University of Michigan CSE-TR-200-94

Unix I/O Performance in Workstations and Mainframes

Peter M. Chen

Computer Science and Engineering Division
Electrical Engineering and Computer Science Department
University of Michigan

David A. Patterson

Computer Science and Engineering Division
Electrical Engineering and Computer Science Department
University of California at Berkeley

Abstract: Rapid advances in processor performance have shifted the performance bottleneck to I/O systems. The relatively slow rate of improvement in I/O is due in part to a lack of quantitative performance analysis of software and hardware alternatives. Using a new self-scaling I/O benchmark, we provide such an evaluation for 11 hardware configurations using 9 variations of the Unix operating system. In contrast to processor performance comparisons, where factors of 2 are considered large, we find differences of factors of 10 to 100 in I/O systems. The principal performance culprits are the policies of different Unix operating systems; some policies on writes to the file cache will cause processors to run at magnetic disk speeds instead of at main memory speeds. These results suggest a greater emphasis be placed on I/O performance when making operating system policy decisions.

Keywords: Input/Output, I/O, performance, evaluation, disk, file cache, main memory, bus, workstation, minisupercomputer, mainframe, benchmark, Unix, Sprite, Alpha AXP/3000, Convex C2, DECStation 5000, HP 730, IBM 3090, RS/6000, SPARCStation 1, SPARCStation 10.

1 Introduction

Input/output has long been the orphan of computer architecture, receiving little attention or respect. Processor performance has improved spectacularly thanks to focussed efforts, but Amdahl's Law tells us that neglecting any portion of the computation will ultimately limit overall performance. We see the impact of these limits today, with massively parallel processors unable to deliver the promised performance on some applications due to I/O bottlenecks. This I/O bottleneck looms for workstations as well. Without advances in I/O performance, advances in processor design will be wasted.

The first step in improving I/O performance is to understand what really affects performance. Alas, there is little quantitative evaluation of I/O. The main purpose of this paper is to provide such quantitative information for Unix I/O. A second purpose is to explore some of the folklore of I/O performance. Certainly one popular saying is that mainframe computers have outstanding I/O performance. This last point is particularly topical, as many customers are considering replacing their mainframes with networks of workstations.

Table 1 shows the machines and operating systems evaluated in this paper. They were selected to be representative of operating systems and hardware systems in use today. This selection was bounded by the systems available at the University of Michigan and the University of California at Berkeley and by the generosity of volunteers at other sites willing to run our benchmark.

Our evaluation shows that I/O performance is limited by the weakest link in the chain between the disk and the operating system. The hardware determines potential I/O performance, but the operating system determines how much of that potential is delivered. In particular, for Unix systems the file cache performance is critical to I/O performance. Our main observations are that file cache performance of Unix on mainframes and minisupercomputers is no better than workstations, and that file caching policy is of overriding importance. Optimized memory systems can increase read performance, but the operating system policy on writes can result in orders of magnitude differences in file cache performance.

These results are found in the next three sections. Sections 2 and 3 of this paper evaluate the performance of disks and file caches, both assuming a workload of only reads. Section 4 shows that operating system policies for writes drastically affect I/O performance.

Section 5 describes the new I/O benchmark that we used to collect performance data for this paper. This benchmark has two components: self-scaling and predicted performance. As a quick overview, this

benchmark tailors a series of workloads for each system that exercises the memory system and the disks. (Previous publications have recorded the accuracy and usefulness of this benchmark; this paper relies on that accuracy and focuses on using this benchmark to evaluate I/O performance.) Sections 2 to 4 use a 32 KB average access size, with half of the accesses being sequential, and the accesses being generated by a single process. Multiple runs determine performance for varying balances of reads and writes, with each section picking the appropriate mix. Readers with more questions about the experimental method should see Section 5 or read [Chen93a].

We conclude this paper with a summary of results, a recommendation that more emphasis in operating systems be given to I/O performance, and a short section on future directions.

2 Disk Subsystem Performance

A comprehensive evaluation of disk performance is problematic. To see why, let's review how I/O works. The data written to disks comes from main memory, and main memory is also the destination of data read from disks. Figure 1 shows there are many component between memory and disks. I/O performance is limited by the slowest component between memory and disks: it can be the main memory, the CPU-memory bus, the bus adapter, the I/O bus, the I/O controller, or the disks themselves. The trend towards open systems exacerbates the complexity of measuring I/O performance, for a particular machine can be configured with many different kinds of disks, disk controllers, and even I/O busses. In addition to

Computer	**Alpha AXP 3000**	**Dec-Station 5000**	**Dec-Station 5000**	**HP 730**	**IBM RS/ 6000**	**Sun Sparc-Station 1**	**Sun Sparc-Station 10**	**Convex C2**	**IBM 3090**
Operating System	OSF/1 1.2(10)	Sprite LFS	Ultrix 4.2A 47	HP/UX 9.01	AIX 3.1.5	SunOS 4.1	Solaris 2.1	ConvexOS 10.1	AIX/ESA on VM
Processor Model	400	200	200	730	550	1+	30	C240	600J VF
Year Proc. Shipped	1993	1990	1990	1991	1991	1989	1992	1988	1990
Approx. $ as tested	$30k	$20k	$15k	$35k	$30k	$15k	$20k	$750k	$1,000k
Proc. Clock Rate	133 MHz	25 MHz	25 MHz	66 MHz	41.7 MHz	25 MHz	33 MHz	25 MHz	69 MHz
Proc. Perf. SPECint92	75	19	19	48	34	12	45	≈ 10–20	≈ 35–45
Cache Size (Levels 1 & 2)	l1:8+8 KB l2: 512 KB	l1:64+64 KB	l1:64+64 KB	l1: 128 +256 KB	l1:8+64 KB	l1: 64 KB	l1: 20+16 KB l2: 1 MB	l1: 8+4 KB	
Memory Size	64 MB	32 MB	32 MB	64 MB	64 MB	28 MB	128 MB	1024 MB	128 MB VM partition
Memory Perf.	284-426 MB/s	100 MB/s	100 MB/s	264 MB/s	222 MB/s	80 MB/s	88 MB/s	200 MB/s	
I/O bus	Turbo-channel	SCSI-I	SCSI-I	Fast SCSI-II	SCSI	SCSI-I	SCSI-I	IPI-2	IBM Channel
Disk(s)	1 SCSI DEC RZ26	3 CDC Wren (RAID 0)	1 DEC RZ55	1 HP 1350SX	1 IBM 93x 2355	1 CDC Wren IV	1 Seagate Elite (5400 RPM)	4 DKD-502 (RAID 5)	1 IBM 3390

Table 1: Machines and operating systems evaluated in this paper. Note that the oldest machines is the Convex C240, which first shipped in 1988. AIX/ESA is run under VM because there are not enough people at that installation to justify running it native. For cache parameters, the first level 1 (l1) number is the instruction cache size and the second l1 number is the data cache size; a single number means a unified cache

the hardware components, the policies of the operating system affect I/O performance of a workload.The sheer number of combinations of components leads to less than satisfying results, for the conclusions we can draw from a few data points are limited.

