Characteristics of I/O traffic in personal computer and server workloads

by W. W. Hsu
A. J. Smith

Understanding the characteristics of I/O traffic is increasingly important as the performance gap between the processor and disk-based storage continues to widen. Moreover, recent advances in technology, coupled with market demands, have led to new and exciting developments in storage systems, particularly network storage, storage utilities, and intelligent self-optimizing storage. In this paper, we empirically examine the physical I/O traffic of a wide range of server and personal computer (PC) workloads, focusing on how these workloads will be affected by the recent developments in storage systems. As part of our analysis, we compare our results with historical data and re-examine some rules of thumb (e.g., one bit of I/O per second for each instruction per second of processing power) that have been widely used for designing computer systems. We find that the I/O traffic is bursty and appears to exhibit self-similar characteristics. Our analysis also indicates that there is little cross-correlation between traffic volumes of server workloads, which suggests that aggregating these workloads will likely help to smooth out the traffic and enable more efficient utilization of resources. We discover that there is significant potential for harnessing "free" system resources to perform background tasks such as optimization of disk block layout. In general, we observe that the characteristics of the I/O traffic are relatively insensitive to the extent of upstream caching, and thus our results still apply, on a qualitative level, when the upstream cache is increased in size.

Processor performance has been increasing at the rate of 60 percent per year[1] while disk access time, being limited by mechanical delays, has been improving by less than 10 percent per year.[2] Compounding this widening performance gap between processor and disk storage, disk capacity has been growing by more than 60 percent per year,[2] so that each disk is responsible for storing and retrieving increasing amounts of data. The overall result of these technology trends, which show no signs of easing, is computer systems increasingly bottlenecked at disk-based storage. The key to overcoming this bottleneck is to understand how storage is actually used, so that new optimization techniques and algorithms can be designed.

A focused examination of the I/O characteristics of real workloads is also needed to determine how recent developments in the storage industry will impact I/O performance. First, storage is increasingly attached to some network, so that it can be shared and accessed directly by multiple servers (e.g., network-attached storage [NAS] for file storage and storage area networks [SANs] for block storage). To achieve good performance for such network storage, the I/O traffic patterns must be known, and the network must be optimized for such patterns. Second, it is anticipated that the storage now managed by various entities within an enterprise will become consolidated through the use of storage utilities or stor-

age service providers (SSPs). Whether such pooling of resources leads to a more efficient use of resources depends on the I/O characteristics of the workloads and, in particular, on whether the workloads are independent. In practice, we will need rules of thumb for the storage and performance requirements of each group of users, as well as realistic traffic models. Third, the rapid growth of processing power available in the storage system[3] makes it possible to build intelligent storage systems that can dynamically optimize themselves for the actual workload.[4] The design of these systems requires a good understanding of how real workloads behave.

In this research, therefore, we empirically examine how storage is used in various personal computers (PCs) and servers from the perspective of evaluating these new storage opportunities. A total of 18 traces gathered from a wide range of environments are examined. We focus in this paper on analyzing the physical I/O traffic and, specifically, we analyze (1) the I/O intensity of the workloads and the overall significance of I/O in the workloads, (2) how the I/O load varies over time and how it will behave when aggregated, and (3) the interaction of reads and writes and how it affects performance. We compare our results with historical data in order to identify trends and to revalidate rules of thumb that are useful for systems design and sizing. To make our results more broadly applicable, we also study the effect of increased upstream caching on our analysis. In a companion paper, we examine how these real workloads are affected by disk improvements and I/O optimizations such as caching and prefetching.[5] The insights gained from this research are instrumental to the block reorganization technique outlined in Reference 4.

The rest of this paper is organized as follows. In the next section we present a brief overview of previous work in characterizing I/O behavior. In the following section, we discuss our methodology and describe the traces that we use. In the next three sections, we analyze the I/O traffic of our workloads in detail. Specifically, we discuss (1) intensity of I/O, (2) variability of I/O traffic over time, and (3) the interaction of reads and writes. In the last section we include our concluding remarks. Because of the huge amount of data that is involved in this study, we can only highlight here some of the results of our research. More detailed graphs and data are presented in Reference 6, as are some of the more involved mathematical analyses.

Related work

I/O behavior at the file system level has been previously characterized in some detail (see for example References 7, 8, and 9). There have also been several studies of the logical I/O characteristics of large database and scientific systems; see References 10 and 11 for a brief bibliography. Compared to the analysis of I/O behavior at the logical level, physical I/O characterization has received much less attention. Part of the reason is that storage-level characteristics are sensitive to the file system and buffer pool design and implementation, so that the results of any analysis are less broadly applicable. But this is precisely the reason to analyze the physical I/O characteristics of many different systems.

Traces of the physical I/Os in large IBM mainframe installations[12] and production VAX/VMS systems[13,14] have been used to study design issues in disk caches. There has also been some analysis of the physical I/O characteristics of UNIX** systems[15] and Novell NetWare** file servers[16] in academic/research environments. Even though PCs running various versions of Microsoft Windows** are now an integral part of many office activities, to the best of our knowledge there has been no published systematic analysis of how physical storage is used in such systems.

Methodology

Trace data can generally be gathered at different levels in the system, depending on the reason for collecting the data. For instance, to evaluate cache policies for the file system buffer, I/O references have to be recorded at the logical level, before they are filtered by the file system buffer. In general, collecting trace data at the logical level reduces dependencies on the system being traced and allows the trace to be used in a wider variety of studies, including simulations of systems somewhat different from the original system. To study physical storage systems, for example, we could filter a logical trace through models of the file system layer in order to obtain a trace of physical I/Os. A commonly used method for obtaining such a logical trace is to insert a filter driver that intercepts all requests to an existing file system device and that records information about the requests before passing these requests on to the real file system device.

However, this approach does not account for I/Os that bypass the file system interface (e.g., raw I/O, virtual memory paging, and memory-mapped I/O). Recent

results[9] show that 15 percent of reads and nearly 30 percent of writes in Windows NT** workloads can be attributed to paging by running programs. In addition, 85 percent of processes now memory-map files, compared with 36 percent that read files and 22 percent that write them. From a practical perspective, the approach of starting with a logical trace to evaluate physical storage systems requires that a large amount of data be collected—which introduces a disturbance to the systems being traced—and then painstakingly filtered away by simulating not only the buffer cache and prefetcher but also how the data are laid out and how the meta-data are referenced. For today's well-tuned systems, these components are complex and the details of their operation are seldom publicly available. For instance, the file system buffer on many systems (e.g., Windows NT) is integrated with the memory manager and dynamically sized, based on perceived workload characteristics. Therefore, the net result of taking a logical trace and filtering it through models of the file system components is not likely to reflect the workload seen by any real storage system. Since file systems today are relatively stable and rarely undergo radical changes, we believe that for the purpose of studying physical storage systems, analyzing traces collected at the physical level is generally more practical and more realistic. This is the method we use in this paper.

In order to make our characterization more useful for subsequent modeling and analysis by others, we fitted our data to various functional forms through nonlinear regression, which we solved by using the Levenberg-Marquardt method. When appropriate, we also fitted standard probability distributions to our data by using the method of maximum likelihood, in order to obtain parameter estimates, and then optimizing these estimates with the Levenberg-Marquardt algorithm.

Trace collection. The traces analyzed in this study were collected from both server and PC systems running real user workloads on three different platforms: Windows NT, IBM AIX*, and Hewlett-Packard HP-UX**. A different trace facility was used on each platform. The Windows NT traces were collected by using VTrace,[17] a software tracing tool for Intel x86 PCs running Windows NT and Windows 2000. VTrace was primarily developed to collect data for energy management studies for portable computers. In this study, we focus mainly on the disk activities, which are collected by VTrace through the use of device filters. We have verified the disk activity collected by VTrace with the raw traffic observed by a bus (SCSI—Small Computer System Interface) analyzer.

After VTrace is installed on a system, each disk request generates a trace record consisting of the time (based on the Intel Pentium** cycle counter), sequence number, file object pointer, disk and partition numbers, start address, transfer size, and flags describing the request (e.g., read, write, synchronous). After the disk request has been serviced, a completion record is written. In a post-processing step, we match up the sequence number recorded in the request and completion records to obtain the service times.

To better understand the I/O behavior of the system, it is useful to be able to associate each disk request with the name of the corresponding file and process. Because VTrace also collects data on file system activities, in most cases we are able to match up the file object pointer with a file open record and thus obtain the filename. When the match fails, we try to determine the filename by looking up the block address in a reverse allocation map that is constructed from the daily snapshots that VTrace takes of the Windows NT file system meta-data. Since VTrace was designed to collect data for energy management studies, it also gathers data about process and thread creations and deletions as well as thread switches. By using the thread create and thread switch trace records, we are able to match I/O requests with the names of the requesting processes. In addition, the thread switch records enable us to determine the overall significance of I/O in these workloads (we look at this in the next section under "Overall significance of I/O").

To keep the amount of data collected manageable, the collection of process and thread trace records is limited to a span of one-and-a-half hours, every three-and-a-half hours. In addition, all trace collection is turned off ten minutes after the cessation of user mouse and keyboard activity. Newer versions of VTrace collect some trace data all the time, but in order to have a consistent set of data we have processed the traces used in this study and deleted trace records that occur more than ten minutes after the last user keyboard or mouse activity. In other words, we consider the system to be idle starting ten minutes after the last user action and lasting until the next user action, and we assume that there is no I/O activity during the idle periods. This means that the traces contain only the I/Os that occur when the user is actively interacting with the system, and which are

Table 1 Trace description

	Design-nation	Primary User/Use	System Configuration: System	Memory (MB)	File Systems	Storage Used (GB)	# Disks	Trace Characteristics: Duration	Foot-print (GB)	Traffic (GB)	Re-quests (10^6)
Personal Computer Workloads	P1	Engineer	333MHz P6	64	1GB FAT 5GB NTFS	6	1	45 days (7/26/99–9/8/99)	0.945	17.1	1.88
	P2	Engineer	200MHz P6	64	1.2, 2.4, 1.2GB FAT	4.8	2	39 days (7/26/99–9/2/99)	0.509	9.45	1.15
	P3	Engineer	450MHz P6	128	4, 2GB NTFS	6	1	45 days (7/26/99–9/8/99)	0.708	5.01	0.679
	P4	Engineer	450MHz P6	128	3, 3GB NTFS	6	1	29 days (7/27/99–8/24/99)	4.72	26.6	2.56
	P5	Engineer	450MHz P6	128	3.9, 2.1GB NTFS	6	1	45 days (7/26/99–9/8/99)	2.66	31.5	4.04
	P6	Manager	166MHz P6	128	3, 2GB NTFS	5	2	45 days (7/23/99–9/5/99)	0.513	2.43	0.324
	P7	Engineer	266MHz P6	192	4GB NTFS	4	1	45 days (7/26/99–9/8/99)	1.84	20.1	2.27
	P8	Secretary	300MHz P5	64	1, 3GB NTFS	4	1	45 days (7/27/99–9/9/99)	0.519	9.52	1.15
	P9	Engineer	166MHz P5	80	1.5, 1.5GB NTFS	3	2	32 days (7/23/99–8/23/99)	0.848	9.93	1.42
	P10	CTO	266MHz P6	96	4.2GB NTFS	4.2	1	45 days (1/20/00–3/4/00)	2.58	16.3	1.75
	P11	Director	350MHz P6	64	2, 2GB NTFS	4	1	45 days (8/25/99–10/8/99)	0.73	11.4	1.58
	P12	Director	400MHz P6	128	2, 4GB NTFS	6	1	45 days (9/10/99–10/24/99)	1.36	6.2	0.514
	P13	Grad. Student	200MHz P6	128	1, 1, 2GB NTFS	4	2	45 days (10/22/99–12/5/99)	0.442	6.62	1.13
	P14	Grad. Student	450MHz P6	128	2, 2, 2, 2GB NTFS	8	3	45 days (8/30/99–10/13/99)	3.92	22.3	2.9
	P-Avg.	—	318MHz	109	—	5.07	1.43	41.2 days	1.59	13.9	1.67
Server Workloads	FS1	File Server (NFS[iii])	HP 9000/720 (50MHz)	32	3 BSD FFS (3GB)	3	3	45 days (4/25/92–6/8/92)	1.39	63	9.78
	FS2	File Server (AFS[iii])	IBM RS/6000	—	23 AIX JFS (99.1GB)	99.1	17	8am–6pm (11/6/2000)	—	1.70	—
	TS1	Time-Sharing System	HP 9000/877 (64MHz)	96	12 BSD FFS (10.4GB)	10.4	8	45 days (4/18/92–6/1/92)	4.75	123	20
	DS1	Database Server (ERP[iii])	IBM RS/6000 R30 SMP[ii] (4X 75MHz)	768	8 AIX JFS (9GB), 3 paging (1.4GB), 30 raw database partitions (42GB)	52.4	13	7 days (8/13/96–8/19/96)	6.52	37.7	6.64
	S-Avg.	—	—	299	—	18.5	8	32.3 days	4.22	74.6	12.1

therefore likely to be noticed by the user. We believe that this is a reasonable approximation in the PC environment, although it is possible that we are ignoring some level of activity due to periodic system tasks such as daemons. This type of activity should have a negligible effect on the I/O load, although it might be important for other types of studies, such as energy usage.

