are sorting the writes and achieving about 1.3 8KB IOs per revolution.

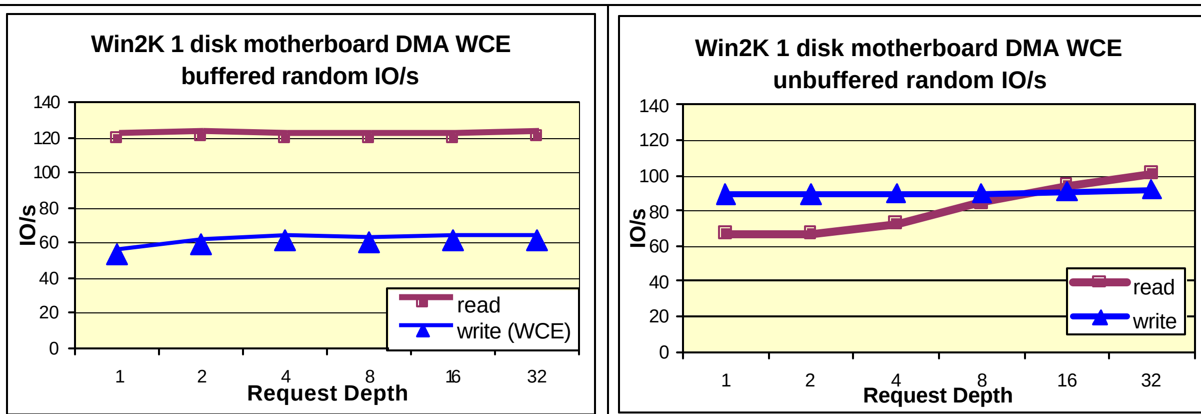


Figure 20 – 8KB random IOs per second vs. request depth. Random IO performance is dependent upon the disk RPM. Compared with a 10K RPM Atlas 10K SCSI drive, the 5400 RPM Fireball IDE drive has 46% fewer RPM, a decrease similar to the decrease in IOs we measured across the two drives. The Atlas 10K however, costs more than two-and-a-half times the cost of a single IDE drive. In terms of IOs per second on a single disk, SCSI is better, but for IOs per dollar, multiple IDE drives have 57% more IOs than a single SCSI drive for the same amount of money.

Buffered random IO for IDE is similar to that of SCSI. Reads cost 22ms per MB on average, while writes cost 35ms per MB. The higher overhead for writes is again due to the cost of the extra read associated with each write request.

Unbuffered random requests cost half to a two-thirds less than their buffered counterparts. Unbuffered requests averaged slightly over 10ms per MB on both writes and reads. This cost is similar for 2, 3, and 4 disk configurations.

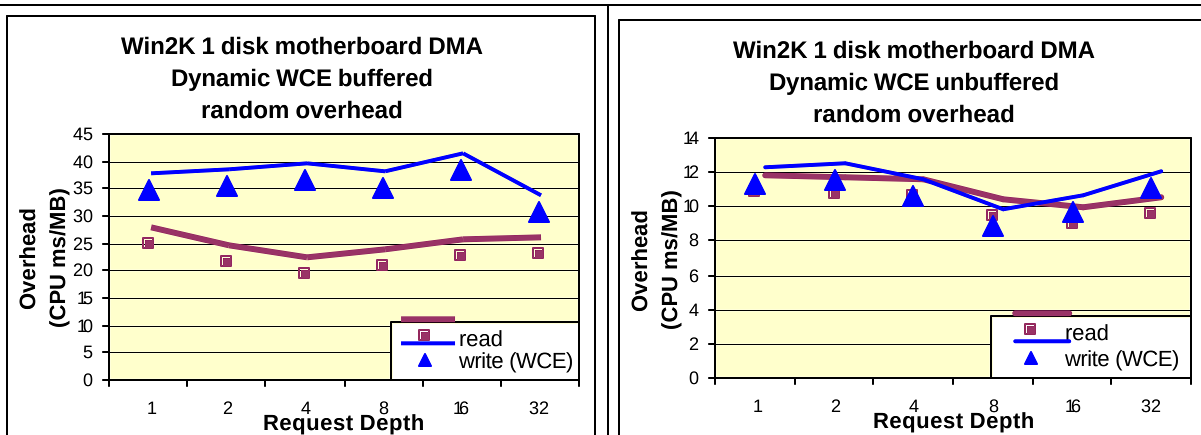


Figure 21 – 8KB random buffered and unbuffered overhead. Buffered random writes are expensive – the overhead per MB was 270ms compared to 20ms per MB buffered reads. Reads and writes show similar unbuffered overheads at 12ms per MB.

Multiple IDE drives provided near linear gains with each additional drive. With four disks at a request depth of one per disk, we were able to achieve over 262 IO/s for reads and 358 IO/s on writes. We measured the peak number of IOs per second possible in the system by sending random requests to each of the disks.

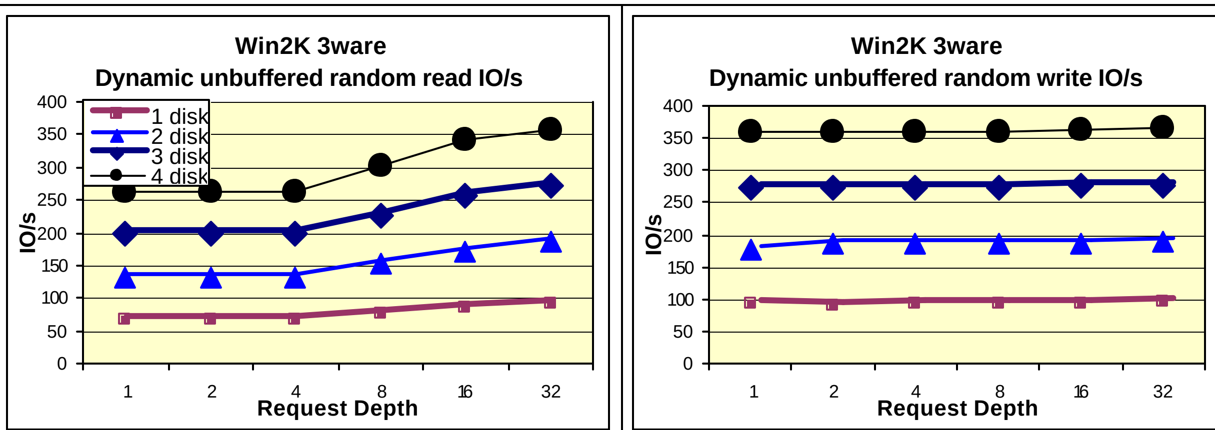


Figure 22 – Windows 2000 Multiple IDE disk IOs per second. As with SCSI, the number of IOs per second scales linearly with the number of IDE drives are added to the system. The request depths shown above are the requests outstanding per drive.

Figure 23 shows unbuffered random KAPS for two disks along with a two disk RAID0 stripe set. Unlike Figure 22, the request depths shown are the total number of IOs outstanding instead of the number per drive. Thus, a four-deep request depth in Figure 23 is equal to two-deep per drive for two disks, or one-deep per drive for four disks. The point to note in Figure 23 is that adding additional disks to a stripe set does not result in linear gains. This is because, unlike the case in Figure 22, random requests aren't being sent to each disk individually but rather to a single logical volume. Thus, parallelism isn't guaranteed. When sending two random requests to a logical volume, there is a 50% chance that both requests will go to the same drive rather than separate drives.