If we were interested in comparing I/O performance of hardware-software systems, then ideally we would use many of the same components to reduce the number of variables. This ideal has several practical obstacles. First, few workstations manufacturers share the same operating system, CPU, or CPU-memory bus, so they may be unique to each machine. And a different CPU-memory bus requires a different bus adaptor. This leaves the I/O bus, I/O controller, and disks to be potentially in common. The problem now is that there is no standard configuration of these components across manufacturers, so it is unlikely that customers would normally buy the same configuration from different manufacturers. This leaves the evaluator the unattractive alternative of purchasing computer systems with common I/O subsystems simply to evaluate performance; few organizations have the budgets for such an effort.

Practical hope for more general results depends on an I/O benchmark becoming so popular that many combinations of components on each machine would be tested, possibly requiring dozens of different configurations per model of processor. Until an I/O benchmark becomes nearly universal, we won't have enough data points to generalize safely about classes of computers. Section 5 describes one I/O benchmark that has the potential to evaluate many different I/O systems.

We can now present our results in proper context. Figure 2 shows disk performance when reading for the machines in Table 1. The Convex minisupercomputer, with the RAID of four disks and the fast IPI-2 I/O bus, is at the top of the chart; the SPARCStation 10 is second due to its the fast single disk. The 3090 mainframe, with its single 3390 disk, comes in a surprisingly low sixth place.

Given the warnings above, we *cannot* say that IBM mainframes have lower disk performance than workstations, nor that Convex has the fastest disk subsystem. We *can* say that the IBM 3090-600J running AIX/ESA under VM performs 32 KB reads to a single IBM 3390 disk drive much more slowly than a Convex C240 running ConvexOS 10 reads 32 KB blocks from a 4-disk RAID. It is not clear what would happen, for example, if many disks were used on the IBM mainframe or if another operating system ran on the mainframe. The conclusions we draw from Figure 2 are that many workstation I/O subsystems can sustain the performance of a high-speed single disk, that a RAID disk array can deliver much higher performance, and that the performance of a single mainframe disk on a 3090 model 600J running AIX/ESA under VM is no faster than many workstations.

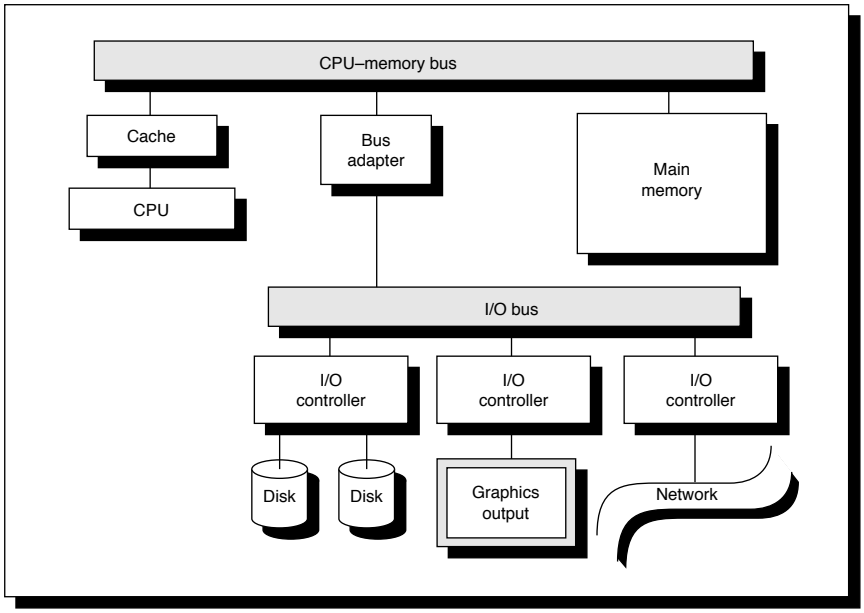

Figure 1: Basic components of an I/O subsystem. The I/O bus is also called a string, with the I/O controller called a string controller. The lowest performance component between memory and disks limits disk performance seen by the application.

3 Basic File Cache Performance

For Unix systems the most important factor in I/O performance is not how fast the disk is, or how efficiently it is used, but *whether* it is used. Unix systems use a *file cache*—a buffer in main memory—to reduce accesses to disks. [Baker91] found that in one Unix workstation environment file caches fulfill 60% of the data read requests, depending on the size of file cache. [McKusick84] found that accesses to file meta data had even higher success rates. Since main memory is much faster than disks, file caches yield a substantial performance improvement, and are found in every Unix operating system.

One portion of our benchmark varies its workload to be sure to access only the file cache when measuring performance of a system; Figure 3 shows this file cache performance for the machines of Table 1. The first thing to notice is the change in scale of the chart: machines read from their file caches 3 to 25 times faster than from their disks. The performance of the file cache is determined by the processor, cache, CPU-memory bus, main memory, and operating system. Except for the size of memory and perhaps the operating system, there is little choice in these components when selecting a computer. Hence observations about commercial systems can be drawn about file cache performance with much more confidence than with the disks, since this portion of the system will be common at most sites.

The biggest surprise is that the mainframe and minisupercomputers did not dominate this chart, given their much greater cost and reputation for high-bandwidth memory systems and CPU-memory busses. The four top performers are DEC Alpha, IBM RS/6000 model 550, HP 730, and IBM 3090 running AIX, all with file cache performance of approximately 30 MB/s. This chart also shows rapid file cache improvement in workstations. For example, both the SPARCStation 10 and DEC Alpha AXP/3000 are more than four times faster than their predecessors, the SPARCStation 1 and the DECStation 5000.

In addition, Figure 3 shows the impact of operating systems on I/O performance; the Sprite operating system offers 1.8 times the file cache performance of Ultrix running on the same DECStation 5000 hard-