Both the IBM AIX and Hewlett-Packard HP-UX traces were collected using kernel-level trace facilities built into the respective operating systems. These trace facilities are completely transparent to the user and add no noticeable processor load. The data collected for each physical I/O include timing information, disk and partition numbers, start address, transfer size, and flags describing the request. More details about the IBM AIX trace facility can be found in Reference 18. The HP-UX trace facility is described in Reference 15.

Trace description. In this study, we use traces collected on both server and PC systems. Table 1 summarizes the characteristics of the traces. The *footprint* of a trace is the cumulative size in bytes of all the data blocks referenced at least once.

The PC traces are denoted as P1, P2, . . . , P14. "P-Avg." denotes a fictitious trace with metrics that are the arithmetic mean of the corresponding values for all the PC traces. These traces were collected on Windows NT PCs over a period ranging from about a month to well over nine months. In this paper, we utilize only the first 45 days of the traces. The users of these systems include engineers, graduate students, one secretary, and several managers.

Table 2 Fraction of I/O activity that is filtered, and fraction of I/O requests that are synchronous

	Fraction of I/O Activity That Is Filtered						Fraction of Synchronous I/O Requests		
	Number of MBs			Number of Requests					
	Read	Write	Overall	Read	Write	Overall	Read	Write	Overall
P-Avg.	0.489	0.193	0.329	0.526	0.589	0.562	0.874	0.710	0.784
S-Avg.	0.412	0.390	0.393	0.416	0.612	0.539	0.845	0.462	0.624

The workload generated by the set of users in our sample is representative, we believe, of the PC workloads in many offices, and especially of those that involve research and development work. Note, however, that the traces should not be taken as typical or representative of other workloads. Despite this disclaimer, the fact that many of our results turn out to resemble results obtained previously, albeit in somewhat different environments, suggests that our findings are to a large extent generalizable.

The servers examined include two file servers, a time-sharing system, and a database server. The first workload (FS1) was taken from a file server supporting nine clients at the University of California, Berkeley. This system was primarily used for compilation and editing. It is referred to as "Snake" in Reference 15. The second workload (FS2) was taken from an Andrew File System (AFS) server at one of the major development sites of a leading computer vendor. The system was the primary server used to support a software development effort. For this system, only per-second aggregate statistics of the I/O traffic were gathered; addresses for individual I/Os were not collected. The FS2 data could therefore be used only in a limited number of analyses. Trace TS1 was gathered on a time-sharing system at an industrial research laboratory. It was mainly used for news, electronic mail, text editing, simulation, and compilation. It is referred to as "cello" in Reference 15. The database server trace (DS1) originates from an enterprise in the health insurance industry. The system traced was running an Enterprise Resource Planning (ERP) application on top of a commercial database. "S-Avg." denotes a fictitious trace with metrics that are the arithmetic mean of the corresponding values for traces FS1, TS1, and DS1.

Our traces capture the actual workloads that are presented to the storage system and are therefore likely to be sensitive to the amount of filtering by the file system cache and/or the database buffer pool. However, we believe that changing the amount of caching upstream will only affect our characterization quantitatively and that on a qualitative level the results still apply. To confirm this hypothesis, we filtered our traces through a least-recently-used (LRU) write-back cache to obtain another set of traces on which to run our analysis. Following the design of most file systems, we allow a dirty block (that is, a block that has been changed) to remain in the cache for up to 30 seconds. When a block is written back, we write out, in the same operation, all the dirty blocks that are physically contiguous, up to a maximum of 512 blocks. The size of the cache is chosen to be the size of the entire main memory in the original systems (Table 1). These filtered traces are denoted by adding an "f" to the original designation. For instance, the trace obtained by filtering P1 is denoted as P1f. We further denote the fictitious trace corresponding to the arithmetic mean of the matrics for all the filtered PC workloads by "Pf-Avg." and that for the filtered FS1, TS1, and DS1 workloads as "Sf-Avg."

In Table 2, we present the fraction of I/O activity that is satisfied by such an additional cache. On average, over 50 percent of the I/O requests are removed by the cache, which shows that the amount of caching has been significantly increased over what was in the original traced systems. Observe further that the traffic volume is reduced less significantly than the number of operations. This is because the smaller requests tend to have a higher chance of a hit, that is, finding the data in the cache. Furthermore, by delaying the writes, we are able to consolidate them into fewer but larger sequential writes. In Table 3 and Figure 1, we present the request size distribution for both the original and the filtered traces. *The average request size for the original workloads is about 7–9 KB.* The filtered traces have larger writes on average but their request size distributions track those

Figure 1 Distribution of request size

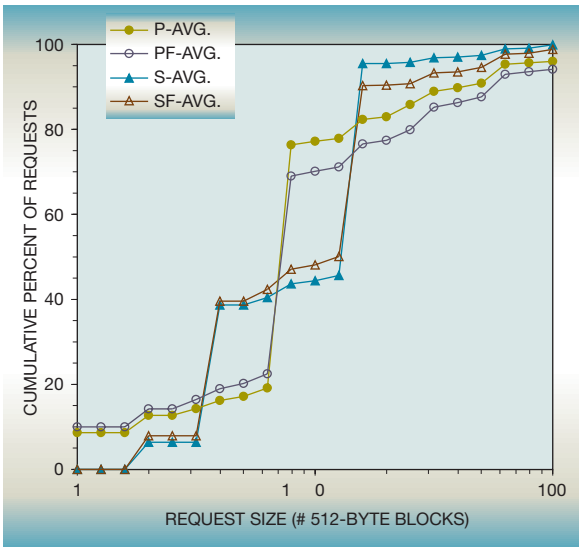


Figure 2 Disk and processor busy time

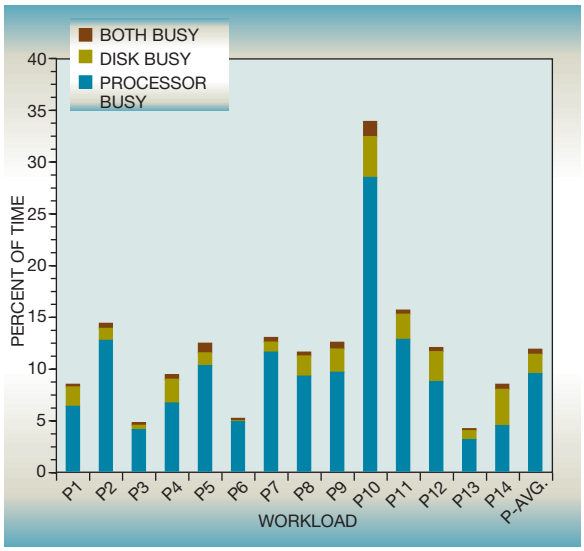


Table 3 Request size (number of 512-byte blocks)

		All requests				Read Requests				Write Requests			
		Avg.	Std. Dev.	Min.	Max.	Avg.	Std. Dev.	Min.	Max.	Avg.	Std. Dev.	Min.	Max.
Long-Run Average	P-Avg.	17.5	28.4	1	373	19.5	26.9	1	156	16.5	29.6	1	373
	Pf-Avg.	27.4	64.3	1	512	21.3	29.3	1	155	34.1	84.2	1	512
	S-Avg.	12.8	11.6	1.67	512	14.1	13.0	1.67	267	12.0	10.2	1.67	427
	Sf-Avg.	16.4	29.8	1.67	512	14.0	13.5	1.67	222	18.9	41.0	1.67	512
Busiest One-Hour Period	P-Avg.	28.6	42.3	1	256	31.5	37.7	1	156	31.8	44.3	1	256
	Pf-Avg.	55.5	93.3	1	512	34.2	38.2	1	155	91	141	1	512
	S-Avg.	15.5	16.9	1.67	219	16.5	16.6	1.67	63.3	16.1	18.9	1.67	219
	Sf-Avg.	25.8	12.7	2.00	213	27	8.73	2.00	51.3	13.1	13.4	2.67	213

of the original traces remarkably well. That the filtered traces maintain the qualitative behavior of the original traces is a result that we will see repeated for different characteristics in the rest of the paper.

Intensity of I/O

We begin our characterization by focusing on the I/O intensity of the various workloads. This is akin to understanding the size of a problem so that we can better approach it. The questions we seek to address in this section include how significant the I/O component is in the overall workload, how many I/Os are generated, and how fast the requests arrive.

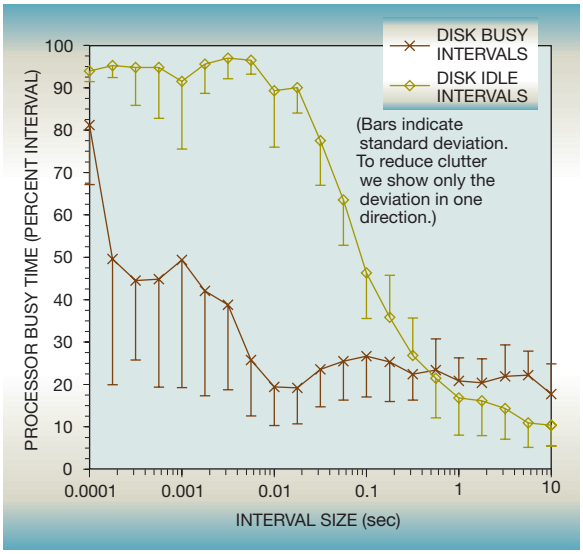
Overall significance of I/O. In Figure 2, we show the percentage of time the disk and processor are busy for the PC workloads. Similar results for the server workloads would be interesting but, unfortunately, this analysis relies on information that is available only in the PC traces. Specifically, we calculate the processor busy time by looking at the thread switch records to determine when the processor is not in the idle loop. The disk busy time is taken to be the duration during which one or more of the disks in the system are servicing requests. Recall that for the PC workloads, we only have trace data for the periods during which user input activity occurs at least once every ten minutes. The results in Figure 2 there-

fore cover only the periods during which the user is actively interacting with the system.

From the figure, the processor is, on average, busy for only about 10 percent of the time, while the disk is busy for only about 2.5 percent of the time. We cannot, however, conclude that the processor and I/O system are "fast enough." CPU idle generally represents user think time (the interval from the time results are displayed to the time the user completes entry of the next command) and occurs in any case in a single user environment. What the results do suggest is that *there is substantial idle time that can be used for performing background tasks, even when the user is active.* The challenge is to harness these idle resources without affecting the foreground work. If this can be done unobtrusively, it will pave the way for sharing idle resources in collaborative computing, a paradigm commonly referred to as peer-to-peer (P2P) computing. In addition, the idle resources can be used to optimize the system so that it will perform better in the future for the foreground task (see, for example, Reference 4). We characterize the disk idle periods in detail in the section "The relative lulls."