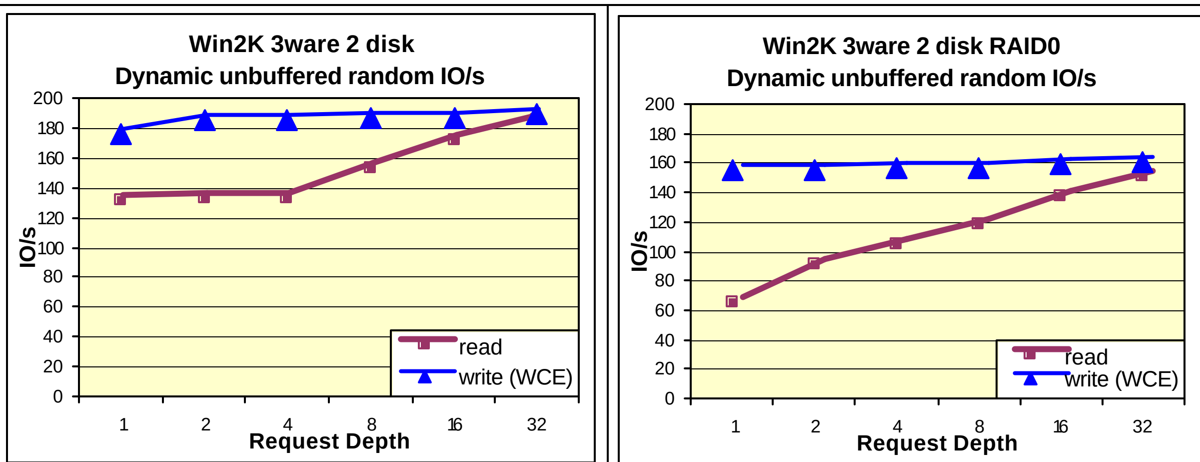
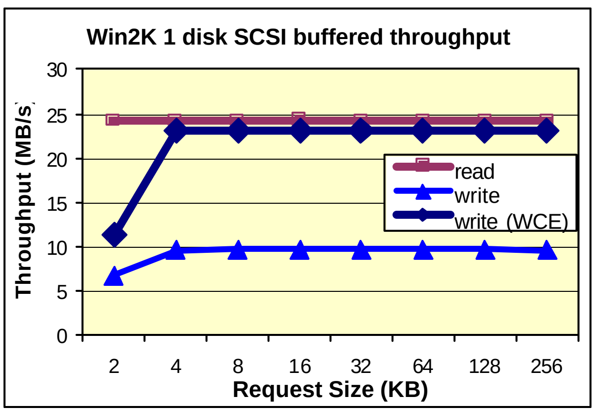


Figure 23 – Windows 2000 Multiple IDE disk random IOs per second: RAID0 vs. peak IOs per second. The graph on the left shows the peak number of IOs per second available in a two disk system. We measured peak number of IOs per second by issuing requests to each of the disks individually, ensuring that the workload was split evenly between the two disks. The request depth shown on the left graph is per disk. The right graph shows the IOs per second achieved with the two disks in a RAID0 stripe set. The request depth shown is total number of outstanding requests, not per disk. RAID0 shows lower IOs per second than the peak IO rate. This is due to the fact that the workload is not evenly split evenly between the two disks. For two deep requests, there is a 50% chance that both IOs will go to the same drive, leaving the other drive idle. When both requests go to a single disk, we lose the parallelism of having that second disk which reduces our overall number of IOs per second.

8.2 *Windows 2000 Out-of-the-box Sequential Buffered IDE Throughput*

Figure 24 shows buffered sequential throughput on both a Quantum Atlas 10K SCSI drive on an Adaptec Ultra160 SCSI controller and a Quantum Fireball 5400 RPM IDE drive connected to a 3ware controller. Despite having half the rotational rate of the Atlas, the Fireball is able to achieve about 80% of the Atlas's throughput — more than 19MBps. This is due to the Fireball having a higher areal density than the Atlas. Thus, the lower rotational rate of the Fireball is offset by a greater number of bits per inch.

There are no surprises here; the buffered throughput trend lines of the IDE drive are similar to that of SCSI drives. Both buffered reads and writes showed good performance for all request sizes and are able to achieve disk speed. Particularly good performance was achieved at small request sizes as FS buffering coalesced the requests into 64KB blocks. This allowed the small requests to run at disk speed. The only exception to this rule was the case of 2KB writes where the throughput was slightly less than 5MBps with WCE. This was because for each 2KB write, Win2K reads a 4KB page off disk and then merges the 2KB write with it. The 4KB read before each 2KB write more than doubles the overhead for each request. Even with this additional overhead, buffered 2KB writes still offered a throughput advantage over unbuffered 2KB writes. Increasing the depth of buffered requests resulted in minimal performance gains.



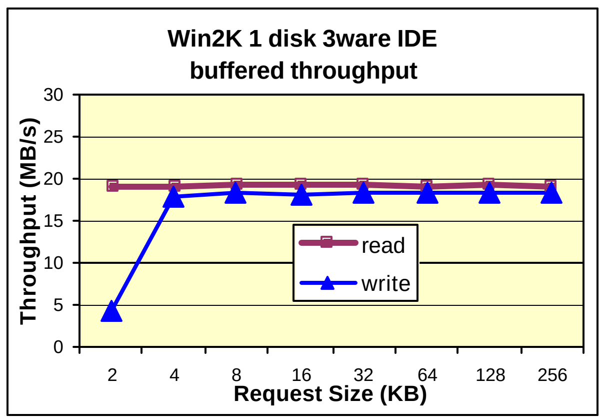


Figure 24 – Windows 2000 SCSI and IDE buffered sequential IO performance. Despite having half the rotation rate, the IDE drive compares favorably to its more expensive SCSI cousin. 2KB writes show the characteristic performance penalty. The IDE drive shows slightly more than half the throughput of its SCSI counterpart for 2KB writes.

As shown in Figure 25, buffered request processor overheads are similar for both SCSI and IDE. This is expected since requests for both the SCSI and IDE drives were offloaded to hardware controllers: an Adaptec Ultra160/m SCSI controller and a 3ware 3W-5400 controller respectively. Differences between the SCSI and IDE overheads are due to differences in their respective drivers.

Due to read-ahead and lazy writes, only part of the total request overhead is being charged to our process. Instead, the process is just being charged for the buffer copy. Therefore, measurements taken with GetProcessTimes() under-report the actual number of processor cycles used. A soaker process was needed to include cycles used by the cache manager into account. That process measures the "free" time, and so can measure the entire system overhead being consumed on behalf of SQLIO. Figure 25 shows the results measured by using the soaker.

In Windows 2000, writes still show a significant penalty for small 2KB requests (because they first read the 4KB page from disk), but the drop in overhead is much more drastic. There is no significant penalty for small requests except for 2KB writes. For small 2KB and 4KB requests, buffered requests show a lower overhead than unbuffered requests due to request coalescing. The 8KB request size is the breakeven point; at request sizes greater than 8KB, unbuffered requests have a lower overhead than buffered requests. The resulting equation is approximately:

$$Cpu\_cost = 12\mu s + RequestSize * .8ns + (Request\_Size/64KB)*45\mu s.$$

The first term is the cost of calling the buffer manager, the second term is the cost of the memory copy, the third term is the amortized cost of the lazy writer.

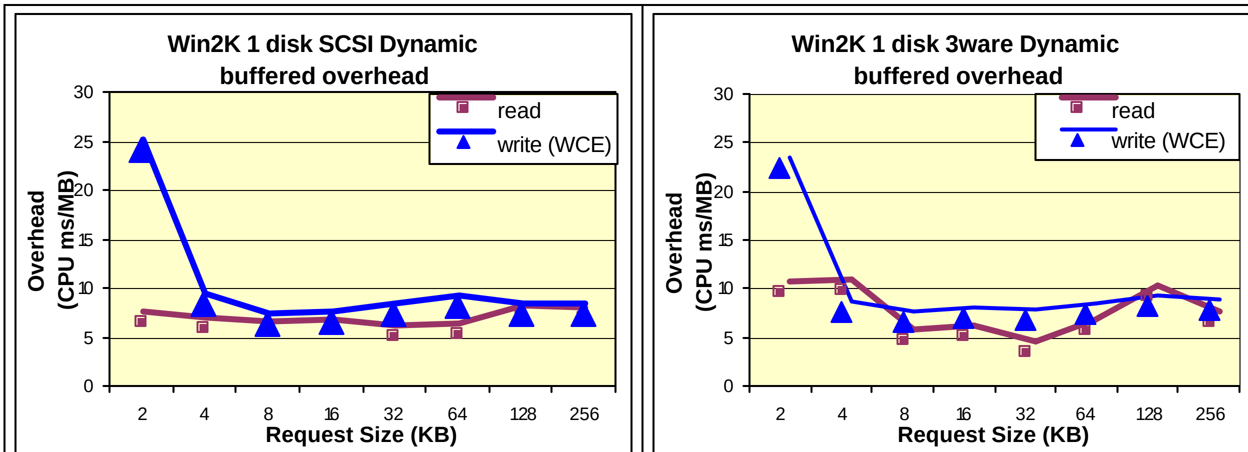


Figure 25 – Windows 2000 buffered sequential overhead. Both SCSI and IDE have similar buffered overheads. This is expected since both offload much of the IO processing on to a separate controller card. 2KB writes show a significantly higher overhead than reads. This is due to the additional 4KB read and merge that takes place before each 2KB write, effectively more than doubling the request overhead incurred. Due to request coalescing, requests smaller than 8KB actually have less overhead with FS buffering enabled. For requests larger than 8KB, unbuffered requests have a lower overhead.