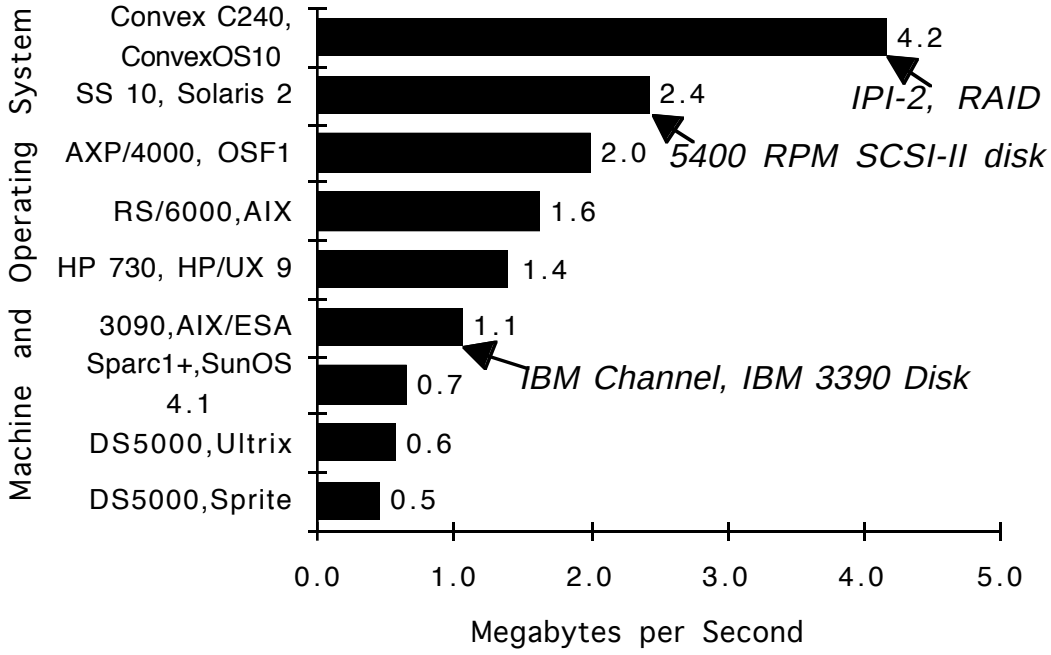

Figure 2: Disk performance for the machines in Table 1. These results are for reads of 32 KB. Except where otherwise noted in the figure, the machines use SCSI I/O buses, SCSI controllers, and SCSI disks that rotate at 3600 RPM. Read performance is from 1.5 to 2.8 times faster than write performance, except for the Sprite system. Sprite's Log-Structured File System is optimized for writes, which are 1.3 times faster than reads on the DS 5000. (Section 5 explains the measurement method of data collection; we started with the measured 100% read performance at nominal access sizes and then interpolated to determine performance for a common 32 KB access size. The only numbers adjusted by more than 15% was for the Convex; Table 3 shows the multipliers used for interpolation.)

ware. Sprite does fewer copies when reading data from the file cache than does Ultrix [Kay92], hence its higher performance.

4 The Impact of Operating System Policies on File Cache Performance

Given that Unix systems have common ancestors, we expected that the operating system policies towards I/O to be the same on all machines. Instead, we find that different systems have very different I/O policies, and some policies alter I/O performance by factors of 10 to 100. These machines are aimed largely at the same customers running the same applications, which calls into question the low performance of some of these policies. One reason for this wide variation may be the sparsity of published results comparing file cache performance on several workstations and operating systems ([Ousterhout90] is one of the few recent examples), and so there has been little quantitative input into these decisions.

4.1 File Cache Size

Since main memory must be used for running programs as well as for the file cache, the first policy decision is how much main memory should be allocated to the file cache. The second is whether or not the size of the file cache can change dynamically. Early Unix systems give the file cache a fixed percentage of main memory; this percentage is determined at the time of system generation, and is typically set to 10%. More recent systems allow the barrier between file cache and program memory to vary, allowing file caches to grow to be virtually the full size of main memory if warranted by the workload [Tanenbaum85, Stern94]. File servers, for example, will surely use much more of their main memory for file cache than will most client workstations.

Our benchmark varies the amount of data and measures performance to determine the maximum file cache size for each system. Figure 4 shows these maximum file cache sizes, both in absolute size and as a percentage of main memory The reason for the large variation in percentage of main memory is the file cache size policy. HP/UX version 8, Ultrix, and AIX/ESA all reserve small, fixed portions of main memory

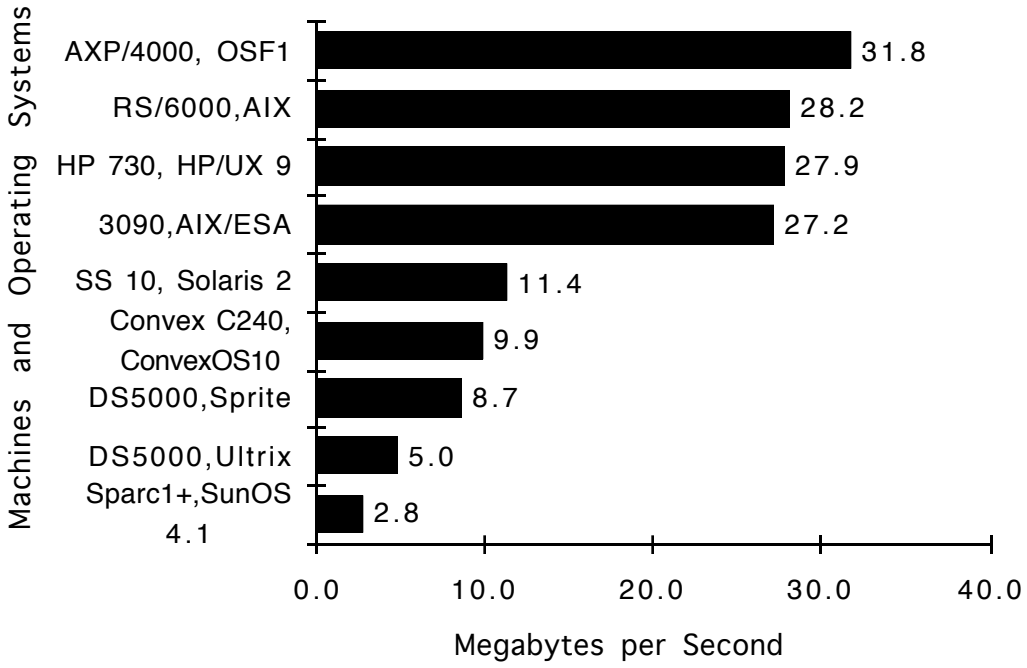

Figure 3: File cache performance for machines in Table 1. This plot is for 32 KB reads with the number of bytes touched limited to fit within the file cache of each system. (Section 5 explains the measurement method of data collection; we started with the measured 100% read performance for accesses within the respective file caches at nominal access sizes and then interpolated to determine performance for a common 32 KB size. Table 3 shows the factors; the only numbers adjusted by more than 10% were for the Convex.)

for the file cache. HP/UX version 9, in contrast, supports dynamically varying file cache size, leading to an 10-fold larger file cache for the HP 730.

[Baker91] demonstrated that Unix workloads benefit from larger file caches, so this fixed-size policy surely hurts I/O performance. Note that Sprite, running on the same hardware as Ultrix, has more than six