I/O is known to be a major component of server workloads (see, for example, Reference 19). But if processor performance continues to increase at the historical rate of 60 percent per year,[1] the results in Figure 2 suggest that *I/O may also become the dominant component of personal computer workloads in the next few years*. More memory will, of course, be available in the future for caching, but the PC systems in our study are already well-endowed with memory. A common way of hiding I/O latency is to overlap it with some computation, either through multiprogramming or by performing I/O asynchronously. From Figure 2, this technique appears to be relatively ineffective for the PC workloads since *only a small fraction (20 percent on average) of the disk busy time is overlapped with computation*. In Figure 3, we compare the processor busy time during the disk idle intervals with that during the disk busy intervals. A disk idle interval refers to the time interval during which all the disks are idle. A disk busy interval is simply the period of time between two consecutive disk idle intervals. Reflecting the low average processor utilization of the workloads, the processor is busy less than 20 percent of the time for the long intervals ($>0.1s$), regardless of whether any of the disks are busy. During the short intervals ($<0.1s$), the processor is busy almost all the time when all the disks are idle, but the processor utilization drops to less than 50 percent when one or more of the disks are busy. Such results imply that *little processing can be overlapped with I/O and that I/O response time is important for these kinds of workloads*.

Figure 3 Processor busy time during disk busy/idle intervals



That only a small amount of processing is overlapped with I/O suggests that there is effectively little multiprocessing in the PC workloads. Furthermore, as shown in Table 2, *I/Os, especially those in the PC workloads, tend to be synchronous*. This means that the system generally has to wait for I/Os to be completed before it can continue with subsequent processing. Observe from the table that although Windows NT provides a common convenient interface for performing both synchronous and asynchronous I/O, on average nearly 80 percent of the I/O requests are flagged as synchronous. Meta-data updates account for most, but not all, of the synchronous writes. Excluding meta-data writes, about half of the writes are synchronous. In traces FS1 and TS1, some I/O requests are not explicitly flagged as synchronous or asynchronous. For these traces, we assume that I/Os are synchronous unless they are explicitly flagged otherwise. Trace DS1 does not tell us whether the I/Os are synchronous.

A common difficulty in using trace-driven simulations to study I/O systems is accounting for events that occur faster or slower in the simulated system than in

Figure 4 Intervals between issuance of I/O requests and most recent request completion

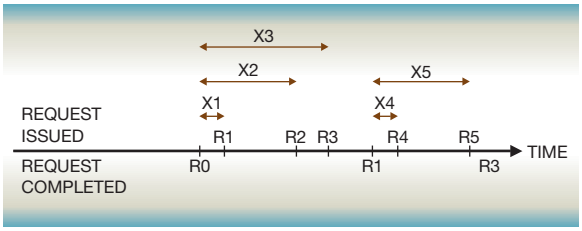
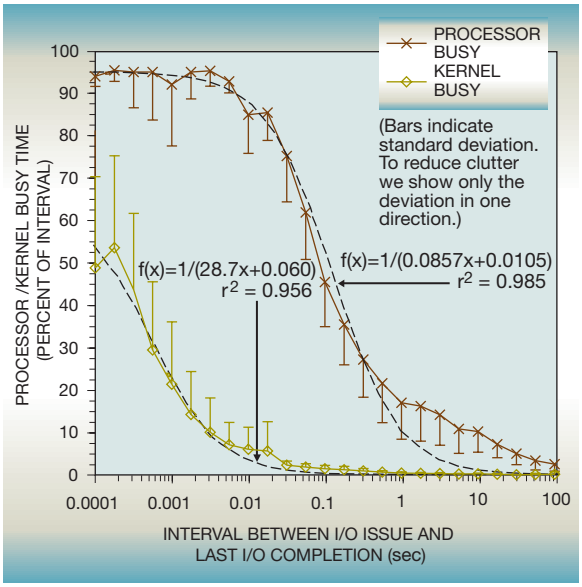


Figure 5 Processor/kernel busy time during intervals between issuance of I/Os and the most recent request completion



the original system. Since the PC workloads have little multiprocessing and most of the I/Os are synchronous, these workloads can be modeled by assuming that after completing an I/O, the system has to do some processing, and the user some "thinking," before the next set of I/Os is issued. For instance, in the timeline in Figure 4, after request R0 is completed, there are delays during which the system is processing and the user is thinking before requests R1, R2, and R3 are issued. Because R1, R2, and R3 are issued after R0 has been completed, we consider them to be dependent on R0. Similarly, R4 and R5 are deemed to be dependent on R1. Presumably, if R0 was completed earlier, R1, R2, and R3 would have been likewise issued earlier. If this in turn causes R1 to be finished earlier, R4 and R5 would similarly move earlier in time.

In Figure 5, we plot the percentage of time the processor is busy during the interval between the time an I/O request is issued and the most recent completion of an I/O request (the Xs in Figure 4). We are interested in the processor busy time during such intervals in order to model what happens when the processing time is reduced through faster processors. From Figure 5, we find that for the PC workloads, the processor utilization during the intervals between an I/O issuance and the last I/O completion is related to the length of the interval by a reciprocal function of the form $f(x) = 1/(ax + b)$ where $a = 0.0857$ and $b = 0.0105$. The reciprocal function suggests that there is a fixed amount of processing per I/O. To model a processor that is n times faster than the processor in the traced system, we would scale down the processing time by n, leaving the user think time unchanged. Specifically, we would replace an interval of length x by one of length $x[1 - f(x) + f(x)/n]$. We believe that for the PC workloads, this is considerably more realistic than simply scaling the interarrival time between I/O requests by n, as is commonly done.

In Figure 5, we also plot the percentage of time the kernel is busy during the intervals between the issue of an I/O request and the most recent I/O completion. We consider the kernel to be busy if the processor is allocated to the kernel process (process ID = 2 in Windows NT). As shown in the figure, the kernel busy time is also related to the length of the interval by a reciprocal function, as we would expect when there is some fixed kernel cost per I/O.

Amdahl's factor and access density. Table 4 presents the average and maximum amount of I/O traffic generated per day by the various workloads. Note that the average is taken over the days when there is some I/O activity recorded in the traces. This means that for the PC workloads, the weekends are, for the most part, ignored. We find that the maximum daily I/O traffic is about two to four times higher than the average. The server workloads are clearly more I/O-intensive than the PC workloads, and we expect that servers today will have even higher rates of I/O activity. Nevertheless, it should still be the case that *collecting a daily trace of referenced disk blocks is quite feasible*. Such a trace could be used later for analysis and optimization (e.g., to optimize disk block placement[4]). Specifically, for the database server work-

Table 4 Daily volume of I/O activity in thousands of I/O requests and megabytes of I/O traffic

	# I/O Requests (10^3)						I/O Traffic (MB)					
	Average			Maximum			Average			Maximum		
	Read	Write	Total	Read	Write	Total	Read	Write	Total	Read	Write	Total
P-Avg.	25	37	62	82	102	183	234	295	529	973	1084	2057
Pf-Avg.	12	15	27	48	30	78	131	236	368	701	856	1557
S-Avg.	209	313	522	530	527	1056	1579	1635	3214	3266	3393	6659
Sf-Avg.	137	113	251	446	162	609	1161	1090	2250	2731	2403	5134

Table 5 Intensity of I/O during the busiest one-hour period

	Avg. Number of Mb/s of I/O				Avg. Number of I/Os			
	Per Second	/sec /MHz	/sec /MIPS	/sec /GB	Per Second	/sec /MHz	/sec /MIPS	/sec /GB
P-Avg.	2.37	0.00697	0.00697	0.457	20.5	0.0632	0.0632	4.04
Pf-Avg.	1.92	0.00569	0.00569	0.372	9.24	0.0312	0.0312	1.94
S-Avg.	3.12	0.0255	0.0511	0.242	46.1	0.462	0.925	4.69
Sf-Avg.	2.98	0.0234	0.0467	0.217	29.5	0.375	0.750	3.99

load, logging eight bytes of information per request will create just over 12 MB of data on the busiest day.

When designing the IBM System/360*, Amdahl observed that the amount of I/O generated per instruction tends to be relatively constant.[20] More specifically, Amdahl's rule of thumb states that a typical data processing system requires approximately 1 Mb/s of I/O bandwidth for every million instructions per second (MIPS) of processing power. This rule of thumb dates back to the 1960s before buffering and caching techniques were widely used. It was recently revalidated for the logical I/O of database workloads in the production environments of some of the world's largest corporations.[10] Due to the advent of caching, however, Amdahl's factor for the ratio of physical I/O bandwidth to MIPS was found to be on the order of 0.05.[10] Because the value of Amdahl's factor determines what constitutes a balanced system, it would be useful to determine if the same value applies to the current set of workloads.

To this end, we calculated the ratio between the I/O intensity (the rate of I/O activity) and the processor speed for our workloads. Unlike the traces used in Reference 10, which cover only the peak periods of the workloads as identified by the system administrator, the traces in the current study span periods of days and weeks and include periods of light activity as well as those of heavy activity. Therefore, in calculating the I/O intensity normalized by processor speed in Table 5, we consider the busiest one-hour interval, which we define as the one-hour interval with the highest I/O bandwidth requirement. The I/O intensity averaged over various time intervals ranging from 100 milliseconds to the trace length is presented in Reference 6. Notice that according to Table 5, the filtered traces have significantly fewer I/O operations during the busiest one-hour interval. However, because the request sizes for the filtered traces are much larger during this period (see Table 3), the bandwidth figures for the filtered traces are just slightly lower than those for the original workloads. In this section, our focus is on establishing general rules of thumb with regard to the I/O intensity of our various workloads. It turns out that the effect of filtering the workloads is not large enough to significantly affect any of our findings.

From Table 5, the server workloads are fairly consistent, generating about 0.02–0.03 Mb/s of I/O for every MHz of processing power. The PC workloads are less I/O intensive, generating about 0.007 Mb/s/MHz on average. In order to determine an order of magnitude figure for the ratio of I/O bandwidth

Table 6 Projected processing power and storage needed to drive various types of I/O interconnect to 50 percent utilization

		Ethernet	Fast Ethernet	Gigabit Ethernet	Ultra ATA-100	Serial ATA	UltraSCSI 320	Fibre Channel	InfiniBand
Bandwidth (Mb/s)		10	100	1000	800	1200	2560	1000	2500
Processing Power (GHz)	P-Avg.	0.718	7.18	71.8	57.4	86.1	184	71.8	179
	Pf-Avg.	0.879	8.79	87.9	70.3	105	225	87.9	220
	S-Avg.	0.196	1.96	19.6	15.7	23.5	50.1	19.6	49.0
	Sf-Avg.	0.214	2.14	21.4	17.1	25.7	54.8	21.4	53.5
Storage (GB)	P-Avg.	10.9	109	1093	875	1312	2799	1093	2733
	Pft-Avg.	13.4	134	1344	1075	1613	3441	1344	3360
	S-Avg.	20.6	206	2063	1650	2475	5281	2063	5157
	Sf-Avg.	23.0	230	2302	1842	2763	5894	2302	5756

to MIPS, we need a rough estimate of the cycles per instruction (CPI) for the various workloads. We use a value of one for the PC workloads because the CPI for the SPEC95 benchmark on the Intel Pentium Pro processor has been found to be between 0.5 and 1.5.[21] For the server workloads, we use a CPI value of two in view of results in References 22 and 23. Based on these estimates of the CPI, we find that the *server workloads generate around 0.05 bits of real I/O per instruction*, which is consistent with the estimated Amdahl's factor for the production database workloads in Reference 10. *The figure for the PC workloads is seven times lower at about 0.007 bits of I/O per instruction*.