8.3 Windows 2000 Sequential Unbuffered IDE Throughput

Figure 26 shows single disk unbuffered throughput across both a SCSI and an IDE disk. Like buffered throughput, unbuffered throughput on the Fireball IDE drive shows good sequential performance. At one deep requests, maximum disk throughput is attained at read requests of 16KB or more and write requests of 64KB or more. Optimal throughput is reached through two-deep reads and one-deep writes (except on the 3ware card which required two-deep writes) which both attain disk speed at 8KB requests. Increased depths showed no additional improvement in throughput.

Unlike the unbuffered SCSI throughput, IDE read throughput suffers at small 2KB request sizes. This is an artifact of the 3ware card and not of the Fireball drives. As Figure 26 shows, when the Fireball IDE drive is connected to the motherboard's IDE channel using DMA, small requests show the same general trend lines as SCSI; 2KB read requests no longer have the dip seen with the 3ware card. The 3ware wasn't handling as many IOs per second as the motherboard IDE channel.

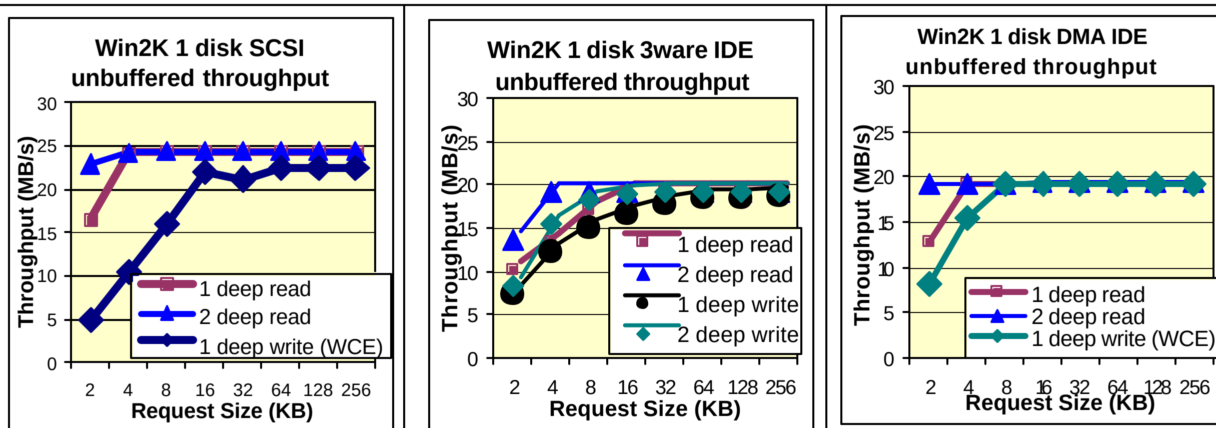


Figure 26 – Win2K SCSI and IDE unbuffered throughput. Like its buffered throughput, the Fireball's unbuffered throughput was very good. The trend lines of the Fireball IDE drive, connected to the motherboard IDE controller with DMA, were similar to that of the Atlas 10K SCSI drives with two-deep as the optimal depth of reads, providing disk speed at all request sizes. A depth of one was optimal for writes, providing disk speed at 8KB and larger requests. When the Fireball was attached to the 3ware card, small 2KB reads suffered. The 3ware card wasn't handling as many IOs per second as the motherboard IDE channel.

Figure 27 shows both SCSI and IDE unbuffered overheads. Only overhead for an IDE drive connected to the 3ware card is shown, as unbuffered overhead for an IDE drive connected to the motherboard controller had almost identical results. Unbuffered overhead for the 3ware card was slightly higher than that of SCSI per request. Each IDE request had a fixed cost of 57?s. The marginal cost was .42?s per KB written and .53?s per KB read. This compares to a fixed cost of 32?s and a marginal cost of .25?s per KB for SCSI.

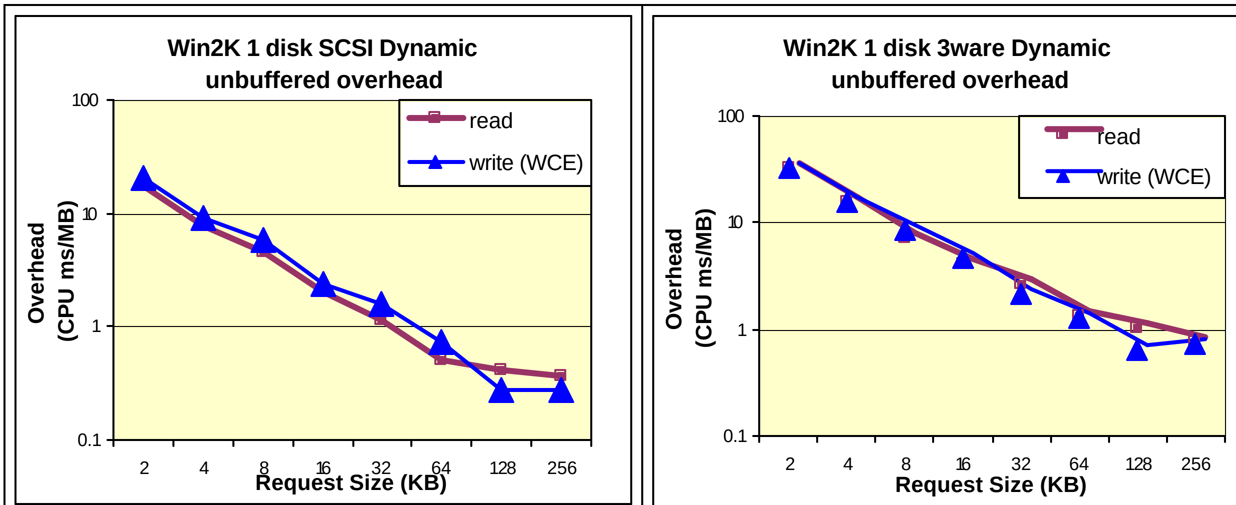
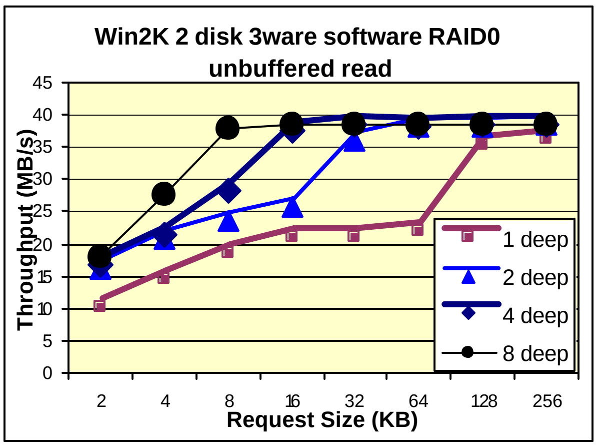


Figure 27 – Win2K SCSI and IDE unbuffered overheads. As shown above, overhead between SCSI and IDE is very similar. Overhead for the 3ware card was slightly higher than that of the Adaptec SCSI controller.

8.4 Windows 2000 Multiple IDE Disk Performance

The previous section showed that good sequential performance is easy to obtain with individual IDE drives. Figure 28 shows unbuffered sequential throughput of two IDE drives connected to a 3ware controller. The drives were bound together into a logical volume using Win2K's software striping.

For sequential workloads, Figure 28 shows that two drives double read throughput and nearly double write throughput. Reads peaked out at 39MBps while writes peaked at 35MBps. As Figure 28 indicates, request depths of four or eight are required to achieve the throughput offered by the second drive. Increasing the number of requests outstanding enables additional parallelism in the system. With only one request outstanding, only one drive is active at a time, effectively serializing requests. The exception to this rule is one-deep requests larger than 64KB in size. NTFS breaks such large requests into 64KB chunks that are then distributed among the drives. Consequently, 128KB and 256KB requests show the greater throughput of deeper requests as Win2K is dividing the large requests into multiple 64KB requests and issuing them concurrently.