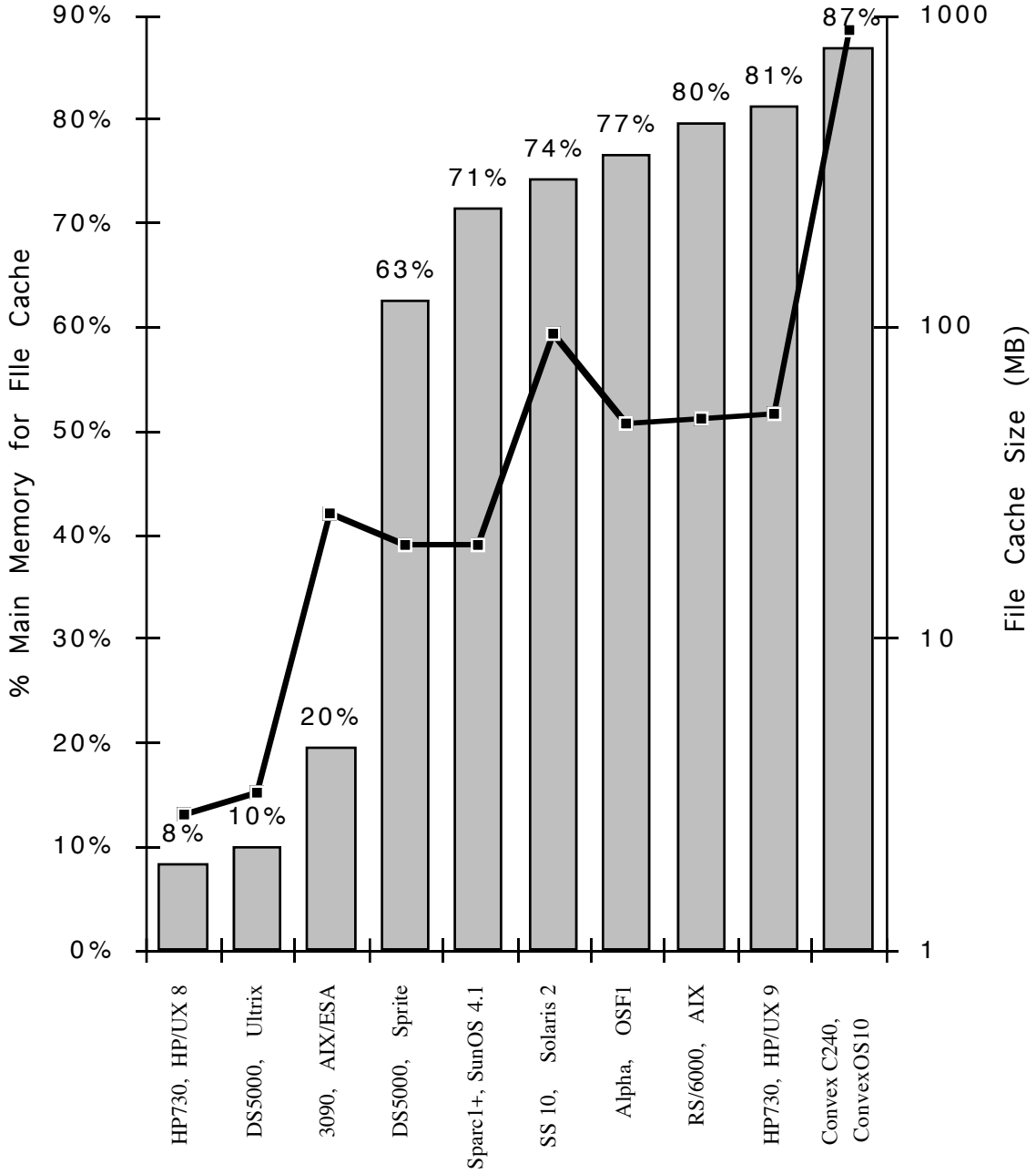

Figure 4: File cache size. The bar graph shows the maximum percentage of main memory for the file cache in the I/O workload of Section 5, while the line graph shows the maximum size in MB using the log scale on the right. Thus the HP 730 HP/UX version 8 uses only 8% of its 32 MB main memory for its file cache, or just 2.7 MB, and the Convex C240 uses 87% of its 1024 MB main memory, or 890 MB, to be its file cache. (Section 5 explains that the Sprite file cache really has two components; 63% is the size of the larger one.)

times the file cache size, and that the SPARCStation 1 has a file cache almost as large as the IBM 3090, even though the mainframe has four times the physical memory of the workstation. When the flexible file cache boundary policy is combined with large main memories, we can get astounding file caches: the Convex C240 file cache is almost 900 MB! Thus workloads that would require disk accesses on other machines will instead access main memory on the Convex.

4.2 Write Policy

Thus far we have unrealistically left the O out of I/O. There is often some confusion about the definition and implications of alternative write strategies for caches. To lessen that confusion, we first review write policies of processor caches. (Readers interested in more about caches can consult textbooks such as [Hennessy90].)

The simplest write policy is *write-through*; writes update the cache and the next level of the memory hierarchy behind the cache. To offer reasonable write performance, almost all write-through caches use a *write buffer*; the write buffer allows the processor to continue immediately after writing the data to the cache and the buffer; the buffer then writes to the next level of the memory hierarchy while the processor does other work. The operating systems community uses the terms *asynchronous writes* or *delayed writes* for writes that allow the processor to continue after updating the write buffer. If writes occur infrequently then this buffer works well; if writes are frequent, then the processor may eventually have to stall until the write buffer empties, limiting the speed to that of the next level of the memory hierarchy. Note that a write-buffer does not reduce the number of writes to the next level; it just allows the processor to continue while I/O is in progress, provided the write-buffer is not full. Because all writes must go to disk in a write-through file cache, we expect the write throughput of such a system to be limited by disk speeds.

The alternative write policy to write-through is called *write-back*; writes simply update the cache block, setting a dirty bit so that the processor knows it must write the block to memory when it is replaced. Multiple updates to the same location simply re-write the appropriate file cache blocks, cancelling earlier writes to that location; write-through caches, in contrast, send every write to the next level of the memory hierarchy.[1] Because this ability to cancel writes drastically changes system performance for writes, and because there has been confusion about the distinction between write-through with write buffers versus write-back, we categorize systems as being either *write-through* (all writes must go through to disk) or *write-cancelling*.

The effectiveness of caches for writes also depends on the policy of flushing dirty data to the disk; to protect against losing information in case of failures, a flushing process (also called a write-back, update, or sync daemon) will occasionally issue a command that forces modified data out of the cache and onto the disk. Most Unix operating systems have a policy of periodically writing dirty data to disk to limit the amount of data that can be lost in a crash; typically, the window is 30 seconds [Leffler89]. Though the flushing process causes more data to be written to disk, it should not have a major effect on the performance of applications that stay in the file cache. This is because such applications touch only file cache blocks, while the flushing process, operating in the background, mainly uses the disk. That is, the application should never have to wait for the flushing process.

Figure 5 shows the file cache performance as we vary the mix of reads and writes[2]. Clearly HP/UX (both with a 30 second and infinite flushing interval), Sprite, and OSF/1 with write-cancellation use a write-cancelling cache because they perform writes at a speed faster than the disk system can sustain. While it is hard to see using this figure, the Sun OS 4.1 write performance is nearly as fast as reads and much faster than disks, so it also cancels writes. We can tell that the other operating systems use write-through because their write performance matches disk speed. Note that three of the highest performers in Figure 3—RS/6000, IBM 3090, and Alpha AXP/3000 (with write-through)—fall to the back of the pack unless the percentage of reads is over 90%.

A write-cancelling cache (independent of the flushing interval) should perform at memory speeds for both reads and writes, hence their lines should be fairly flat in Figure 5. Of the systems in Figure 5 that cancel writes, this holds true for Sprite, SunOS 4.1, and HP/UX with an infinite flushing interval. The others (HP/UX with 30-second flushing and OSF/1 with write-cancellation and infinite flushing interval) per-

1. (The confusion alluded to above concerns the difference between write buffers and write-back; some of the developers we talked to insisted that their systems were using write-back, when they were using write-through with a write buffer.)