Interestingly, surveys conducted between 1980 and 1993 of large data processing mainframe installations found that the number of physical I/Os per second per MIPS was decreasing by just over 10 percent per year to 9.0 in 1993.[24] This figure is about ten times higher than what we are seeing for our server workloads. A possible explanation for this large discrepancy is that the mainframe workloads issue many small I/Os, but data reported in Reference 24 show that the average I/O request size for the surveyed mainframe installations was about 9 KB, which is slightly larger than the 8 KB for our server workloads (Table 3). Of course, mainframe MIPS and reduced instruction set computer (RISC) MIPS are not directly comparable, and this difference could account for some of the disparity, as could intrinsic differences between the workloads. In addition, our calculations are based on the MIPS rating of the system, which is what we have available to us. The mainframe surveys, on the other hand, used utilized MIPS[25] or the processing power actually consumed by the workload. To make our calculations consistent with the survey results, we could factor in the processor utilization when the workload is running. For instance, if the processor utilization is 10 percent, as suggested by our earlier results for the PC workloads, we would multiple our figures by 10. With this adjustment, the PC workloads still generate less than one I/O per second per MIPS. The server traces, unfortunately, do not contain information from which we can derive the processor utilization for these workloads. But we expect the processor in these workloads to be also less than fully utilized so that the number of I/Os generated per second per MIPS by these workloads is actually closer to the survey results than the raw numbers suggest.

Another useful way of looking at I/O intensity is with respect to the storage used (Table 1). In this paper, the storage used by each of the workloads is estimated to be the combined size of all the file systems and logical volumes defined in that workload. This makes our calculations comparable to historical data and is a reasonable assumption unless storage can be allocated only when written to, for instance by using storage virtualization software that separates the system view of storage from the actual physical storage. Table 5 summarizes, for our various workloads, the number of I/Os per second per GB of storage used. This metric is commonly referred to as *access density* and is widely used in commercial data processing environments.[24] The survey of large data processing mainframe installations cited above found the access density to be decreasing by about 10 percent per year to 2.1 I/Os per second per GB of storage in 1993. Notice from Table 5 that for both our base PC and server workloads, the access density is,

on average, about two times higher. The results for the filtered workloads are substantially lower, as one would expect. That the access density of our PC workloads is similar to the figure for our server workloads even though these server workloads are several years older suggests that *PC workloads may be comparable to server workloads in terms of access density*. Note, however, that *as disks become a lot bigger and PCs have at least one disk, the density of access with respect to the available storage is likely to be much lower for PC workloads*.

Table 5 also contains results for the number of bits of I/O per second per GB of storage used. The PC workloads have, on average, 0.46 Mb/s of I/O per GB of storage. By this measure, the server workloads are less I/O-intense with an average of only 0.24 Mb/s of I/O per GB of storage. Based on these results, we project the amount of processing power and storage space that will be needed to drive various types of I/O interconnect to 50 percent utilization. The results are summarized in Table 6. Note that all the modern I/O interconnects offer Gb/s bandwidth. Some of them, specifically Ethernet and Fibre Channel, have newer versions with even higher data rates. For the kinds of workloads we analyze here, the I/O interconnect is not expected to be a bottleneck any time soon. However, we would expect to see much higher bandwidth requirements for workloads that are dominated by large sequential I/Os (e.g., scientific and decision support workloads). In such environments, and especially when many workloads are consolidated into a large server and many disks are consolidated into a sizeable outboard controller, the bandwidth requirements have to be carefully evaluated to ensure that the network or connection between the disks and the host does not become the bottleneck.

Request arrival rate. In Figure 6, we present the distribution of I/O interarrival time. Since this distribution is often needed in modeling I/O systems, we fitted standard probability distributions to it. As shown in the figure, the commonly used exponential distribution, while easy to work with mathematically, turns out to be a rather poor fit for all the workloads. Instead, *the lognormal distribution (denoted Lognorm [μ, σ] where μ and σ are, respectively, the mean and standard deviation) is a reasonably good fit*. Recall that a random variable is lognormally distributed if the logarithm of the random variable is normally distributed. Therefore, the lognormal distribution is skewed to the right or toward the larger values, meaning that there exist long intervals with no I/O arrivals. The

Figure 6 Distribution of I/O interarrival time

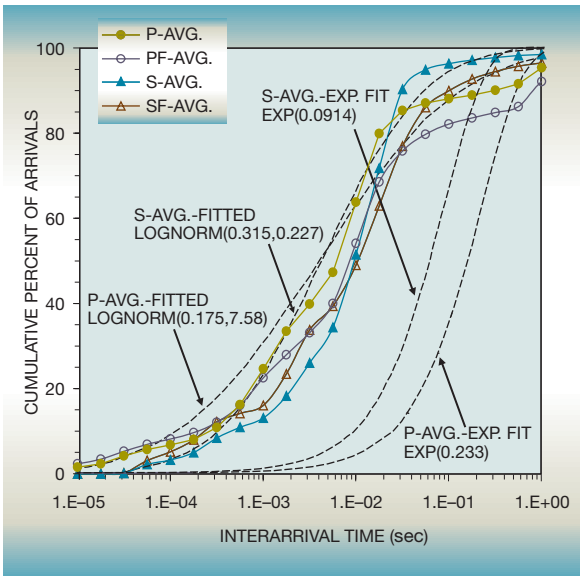
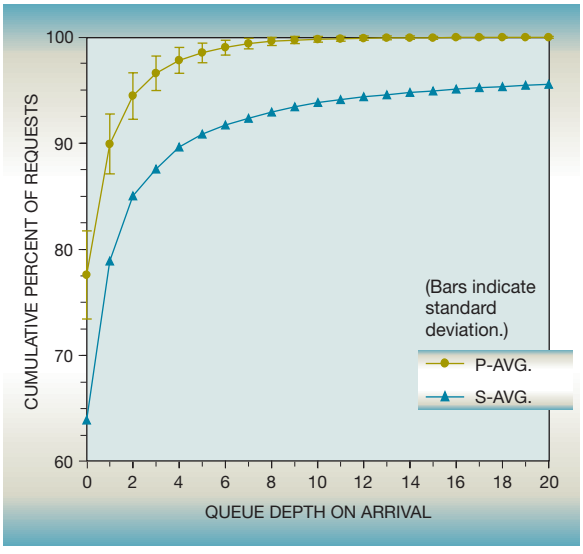


Figure 7 Distribution of queue depth on arrival



long tail of the interarrival distribution could be a manifestation of different underlying behavior such as correlated arrival times, but regardless of the cause, the net effect is that *I/O requests seldom occur singly but tend to arrive in groups*, because if there are long intervals with no arrivals, there must be inter-

Table 7 Queue depth on arrival

	# I/Os Outstanding					# Reads Outstanding					# Writes Outstanding				
	Avg.	Avg.|>0	Std. Dev.	90%-tile	Max.	Avg.	Avg.|>0	Std. Dev.	90%-tile	Max.	Avg.	Avg.|>0	Std. Dev.	90%-tile	Max.
P-Avg.	0.489	2.14	1.34	1.64	34.3	0.216	1.91	0.797	0.643	20.2	0.273	1.94	0.975	0.786	28.6
S-Avg.	4.87	13.4	20.5	6.67	656	0.201	1.58	0.672	1	14	4.66	16.7	20.5	6.33	656

vals that have far more arrivals than their even share. We will analyze the burstiness of the I/O traffic in greater detail in the next section.

Another interesting way to analyze the arrival process of I/O requests is relative to the completion of preceding requests. In particular, if the workload supports multiple outstanding I/O requests, there will be more potential for improving the average I/O performance, for instance, through request scheduling. Figure 7 presents the distribution of queue depth, which we define to be the length of the request queue as seen by an arriving request. In Table 7, we break down the outstanding requests into reads and writes. Note that we consider a request to be in the queue while it is being serviced.

We find that across all the workloads, the read queue tends to be shallow—more than 85 percent of the requests arrive to find the queue devoid of read requests, and the average number of reads outstanding is only about 0.2. Nevertheless, the read queue can be deep at times. If there are read requests in the queue, the average number of them is almost 2 (denoted Avg.$|>0$ in Table 7). In addition, the maximum read queue depth can be more than 90 times higher than the average. Notice that *the server workloads do not appear to have a deeper read queue than the personal system workloads. This finding suggests that read performance in personal system workloads could benefit as much from request scheduling as in server workloads*. We examine request scheduling in detail in Reference 5. Observe further from Table 7 that *the write queue is markedly deeper than the read queue for all the workloads*, as we would expect given that a greater fraction of writes are asynchronous compared to reads (Table 2). The PC workloads appear to have a significantly shallower write queue than the server workloads.

Note that we are looking at the number of outstanding requests from the perspective of the operating system layer at which the trace data were collected. This reflects the potential for request scheduling at any of the levels below, and not just at the physical storage system. Some of the differences among the workloads could be the result of collecting the traces at different levels on the different platforms.

Variability in I/O traffic over time

When I/O traffic is smooth and uniform over time, system resources can be very efficiently utilized. However, when the I/O traffic is bursty as is the case in practice (cf. "Request arrival rate" above), resources have to be provisioned to handle the bursts so that during the periods when the system is relatively idle, these resources will be wasted. There are several approaches to try to even out the load. The first is to aggregate multiple workloads in the hope that the peak and idle periods in the different workloads cancel one another out. This idea is one of the premises of the storage utilities model. Whether the aggregation of multiple workloads achieves the desired effect of smoothening the load depends on whether the workloads are dependent or correlated. We examine below the dependence among our workloads.

The second approach to smoothening the traffic is to try to shift the load temporally, for instance, by deferring or offloading some work from the busy periods to the relative lulls (e.g., write buffering and logging disk arrays[26]) or by eagerly or speculatively performing some work in the hope that such work will help improve performance during the next busy period (e.g., prefetching and reorganizing data based on access patterns[4]). The effectiveness of these attempts at time-shifting the load to even out the traffic depends on the extent to which the traffic is autocorrelated. We analyze the autocorrelation of I/O traffic in order to determine whether it is long-range dependent or self-similar in the section "Self-similarity in I/O traffic." In the section "The relative lulls," we characterize in detail the idle periods to help in

the design of techniques that try to exploit idle resources.

Dependence among workloads. In general, two processes are said to be dependent or correlated if the value a process takes on constrains the possible values that the other process can assume. In the current context, the process is the discretized time series of the I/O traffic generated by a given workload. If two workloads are positively correlated, the peaks in the corresponding processes of the two workloads occur at the same time so that if the two workloads are aggregated, the resulting workload will have higher peaks. If the workloads are negatively correlated, the peaks of one will occur when the other workload is relatively idle. If the workloads are independent, there is no correlation between the volumes of activity of the two workloads. When many independent workloads are aggregated, the resulting traffic will tend to be smooth.

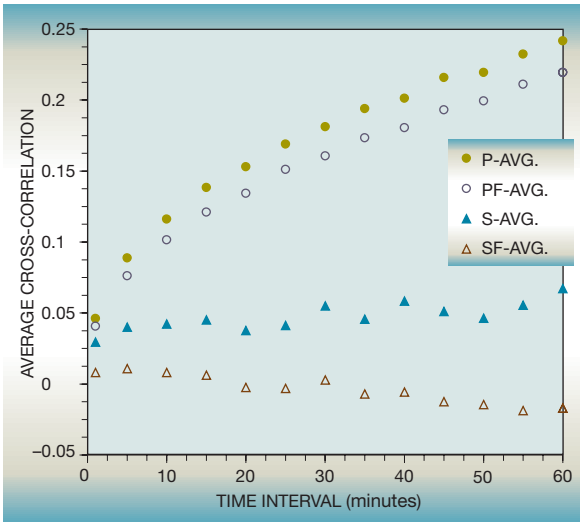
To more formally characterize the dependence among the workloads, we calculate the cross-correlation. The cross correlation between two processes $P(i)$ and $Q(i)$, where $i = 0, 1, 2 \ldots n - 1$, is defined as

$$r_{PQ} = \frac{\sum_i (P(i) - \bar{P})(Q(i) - \bar{Q})}{\sqrt{\sum_i (P(i) - \bar{P})^2}\ \sqrt{\sum_i (Q(i) - \bar{Q})^2}} \qquad (1)$$

The possible values of r_{PQ} range from -1 to 1, with -1 indicating perfect negative correlation between the two processes, 0 indicating no correlation, and 1 indicating perfect positive correlation. For each workload, we consider the I/O arrival process aggregated over fixed intervals that range from one minute to a day. We synchronize the processes by the time of day and the day of week. The results are available in Reference 6.