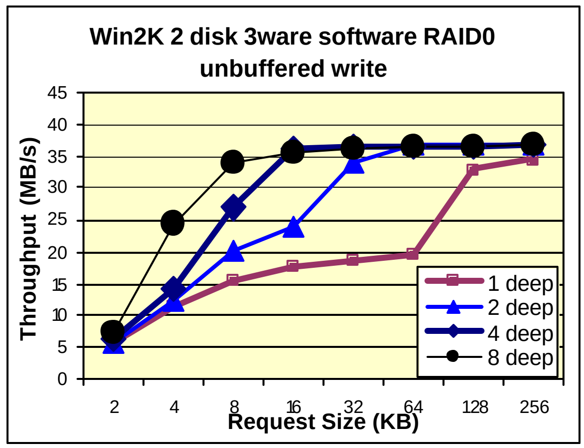


Figure 28 – Win2K 2 disk IDE software RAID0 throughput. Performance at two disks scales nearly linearly with the addition of a disk with reads peaking at 39MBps and writes at 36.5MBps. In order to take advantage of the parallelism offered by the second drive, deeper requests are required.

One of the features provided by the 3ware card is hardware striping. As Figures 28 and 29 show, the sequential throughput attained through 3ware's hardware striping is almost identical to that of Win2K's built-in software striping. The only difference between the two was at one-deep, 64KB requests. For 64KB requests, Win2K had a throughput of 20MBps for both reads and writes, while the 3ware showed a throughput of 30MBps. This implies at one-deep 64KB requests, Win2K is still only accessing one disk at a time.

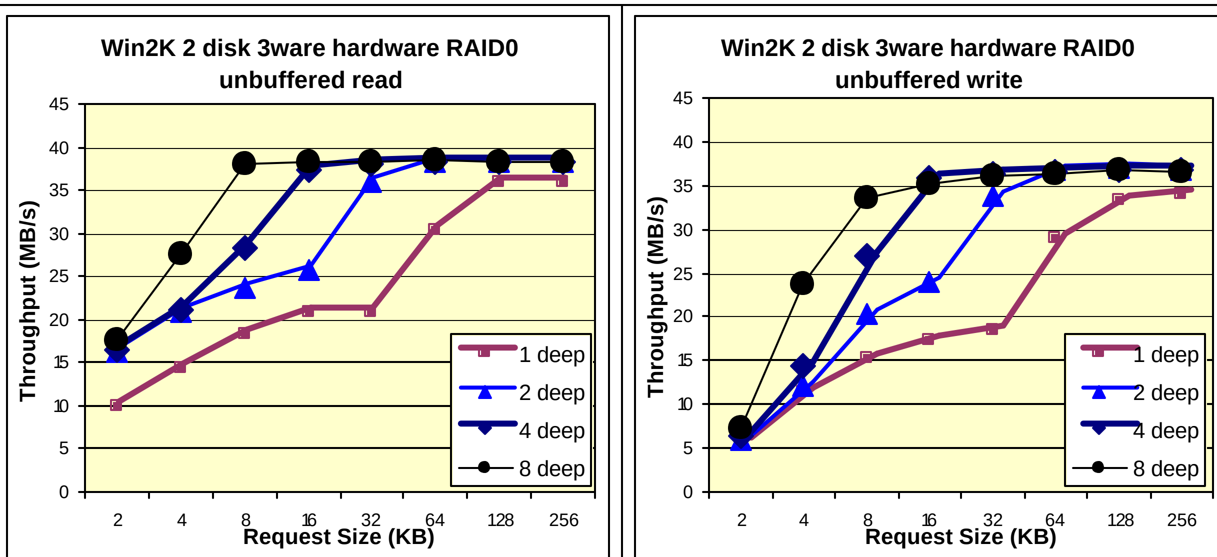


Figure 29 – Win2K 2 disk IDE hardware RAID0 throughput. 3ware's hardware striping shows similar performance to Win2K's software striping. Requests smaller than 64KB show decreased throughput since the stripe chunk size is 64KB. Multiple requests must be issued to a single disk before moving on to the next disk in the stripe.

Figure 30 shows the unbuffered overhead for both hardware and software RAID0. Both hardware and software RAID0 have similar processor cost per MB. RAID0 had a fixed cost of 57?s, and a marginal cost of .46?s per KB read and .52?s per KB written.

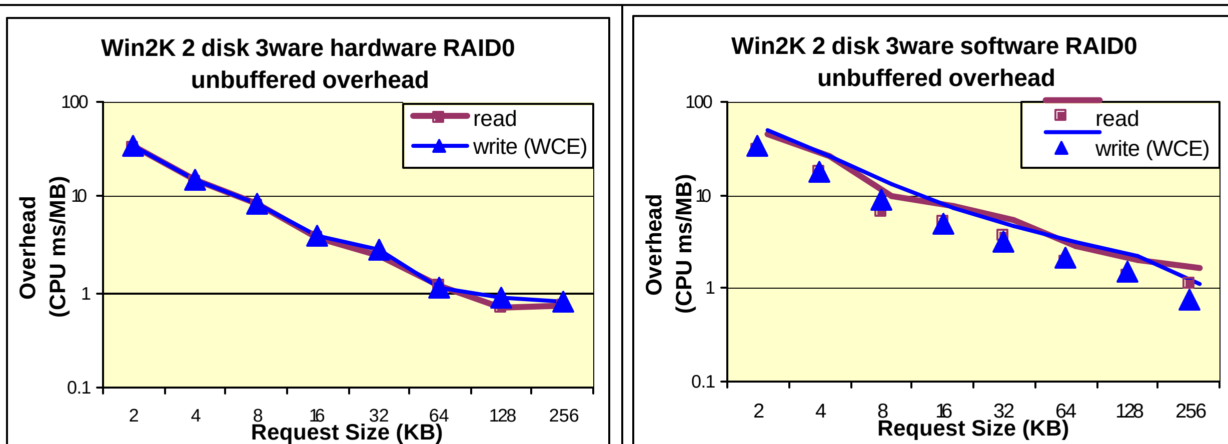


Figure 30 – Hardware and Software RAID0 unbuffered overhead. Windows 2000's software RAID0 and 3ware's hardware RAID0 show almost identical results.

When file system buffering is used, Windows coalesces smaller requests into 64KB chunks and issues one deep requests to the disk. Since software RAID0 peaks at 20MBps for one-deep, 64KB requests while hardware RAID0 peaks at 30MBps, the peak buffered throughputs matched correspondingly. Riedel found that host-based RAID controller cards provided performance generally poorer than software RAID. However, given the 3ware card's hardware RAID0 has slightly less overhead, almost identical unbuffered performance, and much better buffered throughput, it is advantageous to use the hardware RAID0 provided on the controller.

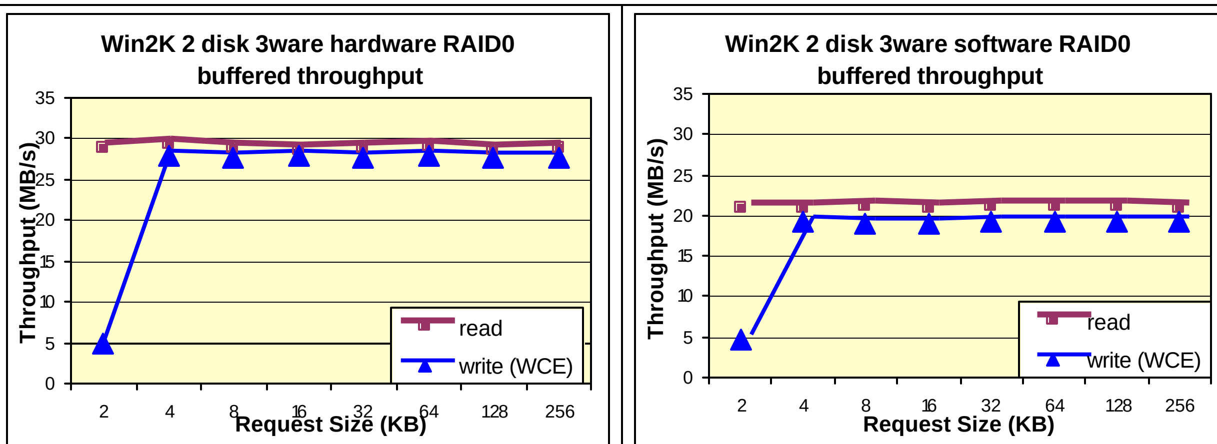


Figure 31 – Hardware and software RAID0 buffered throughput. In the unbuffered throughput section, we described how the 3ware's hardware RAID0 shows better one deep, 64KB request throughput than Win2K's software RAID0. Since FS buffering coalesces multiple requests into 64KB chunks that are issued one deep to the drive, this is reflected in the 3ware's better buffered throughput.