2. For OSF/1, we show performance with the system configured to be write-through (the default) and also configured as write-cancelling.

form writes much slower than reads. To investigate this further, we lengthened the flushing interval for HP/UX to infinity; this caused a huge increase in write performance: from 8 MB/s to 30 MB/s! Note that both these numbers are higher than disk performance, yet somehow the flushing process slows performance by a factor of four. In further measurements, we found that this phenomenon is less pronounced

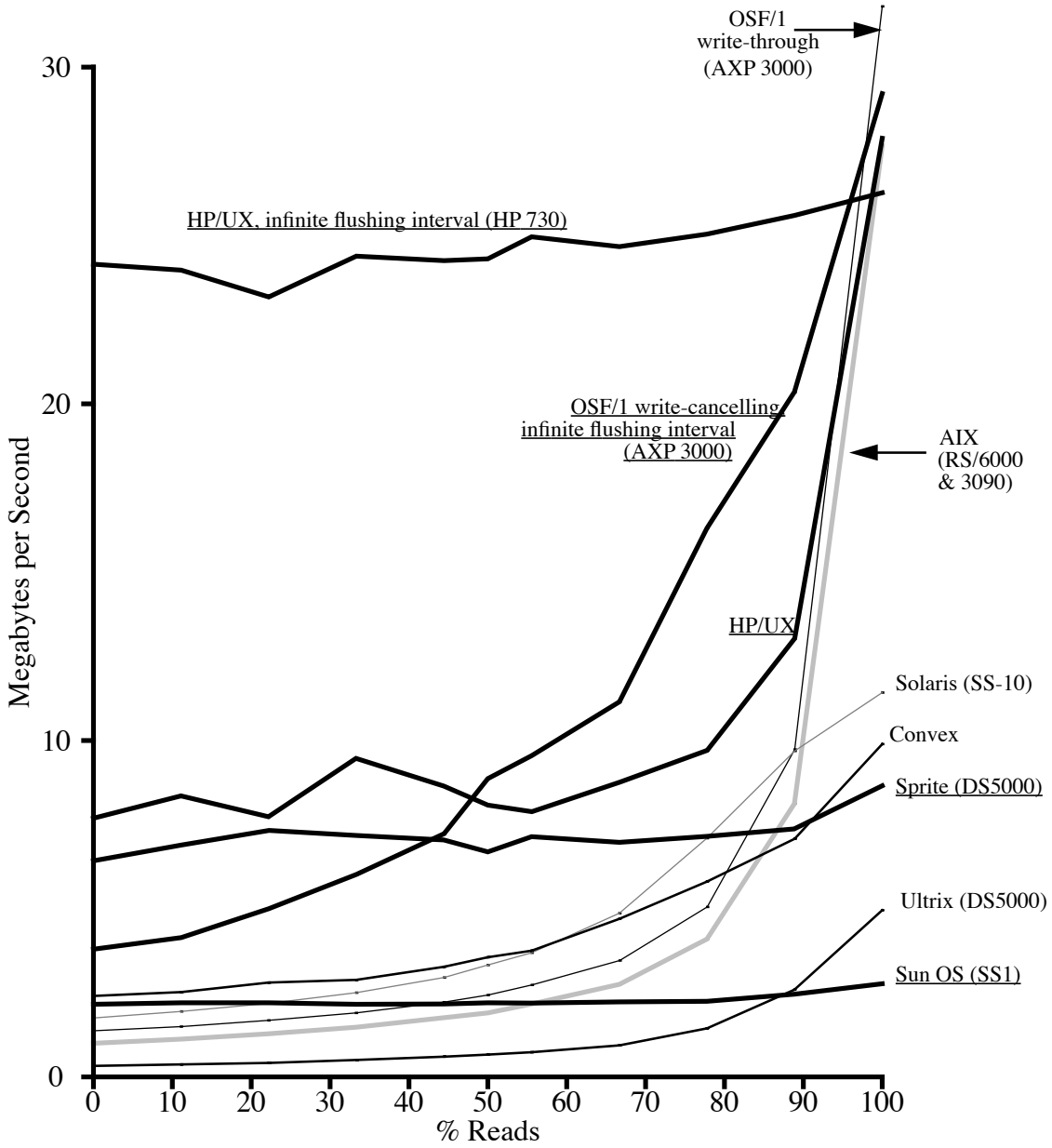

Figure 5: File cache performance vs. read percentage. 0% reads means 100% writes. Note that the high performance of the file caches of the RS/6000, 3090, and AXP/3000 (write-through) are only evident for workloads with ≥ 90% reads. Systems with write-cancelling performance are shown in bold with underlined labels. Access sizes are 32 KB. (Section 5 explains the measurement method of data collection; we started with the measured performance for accesses within the respective file caches at nominal access sizes and then interpolated to determine performance for a common 32 KB size. The only numbers adjusted by more than 10% was for the Convex; Table 3 shows the multipliers. Section 5 also explains the Sprite file cache really has two components; this is performance for the smaller one.)

when touching less MB (Figure 8). We hypothesize that once the flushing process schedules a block to be written to disk, it locks the file cache block and any future application writes to that block must wait for the disk write to complete. Applications that touch less MB create less work for the flushing process to do, hence the flushing process does not slow those application as drastically.

The short lives of files means that files will be deleted or overwritten and so their data need not be written to disk. Baker *et al* found that this 30-second window captures 65% to 80% of the lifetimes for all files [Baker91]. Their paper claims this percentage represents only small files and so almost all data is written through. In a correction to that claim, Hartman and Ousterhout reported that 36% to 63% of the bytes written do not survive a 30 second window; this number jumps to 60% to 95% in a 1000 second window [Hartman93]. According to another study, a small, non-volatile file cache can reduce write traffic by 50% [Baker92]. Given that such short lifetimes mean that file cache blocks will be rewritten, we recommend that all operating system developers consider using a write-cancellation policy, enabling writes to the file cache to perform at memory speeds.

4.3 Write Policy for Client/Server Computing

Thus far we have been ignoring the network and the possibility that the files exist on a file server. We tested this performance with two experiments: one experiment used one SPARCStation 1+ as client and another as a server, and a second experiment using a HP 720 as a client and a HP 730 as a server. Both client-server pairs were connected by an Ethernet local area network. Figure 6 shows file cache performance versus the percentage of reads when the files are reached over the networks.

This experiment brings us to the issue of consistency of files in multiple file caches. The concern is that multiple copies of files in the client caches and on the server create the possibility that someone will access the wrong version of the data. This brings us to a policy decision: how to ensure that no one accesses stale data. The NFS solution, used by SunOS, is to make all client writes go to the server's disk when the file is closed [Sandberg85]. Taking an outsider's perspective, it seems inconsistent that a 30 second delay would be satisfactory for writes to local disk but not for writes to the server disk. Hence HP/UX offers an alternative network protocol, called DUX, which allows client-level caching of writes. The server keeps track of potential conflicts and takes the appropriate action only when the file is shared and someone is writing it. Using a shared bus multiprocessor as a rough analogy to our workstations on the local area network, DUX offers write-cancellation with cache coherency while NFS does write through without write buffers. Readers interested in more details of file cache policies should see [Nelson88].