To summarize the dependence among a set of workloads W, we introduce the average cross-correlation which is defined as $\overline{r_{PQ}}$ where $P \in W$, $Q \in W$ and $P \neq Q$. Figure 8 plots the average cross-correlation for the PC workloads as a function of the time interval used to aggregate the arrival process. The same figure also plots the average cross-correlation among the server workloads. We find that, in general, *there is little cross-correlation among the server workloads, suggesting that aggregating them will likely help to smooth out the traffic and enable more efficient utilization of resources*. Our PC workloads are taken mostly from office environments with flexible work-

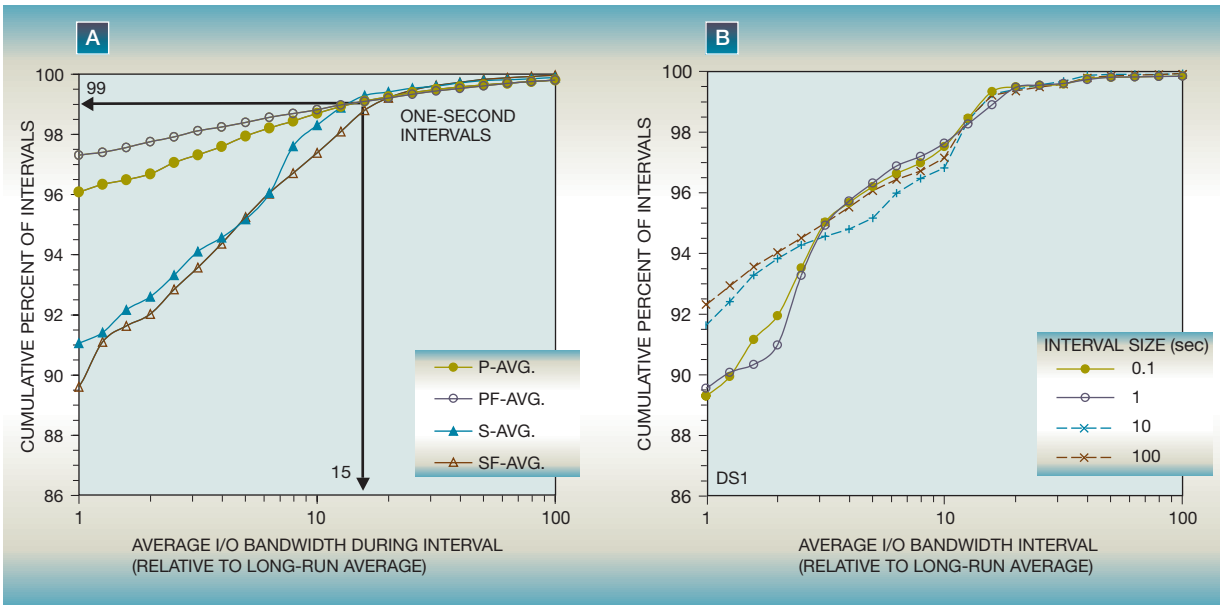
Figure 8 Cross-correlation of volume of I/O activity vs time interval used to aggregate volume



ing hours. Nevertheless, *the cross-correlation among the PC workloads is still significant except at small time intervals. This suggests that multiplexing the PC workloads will smooth out the high frequency fluctuations in I/O traffic, but some of the time-of-day effects will remain unless the PCs are geographically distributed in different time zones*. Note that the filtered workloads tend to be less correlated but the difference is small.

Self-similarity in I/O traffic. In many situations, especially when outsourcing storage, we need rules of thumb to estimate the I/O bandwidth requirement of a workload without having to analyze the workload in detail. We previously computed the access density and found that the server workloads average about five I/Os or about 30 KB worth of I/O per second per GB of data. This result can be used to provide a baseline estimate for the I/O bandwidth required by a workload given the amount of storage it uses. To account for the variability in the I/O traffic, Figure 9A plots the distribution of I/O traffic averaged over one-second intervals and normalized to the average bandwidth over the entire trace. The figure shows that *to satisfy the bandwidth requirement for 99 percent of the one-second intervals, we would need to provision for about 15 times the long-run average bandwidth*. Notice that for all the workloads, there is an abrupt knee in the plots just beyond 99 percent of the intervals. This means that *to satisfy requirements beyond 99 percent of the time would require disproportionately more resources*.

Figure 9 Distribution of I/O traffic averaged over various time intervals



In analyzing the data, we notice that for many of the workloads, the distribution of I/O traffic is relatively insensitive to the size of the interval over which the traffic is averaged. For instance, in Figure 9B, the distributions for time intervals of 0.1s, 1s, 10s, 100s for the database server DS1 are very similar. In Reference 6, we plot the discretized time series of the traffic for TS1 and DS1 and find that the plots look similar for time scales ranging from tens of milliseconds to tens of seconds. In other words, rescaling the time series does not remove the burstiness.

Definition of self-similarity. The phenomenon whereby a certain property of an object is preserved with respect to scaling in space and/or time is described by self-similarity and fractals.[27] Let X be the incremental process of a process Y, that is, $X(i) = Y(i + 1) - Y(i)$. In this context, Y counts the number of I/O arrivals and $X(i)$ is the number of I/O arrivals during the ith time interval. Y is said to be self-similar with parameter H if for all integers m,

$$X = m^{1-H}X^{(m)} \tag{2}$$

where

$$X^{(m)}(k) = (1/m) \sum_{i=(k-1)m+1}^{km} X(i), \qquad k = 1, 2, \ldots$$

is the aggregated sequence obtained by dividing the original series into blocks of size m and averaging over each block, and k is an index over the sequence of blocks. In this paper, we focus on second-order self-similarity, which means that $m^{1-H}X^{(m)}$ has the same variance and autocorrelation as X.

The single parameter H expresses the degree of self-similarity and is known as the Hurst parameter. For smooth Poisson traffic, the H value is 0.5. For self-similar series, $0.5 < H < 1$, and as $H \to 1$, the degree of self-similarity increases. Mathematically, self-similarity is manifested in several equivalent ways and different methods that examine specific indications of self-similarity are used to estimate the Hurst parameter. Many of the statistical methods used to estimate the Hurst parameter assume that the arrival process is stationary. In order to avoid an arrival process that was potentially nonstationary, we selected two one-hour periods from each trace. The first period is chosen to be a high-traffic period, specifically one that contains more I/O traffic than 95 percent of other one-hour periods in the trace. The second period is meant to reflect a low traffic situation and is chosen to be one that contains more I/O traffic than 30 percent of other one-hour periods in the trace.

The interested reader is referred to Reference 6 for details about how we estimate the degree of self-similarity for our various workloads. Here, we simply summarize the Hurst parameter values we obtain (Table 8) and state the finding that *for time scales ranging from tens of milliseconds to tens and sometimes even hundreds of seconds, the I/O traffic is well-represented by a self-similar process*. Note that filtering the workloads does not affect the self-similar nature of their I/O traffic.

Implications of self-similarity in I/O traffic. The I/O traffic being self-similar implies that burstiness exists over a wide range of time scales and that attempts at evening out the traffic temporally will tend to not remove all the variability. More specifically, the I/O system may experience concentrated periods of congestion with associated increase in queuing time. Furthermore, resource (e.g., buffer space, channel bandwidth) requirements may skyrocket at much lower levels of utilization than expected with the commonly assumed Poisson model in which arrivals are mutually independent and are separated by exponentially distributed intervals. This behavior should be considered when designing storage systems, especially when multiple workloads are to be isolated so that they can coexist peacefully in the same storage system, as is required in many storage utilities. Such burstiness should also be accounted for in the service level agreements (SLAs) when outsourcing storage.

More generally, I/O traffic has been known to be bursty but characterizing this variability has been difficult. The concept of self-similarity provides us with a succinct way to characterize the burstiness of the traffic. We recommend that *I/O traffic be characterized by a three-tuple consisting of the mean and variance of the arrival rate and some measure of the self-similarity of the traffic such as the Hurst parameter*. The first two parameters can be easily understood and measured. The third is more involved but can still be visually explained. Table 8 summarizes these parameter values for our various workloads.

It turns out that self-similar behavior is not limited to I/O traffic or to our workloads. Recently, file system activities[28] and I/O traffic[29] have been found to exhibit scale-invariant burstiness. Local and wide-area network traffic may also be more accurately modeled using statistically self-similar processes rather than the Poisson model (see for example Reference 30). However, analytical modeling with self-similar inputs has not been well developed yet. This, coupled with the complexity of storage systems today, means that the effect of self-similar I/O traffic has to be analyzed, for the most part, through simulations. The parameters in Table 8 can be used to generate self-similar traffic for such simulations. See, for example, Reference 6.

Table 8 Hurst parameter, mean, and variance (per second) of traffic arrival rate during the high-traffic period

	Characteristics of High-Traffic Period		
	H	μ(KB/s)	σ^2(KB/s)^2
P-Avg.	0.81	188	769080
Pf-Avg.	0.79	91.6	528538
S-Avg.	0.90	445	627261
Sf-Avg.	0.80	367	528439

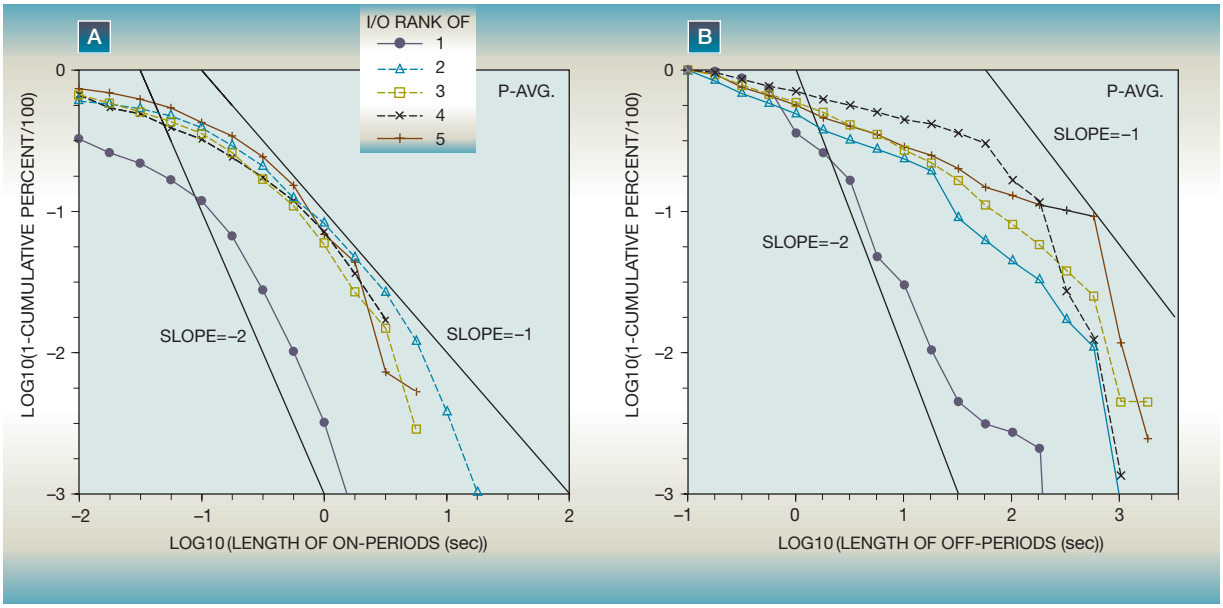
Underpinnings of self-similar I/O traffic. We have seen that the I/O traffic in our workloads is self-similar, but self-similarity is a rather abstract concept. To present a more compelling case and provide further insights into the dynamic nature of the traffic, we relate this phenomenon to some underlying physical cause, namely the superposition of I/O from multiple processes in the system where each process behaves as an independent source of I/O with on periods that are heavy-tailed.