Figure 32 shows the unbuffered sequential throughput of three and four Fireball IDE drives bound together using 3ware's hardware striping. Like the two drive cases explained earlier, request depth and request size are parameters that play a large role in the throughput attained. The larger the two parameters become, the greater the throughput.

At three disks, we begin to reach the PCI bandwidth limitation of the 3ware card we tested. While reads have still scaled fairly linearly with the number of drives, unbuffered writes have plateau at 40MBps – an improvement of only 5MBps over the two drive configuration. For reads, three disks peak at 55MBps.

At four disks, the 3ware card becomes the limiting factor for sequential accesses. Writes show a 1MBps improvement over three disks, while reads show a negligible improvement. The additional fourth disk adds little to overall sequential throughput in our tests on the outer edge of the disk. However, performance at the inner diameter can still be improved substantially by the fourth disk. This can be important in video streaming applications where an important metric is the minimum streaming bandwidth across the entire disk. Although the first generation 3ware card we tested was limited by its PCI interface, second generation cards no longer have this limitation and additional drives should result in almost linear gains – up to 98MBps for reads and up to 70 MBps for writes.

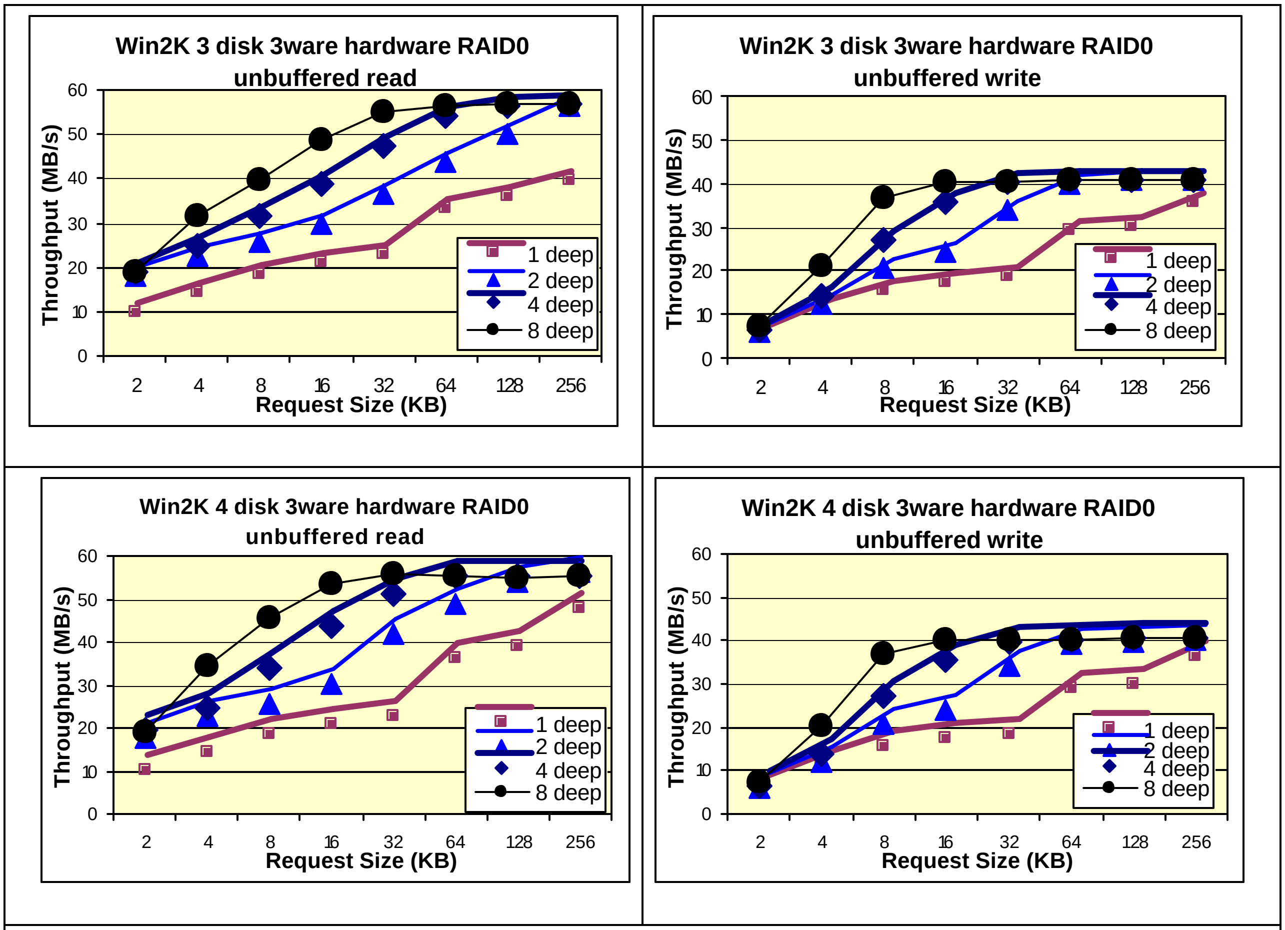


Figure 32 – Win2K three and four disk hardware RAID0 unbuffered sequential throughput. With three drives, reads scale fairly linearly with each additional drive, however writes plateau at 40MBps. When an additional fourth drive is added, reads show a negligible increase in performance. The 3W-5400's PCI interface is unable to keep up with the increased drive bandwidth on the outer edge of the four disks.

One of the advertised features of the 3ware card is a form of mirroring technology called "TwinStor" that offers RAID1+0 read performance using two drives. [Horst] Rather than reading just one drive, TwinStor spreads reads across both drives to provide RAID0 performance. As the odd-looking left graph shows, TwinStor is very much dependent upon request size and depth. It is not until there was a total of 256KB of sequential requests outstanding did TwinStor give us any benefit. TwinStor achieves its best sequential performance with deep read requests and large request sizes. On writes, slightly less than single disk performance was achieved.

The TwinStor algorithm intentionally produces this behavior. The firmware examines the request history to determine if it is mostly sequential or mostly random. Requests smaller than 256KB are treated as random to allow the second disk to independently service a second request.

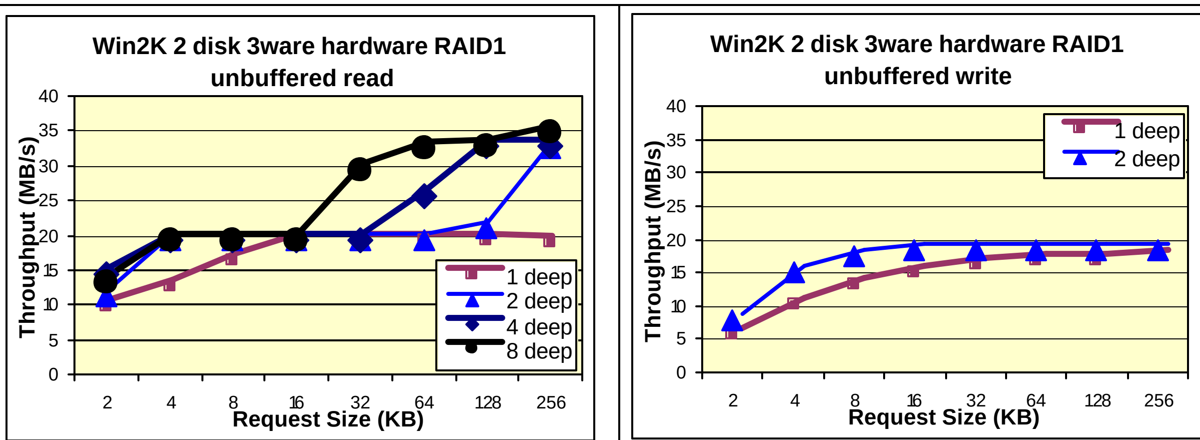
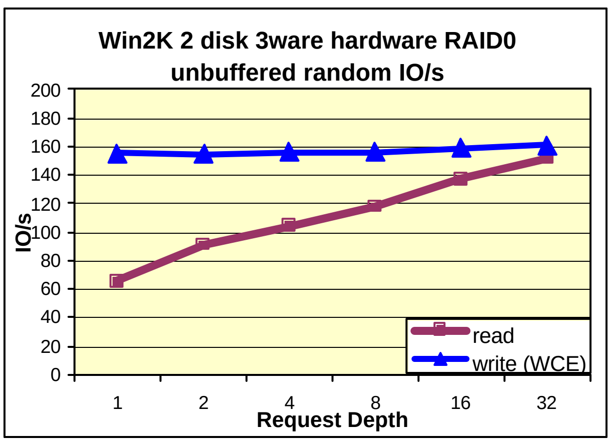


Figure 33 – Win2K two-disk hardware RAID1 sequential throughput. One of the advertised features of the 3ware card is a technology called "TwinStor" which offers RAID1+0 read performance using two drives. However, in order to achieve good rest performance, we had to make deep requests at large request sizes. It is not until there was a total of 256KB of requests outstanding did TwinStor give us any benefit. On writes, slightly less than single disk performance was achieved.