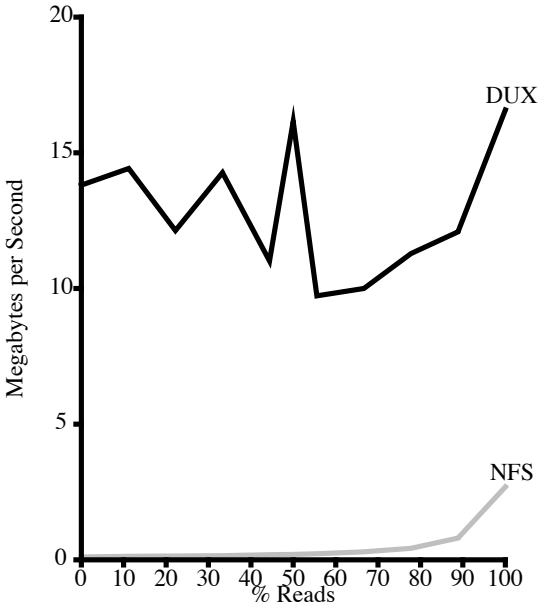

Figure 6: File cache performance vs. percentage of reads for client-server computing. Using a SPARCStation 1+ for both a client and the file server running NFS is literally a hundred times slower than a HP 720 client using a HP 730 server running the DUX network file system. (These are results as measured, with nominal average access sizes of 11 KB for Sparc and 22 KB for HP. Interpolating to a common 32 KB size would increase performance of Sparc by 8% and HP by 6%.)

The 100% read case—the rightmost portion of the graph—shows the differences in performance of the HP and Sparc hardware, where HP is 6 times faster. The rest of the graph shows the differences in performance due to the write policies of their operating systems. The HP system is the clear winner; it is literally 100 times faster than the Sparc system in workloads with mostly writes, and still 25 times faster even when only 20% of the accesses are writes. Put another way, for all but the most heavily read-oriented workloads, the SPARCStation client operates at disk speed while the HP client runs at main memory speed. Adding non-volatile RAM such as Legato System's Prestoserve board, would increase NFS performance somewhat [Lyon90]. However, writing through to NVRAM on a server is still limited by network speeds. In this experiment, Ethernet would limit write performance of NFS to 1 MB/s.

5 A New I/O benchmark: Self-scaling and Predicted Performance

This section describes the I/O benchmark used to collect the data seen in the prior sections.

Current I/O benchmarks suffer from several chronic problems: they quickly become obsolete, they do not stress the I/O system, and they do not help in understanding I/O system performance [Chen93b]. This led to a new approach to I/O performance analysis [Chen93a].

The first step is a *self-scaling benchmark* that automatically and dynamically adjusts aspects of its workload according to the performance characteristics of the system being measured. By doing so, the benchmark automatically scales across a wide range of current and future systems.This scaling is more general than the scaling found in TPC-B [TPC90] and LADDIS [Wittle93], for scaling here varies more than the load on the system.

This first step aids in understanding system performance by reporting how performance varies according to each of five workload parameters; these parameters determine the first-order performance effects in I/O systems. The benchmark reports throughput; however, response time can easily be calculated for any given workload according to Little's Law, since concurrency and request size are known. Here are the five I/O parameters:

1. *Number of unique bytes touched*: This is the number of unique data bytes read or written in a workload; essentially it is the total size of the data set. It models locality using the LRU stack model, with the average access depth in the LRU stack set at half this parameter [Rau79]. The benchmark varies this parameter to reveal file cache size (Figure 4).
2. *Percentage of reads*: The percentage of writes is 100% minus this percentage. Once the size of the files cache were determined, we measured its performance as the read percentage varied to plot Figures 5 and 6. We also used the 100% read performance results for the disks and file caches to plot Figures 2 and 3.
3. *Average I/O request size*: We chose sizes from a Bernoulli distribution, with a coefficient of variation equal to 1.
4. *Percentage of sequential requests*: This is the percentage of requests that sequentially follow the prior request. When set at 50%, half of the accesses are to the next sequential address.
5. *Number of processes*: This is the concurrency in the workload, that is, the number of processes simultaneously issuing I/O. Load can further be adjusted by varying the amount of time a process thinks between I/Os (CPU think time). For this paper, we set CPU think time to zero to maximally exercise the I/O system. This allows us to look at how well these memory system designs and operating system policies will work with the much faster CPUs of the future.

The general idea of the benchmark is to vary a single parameter while the other four parameters remain at a fixed, nominal value.

For the machines reported in this paper, the nominal values are 50% reads, 50% sequential accesses, and a single process issuing I/O requests. Because of the very different performance for file cache and disk accesses, the benchmark automatically picks two values for number of bytes accessed. (If there were a third region of very different performance, it would add that value to be explored; see [Chen93a] for more details on picking parameters.) Table 2 shows the nominal values selected for the machines and operating systems in this paper.

The parameters of Table 2 can be considered workloads that use the resources of the machine effectively. For example, the 4-disk RAID system of the Convex achieves significantly higher bandwidth with larger average access sizes, hence the larger nominal access size of 120 KB. Another example is the modest size of main memory and fixed file cache size policy of the DECStation 5000 running Ultrix are reflected in the number of bytes touched: just 2 MB.

The resulting I/O performance is then plotted for each of the parameters. These plots can contain some surprises. For example, Figure 7 plots performance versus the first parameter, the number of unique bytes, for the Sprite and Ultrix operating systems running on the DECStation 5000. The Sprite curve shows *three* performance plateaus: 0 to 5 MB, 5 to 20 MB, and > 20 MB. This middle plateau was auto-

matically uncovered by the self-scaling benchmark; it discovered that reads were much faster than writes in this plateau. This is due to the effective write cache of Sprite's Log Structure File System (LFS) being much smaller than the read cache, so reads are cached in this region while writes are not. The write cache of LFS is smaller because LFS limits the number of dirty cache blocks to avoid deadlock during cleaning. For writes the effective file cache size is only 5-8 MB, while for reads it is 20 MB. This result surprised the Sprite developers, and it demonstrates the value of a benchmark that automatically explores the parameter space over a benchmark with a fixed set of parameters.

Figure 8 shows the performance for recent workstations and mainframes using the nominal parameters values collected by the self-scaling benchmark, once again as a function of unique bytes touched. These plots give insights into appropriate workloads and resulting performance. The width of the high-performance parts of the curves is determined by the size of the file cache. Under default operating system configurations, the HP 730 offers the highest performance for applications that touch a small amount of

	DEC AXP/ 3000	DEC DS 5000, Sprite	DEC DS 5000, Ultrix	HP 730	IBM RS/600 0	Sun SS 1+	Sun SS 10	Convex C2	IBM 3090
Average Access Size (KB)	21	32	13	40	32	20	27	120	19
No MB Touched: File Cache	21	2, 15	2	39	26	12	48	450	13
No. MB Touched: Disk	75	36	6	75	80	42	157	1376	42

Table 2: The values of the parameters selected automatically by the self-scaling benchmark for the systems in Table 1. The nominal values of the other parameters are 50% reads, 50% sequential accesses, and one process. (As explained below, the Sprite DS 5000 file cache has two sizes.)