A random variable, X, is said to follow a heavy-tailed distribution if

$$P(X > x) \sim cx^{-\alpha}, \text{ as } x \to \infty, c > 0, 1 < \alpha < 2 \quad (3)$$

Such a random variable can give rise to extremely large values with nonnegligible probability. The superposition of a large number of independent traffic sources with on and/or off periods that are heavy-tailed is known to result in traffic that is self-similar[31] (the process does not follow a Poisson distribution, and the assumptions of Palm-Khintchine theorem are not satisfied). In the current context, we consider each process in the system as an independent source of I/O. As in Reference 29, we define an off period for a process as any interval longer than 0.2s during which the process does not generate any I/O. All other intervals are considered to be on periods for the process. This analysis has been shown to be relatively

Figure 10 Length of ON periods (A) and OFF periods (B) for the five most I/O-active processes



insensitive to the threshold value used to distinguish the on and off periods.[31]

Taking the logarithm of both sides of Equation 3, we get

$$log(P(X > x)) \sim log(c) - \alpha log(x), as\ x \to \infty \quad (4)$$

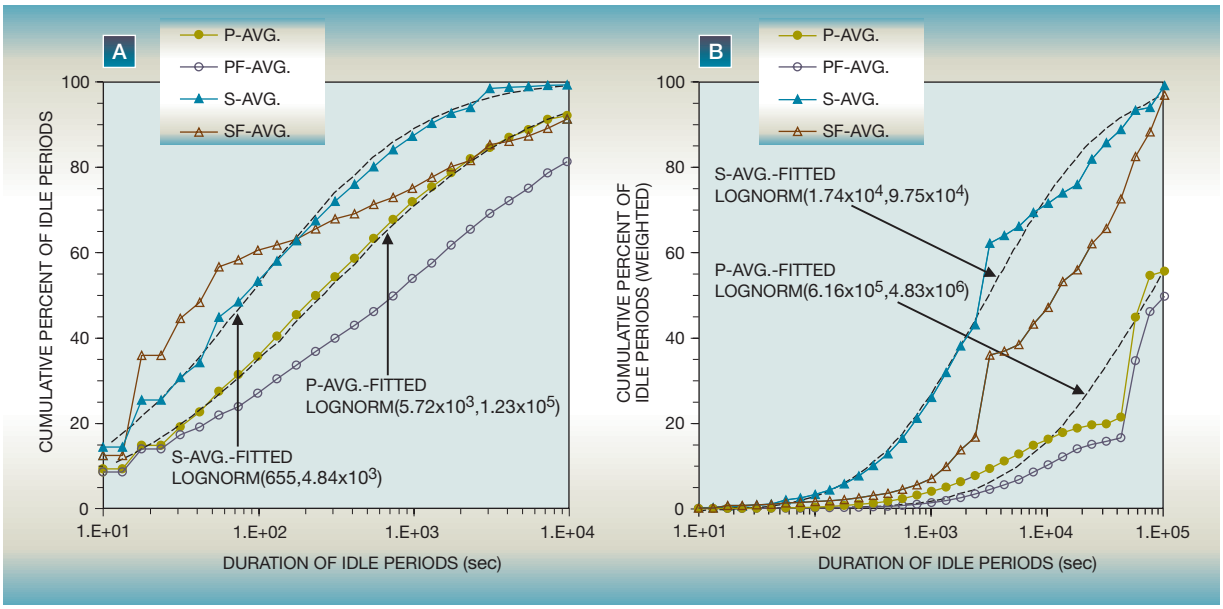
Therefore, if X is heavy-tailed, the plot of $P(X > x)$ versus x on log-log scale should yield a straight line with slope α for large values of x. Such log-log plots are known as complementary cumulative distribution plots or "qq-plots." In Figure 10, we present the qq-plots for the lengths of the on and off periods for the five processes that generate the most I/O traffic in each of our PC workloads. Unfortunately, none of our other workloads contain the process information that is needed for this analysis. As shown in the figure, the on periods appear to be heavy-tailed but not the off periods. This is consistent with results reported in Reference 29 where the lack of heavy-tailed behavior for the off periods is attributed to periodic activity such as the sync daemon traffic. Having heavy-tailed on periods is sufficient, however, to result in self-similar aggregate traffic.

The relative lulls. As discussed earlier, when the I/O load is not constant but varies over time, there may be opportunities to use the relatively idle periods to do some useful work. Here, we characterize in detail the idle periods, focusing on specific metrics that will be helpful in designing techniques that try to exploit idle time.

We consider an interval to be idle if the average number of I/Os per second during the interval is less than some value k. The term *idle period* refers to a sequence of intervals that are idle. The duration of an idle period is simply the product of the number of idle intervals it contains and the interval size. In this study, we use a relatively long interval of 10 seconds because we are interested in long idle periods during which we can perform a substantial amount of work. Note that storage systems tend to have some periodic background activity so that treating an interval to be idle only if it contains absolutely no I/O activity would be far too conservative. Since disks today are capable of supporting in excess of 100 I/Os per second, we select k to be 20 for all our workloads except DS1. DS1 contains several times the allocated storage in the other workloads so its storage system will presumably be much more powerful. Therefore, we use a k value of 40 for DS1.

Based on this definition of an idle interval, we find that for the PC workloads, more than 99 percent of

Figure 11 Distribution of idle periods (A) and idle time (B)



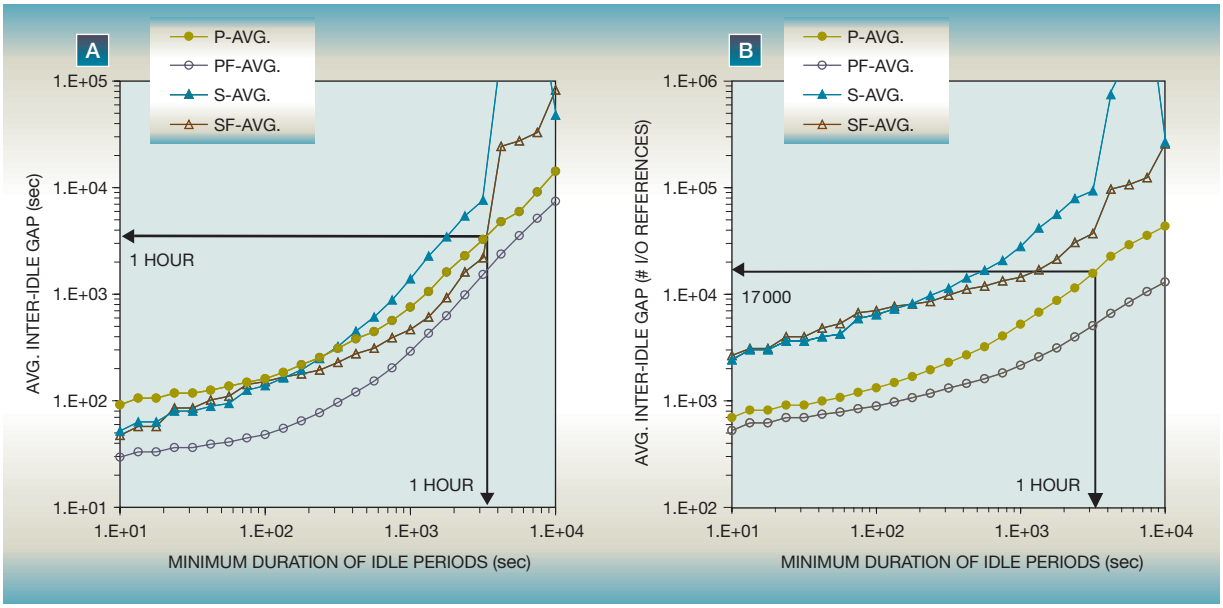
the intervals are idle. The corresponding figure for the server workloads on average is more than 93 percent. Such results clearly indicate that *there are resources in the storage system that are significantly underutilized and that can be put to good use*. Figure 11A presents the distribution of idle periods for our workloads, while Figure 11B shows the corresponding distribution of idle time. Figure 11B is obtained by weighing the idle periods by their duration. We fitted standard probability distributions to the data and found that the lognormal distribution is a reasonably good fit for most of the workloads. Notice that although most of the idle periods are short (less than a thousand seconds), long idle periods account for most of the idle time. This is consistent with previous results[32] and implies that *a system that exploits idle time can get most of the potential benefit by simply focusing on the long idle periods*.

Inter-idle gap. An important consideration in utilizing idle resources is the frequency with which suitably long idle periods can be expected. In addition, the amount of activity that occurs between such long idle periods determines the effectiveness and the feasibility of exploiting the idle periods. For instance, a log-structured file system[33] where garbage collection is performed periodically during system idle time may run out of free space if there is a lot of write activity between the idle periods. In the disk block reorganization scheme proposed in Reference 4, the inter-idle gap, that is, the time span between suitably long idle periods, determines the amount of trace data that have to be accumulated on the disk.

In Figure 12, we consider this issue by plotting the average inter-idle gap as a function of the duration of the idle period. The results show that for the PC workloads on average, idle periods lasting at least an hour are separated by busy periods of about an hour and with just over 17000 references. As we would expect, the server workloads have longer busy periods separated by shorter idle periods. But in both environments, the results indicate that *there are long idle periods that occur frequently enough to be interesting for off-line optimizations such as block reorganization*.[4] In the server environments, we might have to be more meticulous about using the idle time, for instance, by dividing an off-line task into several finer-grained steps that can be scheduled whenever there is a short idle period.

Idle length prediction. In some cases, there is a recovery cost associated with stopping an off-line task before it is completed. Therefore, it is important to predict how long an idle period will last so that the system can decide whether a task should be initiated.

Figure 12 Average duration of busy periods measured in seconds (A) and measured in number of I/O references (B)



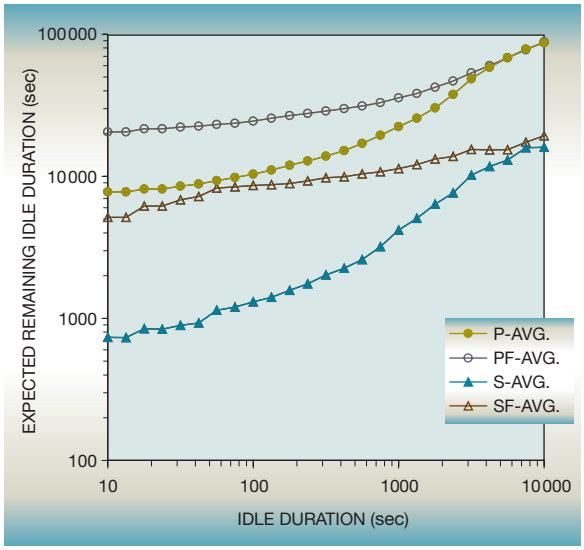
To this end, we calculated the autocorrelation of the sequence of idle period duration at different lags and discover that for all our workloads, there is little correlation between the length of one idle period and the lengths of the immediately preceding periods.[6] In other words, *how long the system will remain idle is not predictable from the lengths of its recent idle periods*. This is in contrast to the strong correlation that previously was observed for a personal UNIX workstation.[32] In that study, the level of activity at which the system was considered idle was not clear. We conjecture that because the personal UNIX workstation in the previous study was not heavily used, the idle periods are determined primarily by the periodic background activity that exists in the system, hence the strong autocorrelation.

In Figure 13, we plot the expected future idle duration, $E[I(x)]$, which is defined as the expected remaining idle duration given that the system has already been idle for x units of time. More formally,

$$E[I(x)] = \sum_{i=x+1}^{\infty} \frac{(i-x)l(i)}{1-L(i)} \qquad (5)$$

where $l(\cdot)$ is the probability distribution of the idle period duration; that is, $l(j)$ is the probability that an idle period has a duration of j, and $L(\cdot)$ is the cumulative probability distribution of the idle period duration; that is, $L(j) = \sum_{i=1}^{j} l(i)$. Observe from Figure 13 that $E[I(x)]$ is generally increasing. In other

Figure 13 Remaining idle duration



words, the longer the system has been idle, the longer it is likely to remain idle. This phenomenon suggests prediction policies that progressively raise the predicted remaining idle duration as the system remains idle. Note that the plot is logarithmic so the rate of increase in $E[I(x)]$ is higher than it appears.