The 3ware card shows good random read IO performance with its hardware RAID1. Figure 34 shows both RAID0 and RAID1 unbuffered random performance with 8KB requests. As the graph on the right indicates, the additional request scheduling being done by the 3ware controller has improved random read IO performance significantly over that of a RAID0 stripe set. At two deep, the 3ware card shows almost linear gains from the second disk. The additional performance is due to both drives containing the same data. In the case of a RAID0 stripe set, there is a 50% chance that both random IOs will be for only one of the disks with two requests outstanding. This means for the second request, half of the time one of the disks will be idle while the other is busy servicing both outstanding requests. This can be seen in the left graph as the single disk performance of 66 IO/s only reaches 99 IO/s (66 IO/s + .5 * 66 IO/s) with two requests outstanding. The RAID1 TwinStor drive however can efficiently dispatch the second IO 100% of the time to the second disk, making full use of the second set of arms. This can be seen on the right graph as two-deep requests result in 119 IO/s or close to two times the 66 IO/s performance of a single disk.

With deep queue depths, total random read throughput is higher with TwinStor than with RAID0 for a different reason: the controller reorders the requests in a way that minimizes disk arm movement on both drives. This produces a more than doubling of random IO performance on two drives compared to one.

Write performance with TwinStor was similar to that of a single disk. This is due to the cost of mirroring: writing two copies of the bits uses both drives at the same time. The additional parallelism offered by the second drive is lost. RAID0 is not subject to this constraint as it lacks data redundancy, so it is able to take advantage of the second drive.



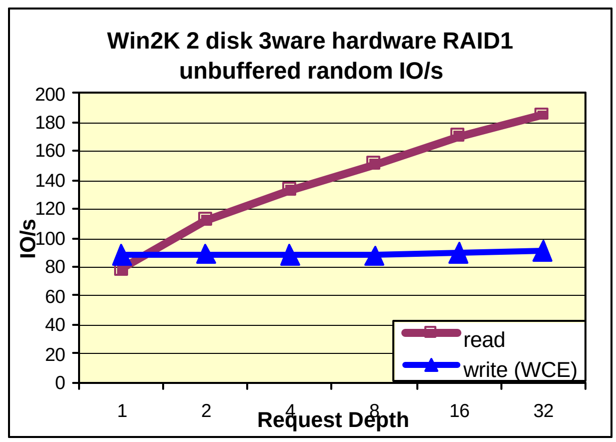


Figure 34 - Win2K two disk hardware RAID1 random IOs per second. By scheduling read requests to both drives, a two disk TwinStor RAID1 array shows better performance than a two-disk RAID0 array. However, write performance is only that of a single disk as both disks are utilized during writes in order to mirror the bits, so the parallelism offered by the second drive is lost. The RAID0 array is not subject to this constraint, so it is able to take advantage of the additional disk.

As shown in Figure 35 below, the overhead measured for hardware RAID1 is similar to the overhead for a single disk. This is as expected as hardware RAID1 offloads the additional RAID1 processing onto the 3ware card requiring additional processor cycles. The fixed cost for a single disk was 57?s, while the marginal cost was .53?s per KB read and .42?s per KB written. A RAID1 set had a fixed cost of 57?s, and a marginal cost of .55?s per KB written and .60?s per KB read.

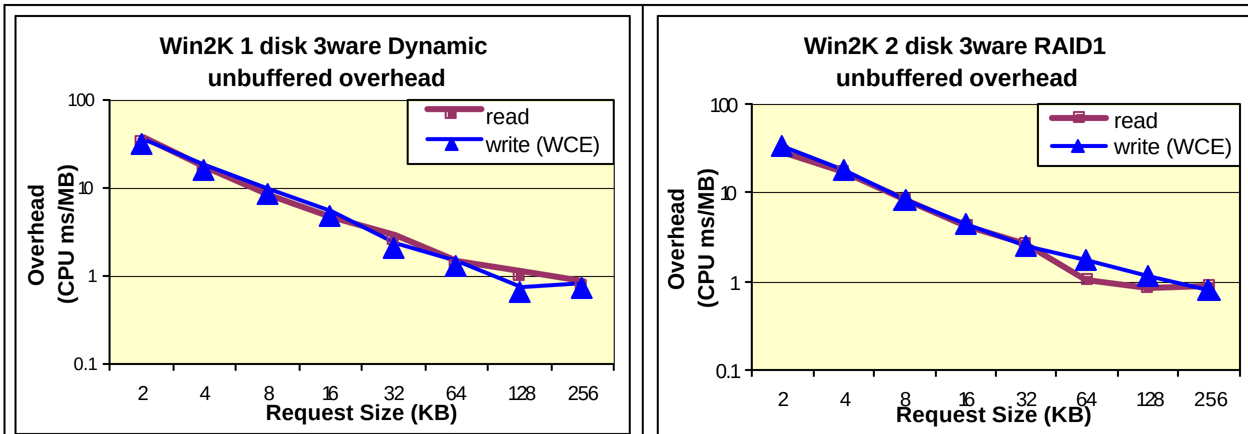


Figure 35 – Win2K hardware RAID1 overhead. Hardware RAID1 has similar overhead to that of a single disk. This is as expected as hardware RAID1 offloads all processing onto the 3ware card rather than burdening the processor.

To repeat, as suggested in Table 6, the next generation of 3Ware host-bus adapters and drivers improve on many of these results.

9 Network IO Performance

So far, we have been examining local IO performance using Windows 2000. As networking and broadband have become more ubiquitous, file-sharing performance is becoming more important, even within the home. In this section, we briefly examine network IO performance using CIFS, the Common Internet File System. CIFS was chosen because it is the network file system most commonly used with Windows NT and Windows 2000 installations. This section is a brief overview rather than a comprehensive study of the intricate world of network IO performance. In our configuration, we measured a point-to-point connection with very low latency. Networks with greater latency or multiple streams may see different results.

We connected the new-new Dell Precision 420 to a comparable Dell Precision 410 machine with two 750MHz PIII processors and 256MB SDRAM. In order to ensure that the disks on the Precision 410 were not the bottleneck, we used three Atlas 10K drives and used RAID0 to create a stripe set. Local throughput on the Precision 410 was greater than that of the highest throughput we were able to achieve through our simple Gigabit Ethernet LAN. We used two SysKonnect SK-9821 copper Gigabit Ethernet cards and a single crossover cable to connect the two computers together. Both computers were running Windows 2000 Server with the addition of the alpha versions of the SysKonnect drivers for the SK-9821. We got a 7% increase in throughput with the following registry key settings. All tests reported here use these settings:
The keys are all relative to

```
Hkey_local_machine\System\CurrentControlSet\Services\Tcpip\Parameters
        MaximumFrameSize = 9014       /* use jumbo frames
        NumberOfReceiveBuffers = 100  /* use memory if you need it
        NumberofTransmitBuffers = 50  /* ditto
        MaxIRQperSec = 2500           /* interrupt coalescing
                                      /* and moderation
```

The tests were run using the same methodology that we applied to local IO. The UNC and mapped drive tests were run on the new-new machine. The network tests were run on the new-new machine with it as the client and the Precision 410 as the server. We wanted to see what out-of-box performance we could attain.