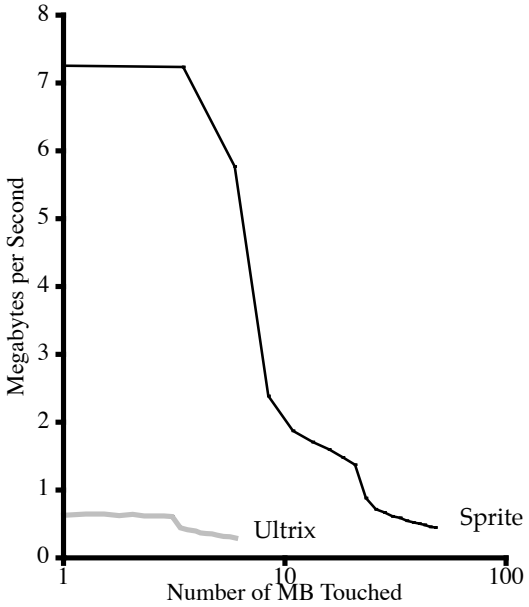

Figure 7: I/O performance of two operating systems on the DECStation 5000 varying the number of megabytes accessed. Note that the X axis is on a log scale. The small size and write-through policy of the Ultrix file cache explain its low performance. The Sprite curve shows three performance plateau; see the accompanying text for an explanation. (Results are as measured with 50% reads; adjusting for common access sizes would make no relative difference in these two curves.)

data[3]; yet workloads that would need to go to disk on other systems can be satisfied by the very large file cache of the Convex.

The self-scaling benchmark increases our understanding of a system and scales the workload to remain relevant as technology advances. It complicates the task of comparing results from two systems, however. The problem is the benchmark may choose different workloads on which to measure each sys-

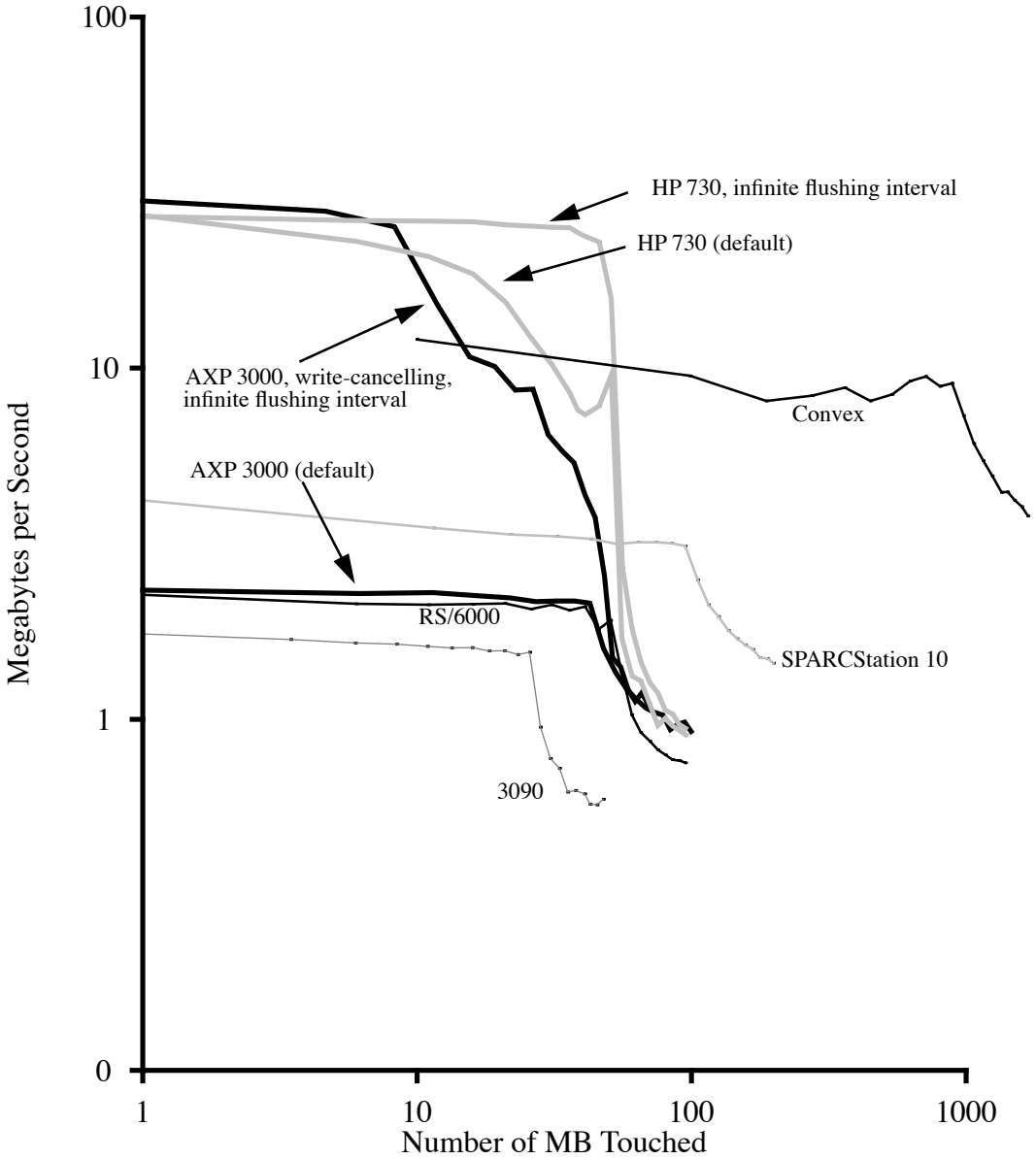

Figure 8: Performance vs. megabytes touched for recent workstations and the mainframes in this paper. Note the log-log scale. In contrast to other figures, these results use the nominal values selected by the self scaling benchmark. The primary difference is the use of 50% reads and an average access size of 120 KB for the Convex; adjusting for a common access size would halve Convex performance but make little change to the other lines in this plot.

3. Without the flushing process, applications on the HP 730 can get very high performance (30 MB/s) with up to 50 MB of file space.

tem. Also, although the output graphs from a self-scaling evaluation apply to a wider range of applications than today's I/O benchmarks, they stop short of applying to all workloads.

Hence the second part of this new approach is to estimate performance of other workloads. We estimate performance for unmeasured workloads by assuming the *shape* of a performance curve for one parameter is independent of the values of the other parameters. This assumption leads to an overall performance equation of

$$Performance(X, Y, Z, \ldots) = Performance(X_{nominal}, Y_{nominal}, Z_{nominal}, \ldots) \times f_X(X) \times f_Y(Y) \times f_Z(Z) \times \ldots$$

where X, Y, Z, ... are the parameters. Thus to use the same average access sizes in Figures 2, 3, and 5 we adjusted performance in the self-scaling benchmark plot by the ratio of the performance at 32 KB to the performance at the nominal access size (see Table 3 in the Appendix). [Chen93a] has shown that this technique yields accurate performance estimates, within 10% for most workloads.

This paper is a perfect example of the use of predicted performance. We came into this paper without preconceived notions about what the results should be, much less what values to set the nominal parameters. The results from a single run of the self-scaling benchmark were used to discover the problems about write policy for file caches. Although it would be possible to go back to all 11 systems and rerun the workloads with a common block size and percentage of reads, thereby making the collection of data of other parameters "unnecessary", there is little reason to do so. The numbers collected in the original pass are close for all but one system. It seems difficult to justify such a new run: it would take weeks to arrange time to remeasure 11 systems, and new measurements would not change our conclusions.

Readers interested in a copy of the self-scaling benchmark can obtain it via anonymous FTP from the Internet host `ftp.cs.Berkeley.EDU` under the directory `ucb/benchmarks/pmchen`.