To better understand how such prediction policies should be designed, we also calculated the hazard rate of the idle period duration. The hazard rate is simply the likelihood that an idle period ends with a duration of at most $k + r$ time units given that it is already k units long. In other words, given that the system has been idle for k units, $H(k, r)$ is the probability that a task initiated now and requiring r units of time will not be completed before the system becomes busy again. More formally,

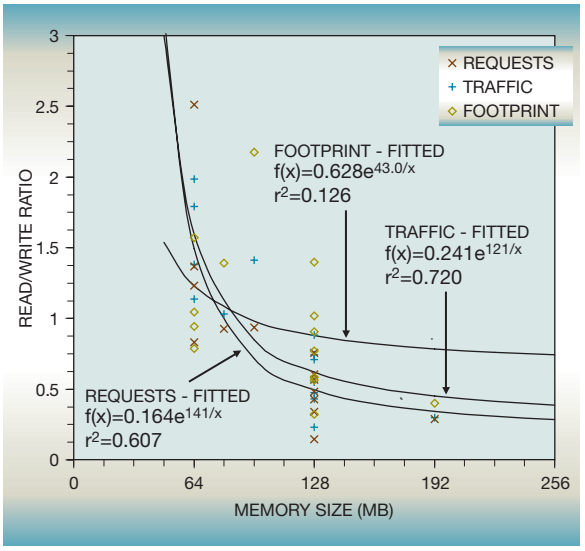
$$H(k, r) = \frac{\sum_{i=0}^{r} l(k + i)}{1 - L(k - 1)} \qquad (6)$$

We find that the hazard rate increases with r, meaning that the chances for the task not to be completed before the system becomes busy again increase with the length of the task, as we would expect. In addition, the hazard rate generally declines as the length of time the system has already been idle increases.[6] *This result supports again the idea of predicting the remaining idle period duration by conditioning on the amount of time the system has already been idle.*

Interaction of reads and writes

In general, the interaction between reads and writes complicates a computer system and throttles its performance. For instance, static data can be simply replicated to improve not only the performance of the system but also its scalability and durability. But if the data are being updated, the system has to ensure that the writes occur in the correct order. In addition, it has to either propagate the results of each write to all possible replicated copies or to invalidate these copies. The former usually makes sense if the updated data are unlikely to be updated again but are likely to be read. The latter is useful when it is highly likely that the data will be updated several more times before the data are read. In cases where the data are being both updated and read, replication may not be useful. The read-write composition of the traffic, together with the flow of data from writes to reads, is therefore an extremely important workload characteristic and the focus of this section.

Figure 14 Read/write ratio as function of memory size



Read/write ratio. A wide range of read/write ratios has been reported in the literature. In addition to intrinsic workload differences, the read/write ratio also depends to a large extent on how many of the reads and writes have been filtered by caching, and on the kinds of I/Os (e.g., user data, paging, file system meta-data) that are tabulated. Because main memory is volatile, the amount of write buffering performed by the file system cache is typically limited. For example, UNIX systems have traditionally used a policy of periodically (once every 30 seconds) flushing the dirty blocks in the file cache to disk so as to limit the amount of data that can potentially be lost in a system failure. In Windows NT, one quarter of the dirty data in the file cache is written back to disk every second.[34] Therefore, more of the reads than writes are filtered by the file system cache. The file system also adds meta-data writes, which may account for more than half of the physical writes (more than 72 percent in Reference 15 and more than 53 percent in our PC workloads). Thus at the logical level, the read/write ratio is generally much higher than at the physical level.

For instance, the ratio of logical read to write traffic has been reported to be between 3.7 and 6.3 for desk-

Figure 15 Miss ratio with LRU write-back cache (512-byte blocks)

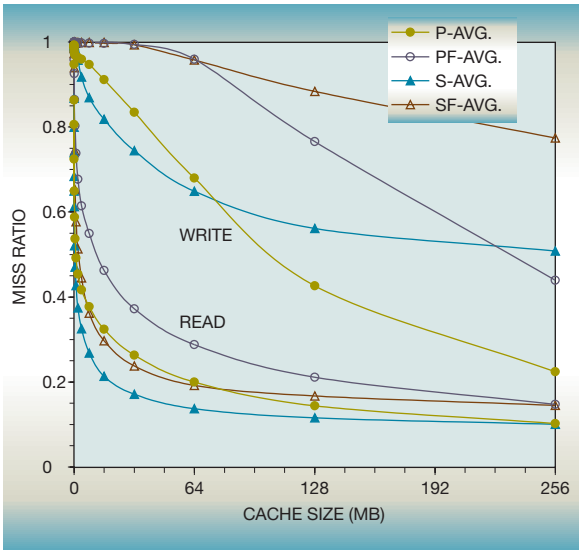


Table 9 Read/write ratio

	Read/Write Ratio		
	Requests	Traffic	Footprint
	# Read Requests / # Write Requests	MB Read / MB Written	# Unique Blocks Read / # Unique Blocks Written
P-Avg.	0.816	0.932	0.988
Pf-Avg.	0.965	0.607	0.888
S-Avg.	0.706	0.870	1.24
Sf-Avg.	1.12	0.843	1.19

top workstation workloads,[9] and the ratio of logical read to write operations has been found to be between 3 and 4.5 in various office environments.[35] But at the physical level, the read/write ratio has been observed to range from only about 0.4 to 1 for Novell NetWare file servers[16] and from about 0.7 to 0.8 for several HP-UX systems.[15] These figures are comparable to the physical read/write ratio we obtained, which are presented in Table 9. Observe that *for the server workloads and the PC workloads on average, the ratio of read to write requests ranges from 0.71 to 0.82, which means that writes account for about 60 percent of the requests*. Interestingly, mainframe data processing workloads appear to have a higher read/write ratio. For example, measurements conducted at the physical level at 12 moderate-to-large MVS* installations running mainly data processing applications (circa 1993) found the read/write ratio to be about 3.5.[24] Analysis of the logical I/O traffic of the production database workloads of ten of the world's largest corporations of about the same period found the read/write ratio to average roughly 10.[10,11]

In calculating the average read/write ratio presented in Table 9, we observe that for the PC workloads, the read/write ratio appears to be negatively correlated with the memory size of the system. Unfortunately, we do not have enough data points to observe any trends for the server workloads. In Figure 14, we plot the read/write ratio for the PC workloads as a function of the memory size. As shown in the figure, the read/write ratio is approximately related to the memory size by an exponential function of the form $f(x) = ae^{b/x}$ where a and b are constants. The model is limited by the few data points we have, but it predicts that with an infinitely large memory, that is, as $x \rightarrow \infty$, there will be about six writes for every read. Such results support the prediction that almost all reads will be absorbed by the larger buffer caches in the future so that physical I/O will become dominated by writes.[33] However, that the read/write ratio remains relatively consistent across all our workloads, which span a time period of eight years, suggests that workload changes may have a countereffect. Also, the fact that the ratio of read footprint to write footprint decreases, albeit slowly, with memory size, suggests that effects (e.g., workload differences) other than an increase in caching, could also be at work here.

If writes become increasingly dominant, a pertinent question to ponder is whether physical read performance really matters. In Figure 15, we plot the read and write cache miss ratios assuming a write-back cache with the least-recently-used (LRU) cache replacement policy. We define the miss ratio to be the fraction of requests that cannot be satisfied by the cache but that result in a request to the underlying storage system. Observe that the plots for the filtered workloads are simply a translation of those for the original workloads; the behavior is qualitatively similar. In this experiment, we are in essence simulating a second-level cache. The upstream file system cache and/or the database buffer pool have captured significant portions of any read reuse but because they are volatile, they cannot safely cache the writes. Therefore, *the writes observed at the storage level ex-*

hibit much stronger locality than the reads. In other words, although read caching by the file system or the database buffer can eliminate most of the reads, if writes are delayed long enough by using nonvolatile memory, write requests can similarly be very significantly reduced. In fact, for practically all the workloads, a small cache of 1 MB eliminates more than half the writes.

Furthermore, unlike reads, which tend to be synchronous, writes can be effectively rendered asynchronous through the use of write caching or buffering.[5] In addition, the effective latency of writes can often be reduced by writing data asynchronously or in a log[33] or by using write-ahead logging. Recent results (see for example Reference 36) further suggest that because of the widening performance gap between processor- and disk-based storage, file system read response times may be dominated by disk accesses even at very high cache hit rates. Therefore, *the performance of read I/Os continues to be very important*.

Working set overlap. The working set $W(t, \tau)$ is defined as the set of blocks referenced within the last τ units of time.[37] More formally,

$$W(t, \tau) = \{b : Count(b, t - \tau, t) >= 1\} \qquad (7)$$

where $Count(b, t_1, t_2)$ denotes the number of times block b is referenced between t_1 and t_2. In Figure 16, we plot the average and maximum daily working set size for our workloads. Note that we define the working set of day x as $W(t =$ midnight of day x, $\tau = 1$ day). To understand the interaction between reads and writes, we classify the blocks referenced into those that are read, written, and both read and written. Specifically,

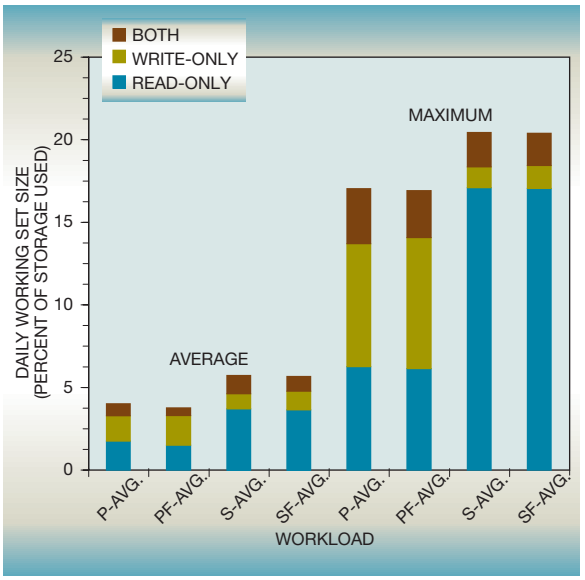
$$W_{read}(t, \tau) = \{b : ReadCount(b, t - \tau, t) >= 1\}$$

$$W_{written}(t, \tau) = \{b : WriteCount(b, t - \tau, t) >= 1\}$$

$$W_{both}(t, \tau) = W_{read}(t, \tau) \cap W_{written}(t, \tau)$$

On average, the daily working set for the various workloads ranges from just over 4 percent (PC workloads) to about 7 percent of the storage used (FS1). The size of the working set is not constant but fluctuates day-to-day so that the maximum working set is several times larger than the average. Notice from Figure 16 that the working set of blocks that are both read and written is small, representing less than 25 percent of the total working set size for all the workloads. To better understand the interaction between the blocks that are read and those that are written,

Figure 16 Daily working set size



we introduce the idea of the generalized working set $W(t, \tau, c) = \{b : Count(b, t - \tau, t) >= c\}$. The working set first introduced in Reference 37 is simply the special case where $c = 1$. Figure 17 presents the average daily generalized working set size for our workloads as a function of c, the minimum number of times a block is referenced in a day for it to be considered part of the working set. The figure shows that for all the workloads, the relationship between the average size of the daily generalized working set and c can be approximately described by a reciprocal function of the form $f(c) = a/c^b$ where a and b are positive constants. That the working set decreases sharply as c increases beyond unity indicates that *only a small fraction of the data stored is in active use, suggesting that it is probably a good idea to identify the blocks that are in use and to optimize their layout as in Reference 4*. Notice also that *the amount of data that is both actively read and updated is clearly very small*. In the next section, we examine this further by looking at the dependencies between reads and writes.