Figure 36 shows both read and write throughput to a local Quantum Atlas 10K disk using both Universal Naming Convention (UNC) path (such as \\lchung\e$) and a network mapped drive (the network share \\lchung\e$ masquerades as drive F:). Compared with accessing the local drive directly, accessing the disk through UNC paths and mapped drives gives substantially lower performance – especially if requests are larger than 16KB.

Mapped drives show better read performance than UNC paths at request sizes of 64KB and above. Where our mapped drive ran at near disk-speed with large requests, UNC reads dropped to the one and two MB per second range. As the request size increased, the throughput decreased. Interestingly, mapped drives showed good performance at large request sizes, however additional depth caused throughput to decrease. Overall, a depth of two seemed to be the optimum value for both UNC and mapped drives, reading and writing. For good performance, drives should be mapped rather than accessed using their UNC paths.

Unlike reads, writes showed disappointing results for both UNC and mapped drives. Throughput increased with request size until disk speed was achieved with 16KB requests, then suddenly dropped to 2.1MBps with 32KB requests, and finally plateaued at 5.5MBps to 6MBps – showing conclusively that size does matter!

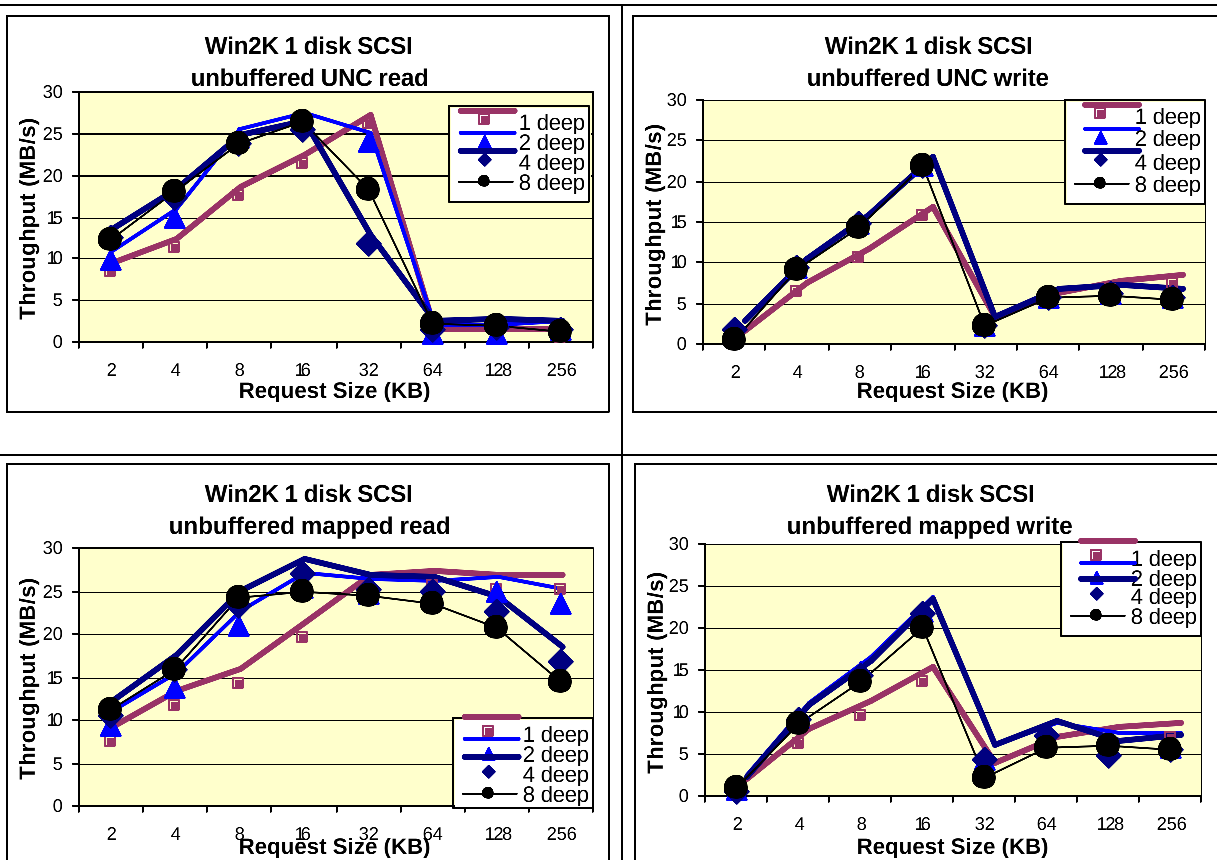


Figure 36 – A comparison between UNC paths and mapped drives accessing a local Quantum Atlas 10K drive. Accessing files using both UNC and mapped drives adds additional system layers and overhead to each request. Compared with single disk results, this results in degraded throughput. Although write performance is the same for both UNC and mapped drives, read performance differs substantially. For good performance, mapped drives should be used.

Accessing local files through UNC and mapped drive paths forces the IO to go through the network stack despite the fact that the IO is local. This combined with some sub-optimal buffering and serialization choices result in poor performance. Since programs will increasingly be using network-transparent names from the Active Directory, this will be an increasingly important issue.

The primary cost of UNC and mapped drives is processor time. Compared with single disk overhead, requests to UNC and mapped drives can cost ten times more processor time per MB. On our machine, 2KB unbuffered mapped reads consumed over 71% of both processors. On small request sizes, UNC showed significantly less overhead for small 2KB writes than on mapped drives. Mapped drives had a fixed overhead of 563µs per write request while UNC paths had a fixed overhead of 454µs per write. 2KB reads showed a smaller impact. As the request size increases, both UNC and mapped drives converge to 27ms per MB.

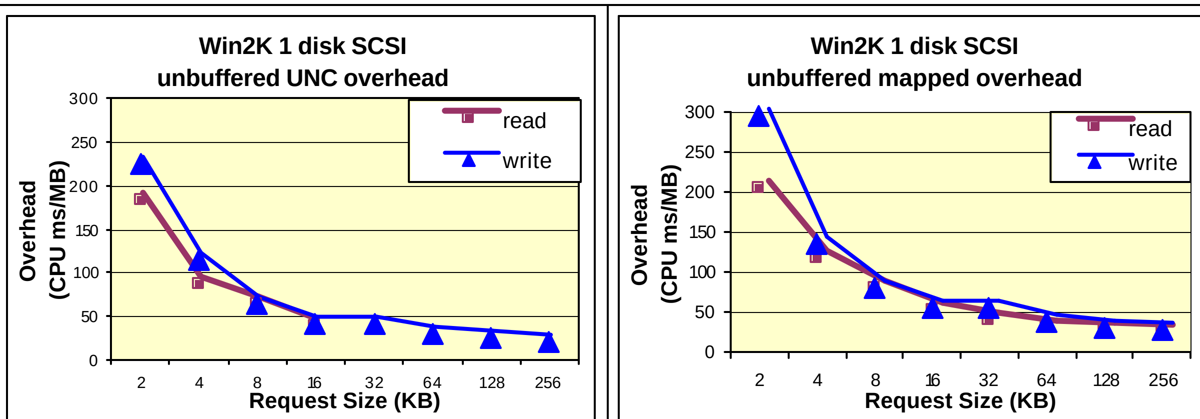
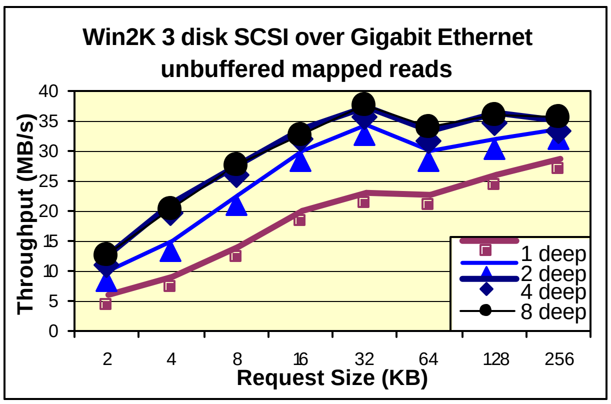


Figure 37 – Unbuffered UNC and mapped overhead. UNC and mapped IO are expensive, with up to ten times as much overhead per MB compared with local drives. UNC showed lower overhead than mapped drives for small requests, however this difference in overheads diminishes with increased request size. This implies a higher fixed overhead for mapped drives. We've truncated the left graph's read overhead at the 16KB request size for reads. This is due to the large performance degradation that's seen with UNC reads with large request sizes. As the number of IOs per second dropped to single digits, the error bound on our soaker became larger than the results, so we were unable to measure that overhead.