6 Conclusion

Using a new approach to I/O benchmarking—self-scaling evaluation and predicted performance—we have evaluated I/O performance of 11 hardware/software systems. As mentioned in the introduction, I/O is limited by the weakest link in the chain between the disk and the operating system. The hardware determines the potential I/O performance, but the operating system determines how much of that potential is delivered. We have found differences of factors of 100; given processor battles are won or lost on factors of 2, I/O seems ripe for attention and respect. This benchmark exercises I/O systems exclusively, and while many systems are not yet I/O bound, the continuing 50%per year advance in processor speed suggests most will soon become so.

As a result of the studies in this paper, we conclude:

- File cache performance in workstations is improving rapidly, with over four-fold improvements in three years for DEC (AXP/3000 vs. DECStation 5000) and Sun (SPARCStation 10 vs. SPARCStation 1+).
- File cache performance of Unix on mainframes and minisupercomputers is no better than on workstations.
- Current workstations can take advantage of high performance disks.
- RAID systems can deliver much higher disk performance.
- File caching policy determines performance of most I/O events, and hence is the place to start when trying to improve I/O performance.

Given the wide range of performance from systems aimed at the same market, we recommend that operating system developers take a closer, quantitatively-based look at their policies concerning file cache sizes and write-cancellation policies.

7 Future Directions

We hope that this paper inspires others to stress and evaluate the I/O system beyond the file cache. For example, determining the maximum number of disks that each system could support would offer insight into the capacity of these systems. As we mentioned in Section 2, this is no small effort. It would also be important to understand which architectural options—interleaved memory, block copy hardware, high-performance DRAMs—will give the highest potential file cache performance. And of course it would be interesting to expand the study in this paper to more machines and operating systems.

One potential future direction for the benchmark is changing how manufacturers and users evaluate I/O. If manufacturers published its results over a range of I/O options for a particular system, users could use predicted performance to estimate, *without further measurements*, the I/O performance of their specific workloads. As the price of each configuration is easily calculated, the price/performance of systems that

match the users' needs are also easily calculated, simplifying the trade-off between number of disks, faster disks, more main memory, and so on.

8 References

[Baker91] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout. Measurements of a Distributed File System. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 198–212, October 1991.

[Baker92] Mary Baker, Satoshi Asami, Etienne Deprit, John Ousterhout, and Margo Seltzer. Non-Volatile Memory for Fast Reliable File Systems. In *Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pages 10–22, October 1992.

[Chen93a] Peter M. Chen and David A. Patterson. A New Approach to I/O Performance Evaluation–Self-Scaling I/O Benchmarks, Predicted I/O Performance (conference version). In *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 1–12, May 1993.

[Chen93b] Peter M. Chen and David A. Patterson. Storage Performance–Metrics and Benchmarks. *Proceedings of the IEEE*, 81(8):1151–1165, August 1993.

[Hartman93] John H. Hartman and John K. Ousterhout. Letter to the Editor. *Operating Systems Review*, 27(1):7–9, January 1993.

[Hennessy90] John. L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1990.

[Kay92] Jonathan Kay and Joseph Pasquale. A Performance Analysis of TCP/IP and UDP/IP Networking Software for the DECStation 5000. Technical Report Sequoia Technical Report 92/18, University of California at Berkeley, December 1992.

[Leffler89] Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.3BSD Unix Operating System*. Addison-Wesley Publishing Company, 1989.

[Lyon90] Bob Lyon and Russel Sandberg. Breaking Through the NFS Performance Barrier. Technical report, Legato Systems, Inc., 1990.

[McKusick84] Marshall Kirk McKusick, William N. Joy, Samuel J. Leffler, and Robert S. Fabry. A Fast File System for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, August 1984.

[Nelson88] Michael N. Nelson, Brent B. Welch, and John K. Ousterhout. Caching in the Sprite Network File System. *ACM Transactions on Computer Systems*, 6(1):134–154, February 1988.

[Ousterhout90] John K. Ousterhout. Why aren't operating systems getting faster as fast as hardware? In *Proceedings USENIX Summer Conference*, pages 247–256, June 1990.

[Rau79] B. Ramakrishna Rau. Program Behavior and the Performance of Interleaved Memories. *IEEE Transactions on Computers*, C-28(3):191–199, March 1979.

[Sandberg85] Russel Sandberg, David Goldbert, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and Implementation of the Sun Network Filesystem. In *Proceedings of the Summer 1985 Usenix Conference*, 1985.

[Stern94] Hal Stern. Virtual Amnesia. *Advanced Systems (previously SunWorld)*, pages 17–20, February 1994.

[Tanenbaum85] Andrew S. Tanenbaum and Robbert Van Renesse. Distributed Operating Systems.

Computing Surveys, 17(4):419–470, December 1985.

[TPC90] TPC Benchmark B Standard Specification. Technical report, Transaction Processing Performance Council, August 1990.

[Wittle93] Mark Wittle and Bruce E. Keith. LADDIS: The next Generation in NSF File Server Benchmarking. In *Proceedings of the Summer 1993 USENIX Technical Conference*, 1993.

9 Appendix: Other I/O Measurement

	DEC Alpha AXP/3000	DEC DS 5000, Sprite	DEC DS 5000, Ultrix	HP 730	IBM RS/6000	Sun SS 1+	Sun SS 10	Convex C2	IBM 3090
Nominal Average Access Size (KB)	21	32	13	40	32	20	27	120	19
File Cache Performance at nominal Size (MB/sec)	2.2	6.7	0.6	7.7	2.1	2.2	3.3	8.1	1.6
File Cache Performance at 32 KB Size(MB/sec)	2.4	6.7	0.7	8.1	2.1	2.2	3.3	3.6	1.8
File Cache Performance Multiplier to adjust to 32 KB size	1.11	1.00	1.04	1.06	1.00	1.00	1.02	0.44	1.10
Disk Performance at nominal Size (MB/sec)	1.1	0.5	0.3	1.1	0.8	0.4	1.6	4.3	0.6
Disk Performance at 32 KB Size(MB/sec)	1.2	0.5	0.4	.97	0.8	0.4	1.7	1.9	0.6
Disk Performance Multiplier to adjust to 32 KB size	1.15	1.00	1.13	0.91	1.00	1.05	1.03	0.45	0.99

Table 3: These are the file cache and disk performance values at the nominal access size for 50% reads, 50% sequential, one process. The multipliers in the fourth and last rows were used to adjust Figures 2, 3, and 5 to determine performance at the same 32 KB average access size.The number of unique bytes used to ensure file cache or disk accesses are listed in Table 2.

10 Acknowledgments

The work described here was supported in part by the National Science Foundation under grants CCR-8900029 and MIP-8715235, and the National Aeronautics and Space Administration and the Defense Advanced Research Projects Agency under contracts NAG2-591 and DABT63-92-C-0007.

We would like to thank the following people for either running the benchmark or giving us access to the machines: Jon Forrest, Karem Sakallah, and Mike Riepe (Alpha AXP); Eugene Miya, Dave Tweten, and Tom Woodrow (Convex); Carl Staelin (HP 730); Shashi Sathaye (IBM 3090); Sid Bytheway (RS/6000); and Ed Kelly, Rajiv Khemani, and Jim Voll (SPARCStation 10). We would also like to thank the following people for making helpful comments on a draft of this paper: Remzi Arpaci, Mary Baker, Ann Drapeau, Mike Dahlin, John Hartman, John Ousterhout, Ken Shirriff, and Randy Wang.