CIFS read throughput over Gigabit Ethernet was very good. We were able to achieve over 37.5MBps using sequential 32KB read requests. At 1GBps, the PAP of Gigabit is 125MBps. In round numbers, the half-power point is 63MBps. Although we were unable to achieve the half-power point, this is likely due to the PCI bus as our machines did not reach processor or disk saturation. At 37.5MBps, the server must read from the SCSI controller at 37.5MBps and then write to the Gigabit Ethernet adapter at 37.5MBps. The total bandwidth being used by both reads and writes is 37.5 * 2 or 75MBps, which is nearly the maximum throughput we were able to achieve using local disks.

Unbuffered write throughput was less than half the read throughput. Using 16KB requests, we were only able to achieve 15MBps. With requests larger than 16KB, a performance hit similar to the local UNC case shown in Figure 34 is seen. This indicates that the poor write performance may be due in part to the drive mapping translation. In addition, unlike reads requests that stream data from the server with little acknowledgement from the client, writes require more handshaking as we observed much higher bi-directional network traffic with writes.

Processor utilization for reads is less per MB than that for writes. Whichever system was having data pushed to it had the higher overhead. For reads from the server, processor utilization on the server was 28% compared to 32.2% on the client for 2KB four-deep read requests. For 128KB requests, the server processor utilization was 14% while the client processor utilization was 23.4%. For writes to the server, the server processor utilization was 11.6% for 2KB writes and 7% for 128KB writes, while the client's processor consumption was 6% and 9.3% respectively.



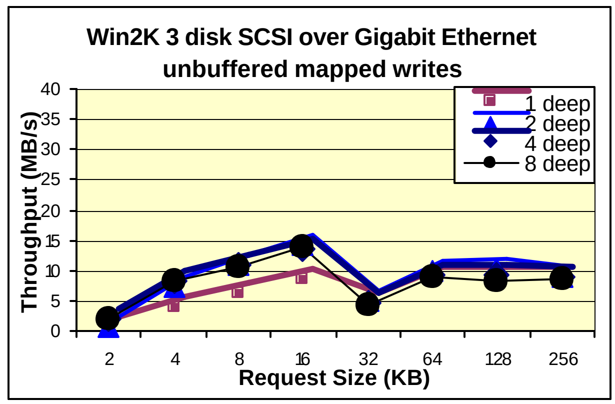


Figure 38 – Unbuffered mapped throughput over Gigabit Ethernet. CIFS read performance is very good: at 37.5MBps it reaches PCI bus saturation on the server. It is likely that with a faster PCI bus, the throughput would be greater. Unbuffered writes showed poorer performance, with peak write performance less than half peak read performance. The curves of the write throughput graph are similar to those of mapped writes in Figure 35.

10 Summary

To summarize:

- Win2K and both SP6 and SP3 of NT4 show similar one-disk sequential I/O performance.
- NT4SP6 has no performance penalty for small-unbuffered requests. NT4SP3 shows significant penalties for both unbuffered synchronous reads and unbuffered asynchronous requests.
- Basic and dynamic volumes in Win2K produce similar results.
- For sequential workloads, 32bit 33MHz PCI is now the major bottleneck. PCI bus saturation can be achieved with only three disks, a configuration not uncommon on workstation machines.
- Random workloads limited by the PCI bus. A PCI bus can handle about 10,000 random IOs per second, approximately the capacity of 100 disks.

Single disk IDE performance is very good, especially relative to their much more expensive SCSI cousins. Good performance is easy to attain; the rules for optimizing performance on SCSI drives still hold true for IDE. Although the 3ware card we tested was hampered by its PCI bus interface and unable to take full advantage of all four disks, second generation 3ware cards show good PCI performance and no longer have this limitation.

- Single disk IDE drives are the clear winner on bang for the buck. For sequential loads, they come very near the performance of SCSI drives that are more than three times as expensive. On random IO, IDE drives have a 25% better IOs per second per dollar ratio than SCSI drives.
- 3ware's IDE adapter card offers a linear improvement in sequential throughput for reads and good scalability for writes up to three IDE drives. Filling the last slot with a fourth drive gives little to no improvement in sequential IO performance on our outer band measurements due to limitations in our 3ware card's PCI bus interface. Second generation 3ware cards have no such limitation. Random IO performance is still good, as the throughputs involved are much lower so four drives don't come anywhere near saturating the card or bus.
- IDE drives show similar overhead to that of SCSI drives, unless PIO is used. Avoid PIO as it severely degrades performance while dramatically increasing processor overhead. DMA is needed for good performance.

Table 10: Summary of the performance and price-performance of SCSI and IDE drives

	SCSI: Quantum Atlas	IDE: Quantum Fireball
GB/drive	18	26
KAPS/drive (KB accesses/s/disk)	105	78
MAPS/drive (MB accesses/s/disk)	20	15
$/drive	534	209
MB/$	34	124
KAPS/$	0.20	0.37
MAPS/$	0.04	0.07

Network access to local and remote files has high overhead and some serious performance problems. The good news is that using Gigabit Ethernet, a client can read at nearly 40MBps from a server and can write at half that rate. But there are many irregularities in the performance curves, and the overheads are quite high.

Table 11. Cpu cost of IOs on 733MHz Pentium III.

<table>
<tr><td rowspan="4">Random 8KB</td><td rowspan="2">Buffered</td><td>Read</td><td colspan="2">150 µs</td></tr>
<tr><td>Write (= 64KB read + 8KB write)</td><td colspan="2">275 µs</td></tr>
<tr><td rowspan="2">Unbuffered</td><td>Read</td><td colspan="2">70 µs</td></tr>
<tr><td>Write</td><td colspan="2">85 µs</td></tr>
<tr><td colspan="3"></td><td>Fixed (us)</td><td>Marginal (us/KB)</td></tr>
<tr><td rowspan="4">Sequential</td><td rowspan="2">Buffered</td><td>Read</td><td>12 µs</td><td>6 µs /KB</td></tr>
<tr><td>Write</td><td>12 µs</td><td>7 µs /KB</td></tr>
<tr><td rowspan="2">UnBuffered</td><td>Read</td><td>50 µs</td><td>0.5 µs /KB</td></tr>
<tr><td>Write</td><td>50 µs</td><td>0.5 µs /KB</td></tr>
<tr><td rowspan="2">Network sequential (CIFS/SMB)</td><td>UNC local</td><td>Read and write</td><td>450 µs</td><td>450 µs /KB</td></tr>
<tr><td>Mapped local</td><td>Read and write</td><td>560 µs</td><td>560 µs /KB</td></tr>
</table>

11 Acknowledgements

Thanks go to Maurice Franklin for allowing us to use some of his code as our primary test program. Maurice Franklin and Bruce Worthington also deserve thanks for allowing us to consult them with our questions.

Thanks also to Jim Gemmell and Don Slutz for giving us a better understanding of Windows 2000. Jim Gemmell also helped us better understand disk caching and scheduling algorithms.

12 References

[Horst] Horst, Robert, "TwinStor Technology: A Compelling Case for Multiple Drives in PCs, Servers, and Workstations," 3ware Technical Report TR-1999-2, 1999. http://www.3ware.com/products/TwinStor.pdf

[Nagar] Nagar, Rajeev. Windows NT File System Internals: A Developer's Guide. Sebastopol, CA: O'Reilly & Associates, 1997.

[Rambus] Rambus Based Products Benchmarks. http://www.rambus.com/products/products_benchmark_840.html

[Riedel] Riedel, Erik, Catherine van Ingen, and Jim Gray. "A Performance Study of Sequential IO on WindowsNT 4.0," Microsoft Research Technical Report MSR-TR-97-34, 1997.

[ZD] Ziff Davis WinBench 99 Benchmark. http://www.zdnet.com/zdbop/stories/main/0,8829,2326114,00.html

[Hennesy&Patterson] Hennessy, John L. and David A. Patterson. *Computer Architecture: A Quantitative Approach.* Palo Alto, CA: Morgan Kaufmann Publishers Inc., 1990.

[Richter] Richter, Jeffery. *Advanced Windows: The developer's Guide to the Win32 API for Windows NT 3.5 and Windows 95*. Redmond, WA: Microsoft Press, 1995.

[Solomon] Solomon, David A. *Inside Windows NT Second Edition*. Redmond, WA: Microsoft Press, 1998.