Read/write dependencies. Dependencies are generally classified into three categories: *true dependencies* (read-after-write or RAW), *output dependencies* (write-after-write or WAW), and *anti dependencies* (write-after-read or WAR). A RAW is said to exist between two operations if the first operation writes a

Figure 17 Average daily generalized working set size

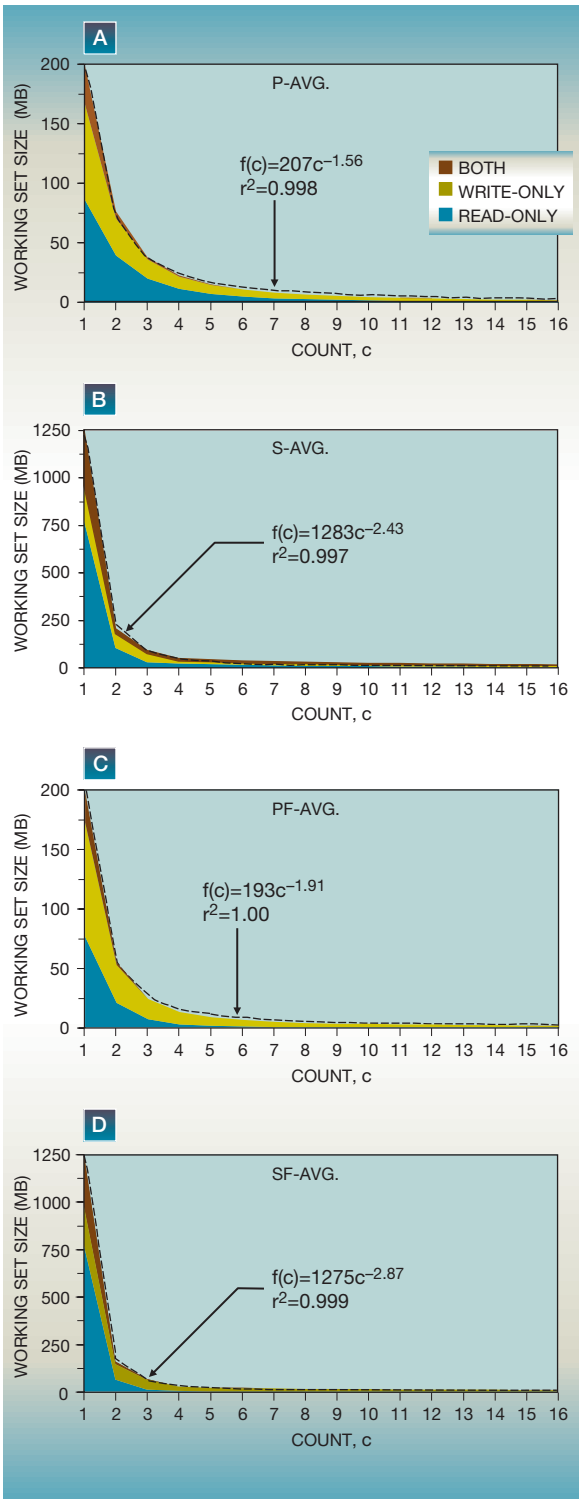
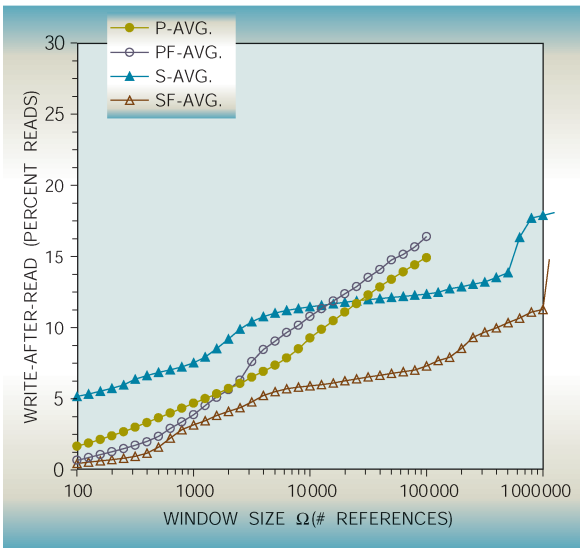


Figure 18 Frequency of occurrence of write-after-read



block that is later read by the second operation and there is no intervening operation on the block. WAW and WAR are similarly defined.

In Figure 18, we plot the percentage of reads for which there is a write within Ω references that constitute a WAR. We refer to Ω as the window size. Observe that even for a large window size of 100000 references, less than 25 percent of the reads fall into this category for all the workloads. In other words, *blocks that are read tend not to be updated so that if disk blocks are replicated or reorganized based on their read access patterns, write performance will not be significantly affected*. Notice from Figure 19 that all the workloads contain more WAW than RAW. This implies that *updated blocks are more likely to be updated again than to be read, suggesting that if we do replicate blocks, we should only update one of the copies and invalidate the rest rather than update all the copies*. In other words, a write-invalidate policy will likely work better than a write-broadcast policy. Again, we see that the results for the filtered traces are quantitatively different from those for the original traces, but they lead to the same conclusions.

For the PC traces, we are able to match up I/O requests with the corresponding filename. To better understand the dependencies, we rerun the RAW and WAW analysis for these workloads excluding references to the file system meta-data and to log files,

which respectively constitute about half, and a quarter, of all the write requests. The results are summarized in Figure 20. We consider meta-data references to be those that access blocks belonging to directories and to the following files: $mft, $attrdef, $bitmap, $boot, $logfile, $mftmirr, and $upcase. Log references are taken to be those that access blocks belonging to the following files: *\config*, *\profiles*, *.log, log.*, and netscape.hst. Observe that once the meta-data and log references are filtered out, it is still the case that updated blocks are more likely to be updated again than to be read, but less so. In other words, although meta-data and log writes make it more likely for an updated block to be updated again than to be read, they are not the only cause for this behavior.

Conclusions

In this paper, we empirically analyze the I/O traffic of a wide range of real workloads with an emphasis on understanding how these workloads will respond to new storage developments such as network storage, storage utilities, and intelligent self-optimizing storage. As part of our analysis, we also study the effect of increased upstream caching on the traffic characteristics seen by the storage system and discover that it affects our analysis only quantitatively. Our major findings follow.

Importance of I/O innovation/optimization—I/O is known to be a major component of server workloads, and improving the I/O performance for these workloads is critical. Our results suggest that if processors continue to increase in performance according to Moore's Law, I/O is likely to also become a dominant component of personal computer workloads in the next few years. Our data show that consistently across all the workloads, writes account for about 60 percent of the requests. However, just as read caching by the file system or the database buffer can eliminate most of the reads, if writes are delayed long enough (e.g., by using nonvolatile memory), write requests can similarly be very significantly reduced. In fact, for practically all the workloads, a small write-back cache of 1 MB eliminates more than half the writes. Therefore, we believe that the performance of read I/Os is likely to continue to have a direct impact on application performance. As part of our analysis, we re-examine Amdahl's rule of thumb for a balanced system and discover that our server workloads generate on the order of 0.05 bits of physical I/O per instruction, consistent with our earlier work using the production database workloads of some

Figure 19 Frequency of occurrence of read-after-write and write-after-write

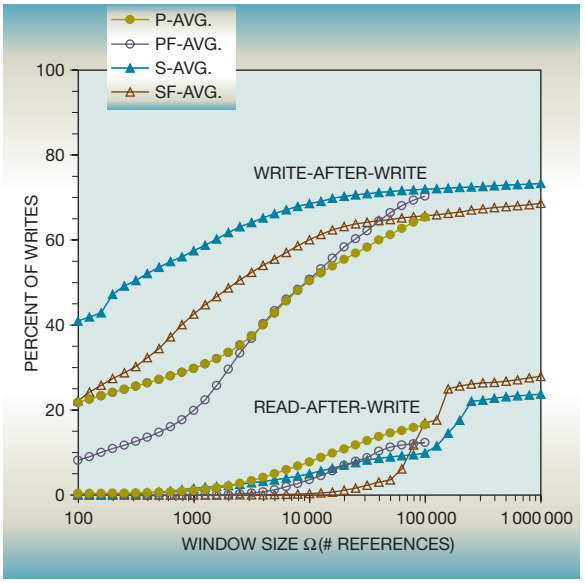
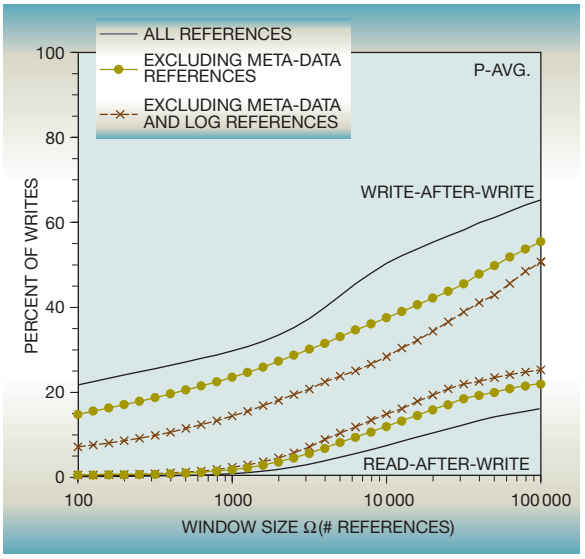


Figure 20 Frequency of occurrence of read-after-write and write-after-write when meta-data and/or log references are excluded



of the world's largest corporations.[10] The figure for the PC workloads is seven times lower at about 0.007

bits of physical I/O per instruction per second. We also find that the average request size is about 8 KB.

Burstiness of I/O traffic—Across all the workloads, read and write I/O requests seldom occur singly but tend to arrive in groups. We find that the write queue is very much deeper than the read queue. Our analysis also indicates that there is little cross-correlation in traffic volume among the server workloads, suggesting that aggregating them will likely help to smooth out the traffic and enable more efficient utilization of resources. As for the PC workloads, multiplexing them will remove the high frequency fluctuations in I/O traffic, but some of the time-of-day effects are likely to remain unless the PCs are geographically distributed in different time zones. In addition, our results show that to satisfy I/O bandwidth requirements 99 percent of the time, we would need to provision for 15 times the long-run average bandwidth. Going beyond 99 percent of the time would require disproportionately more resources. It turns out that for time scales ranging from tens of milliseconds to tens and sometimes even hundreds of seconds, the I/O traffic is well-represented by a self-similar process. This implies that the I/O system may become overwhelmed at much lower levels of utilization than expected with the commonly assumed Poisson model. Such behavior has to be taken into account when designing storage systems, and in the service level agreements (SLAs) when outsourcing storage. We recommend that I/O traffic be characterized by a three-tuple consisting of the mean and variance of the arrival rate, and the Hurst parameter.

Potential for harnessing "free" resources—We find that our PC workloads contain a lot of processor idle time for performing background tasks, even without having to deliberately leave the computer on when the user is away. The storage system is also relatively idle. For all the workloads, a system that exploits idle time can get most of the potential benefit by simply focusing on the long idle periods. In both the PC and server environments, there are idle periods that are both long enough and that occur frequently enough to be interesting for off-line optimizations such as block reorganization. In the server environment, we might have to be more meticulous in using the available idle time, for instance, by dividing an idle-time task into several finer-grained steps that can be scheduled whenever there is a short idle period. Our results suggest that the length of an idle period can be predicted more accurately by conditioning (using conditional probabilites) on the amount of time the system has already been idle than from the lengths of the recent idle periods.

Opportunity for block reorganization—In general, I/O traffic is low enough so that collecting a daily trace of block references for later analysis and optimization results in a manageable amount of data. We discover that only a small fraction of the data stored is in active use, suggesting that it is probably a good idea to identify the blocks that are in use and to optimize their layout. In addition, the amount of data that is both actively read and updated is very small. Moreover, blocks that are read tend not to be updated so that if blocks are reorganized or replicated based on their read access patterns, write performance will not be significantly affected. Because updated blocks are more likely to be updated again than to be read, if blocks are replicated, a write-invalidate policy will tend to work better than a write-broadcast policy.

Acknowledgments

The authors would like to thank Ruth Azevedo, Carlos Fuente, Jacob Lorch, Bruce McNutt, Anjan Sen, and John Wilkes for providing the traces used in this study. In addition, the authors are grateful to Jai Menon, John Palmer, and Honesty Young for helpful comments on versions of this paper.

Funding for this research has been provided by the State of California under the MICRO program, and by AT&T Laboratories, Cisco Corporation, Fujitsu Microelectronics, IBM Corporation, Intel Corporation, Maxtor Corporation, Microsoft Corporation, Sun Microsystems, Toshiba Corporation, and Veritas Software Corporation.

Cited references

1. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Second Edition, Morgan Kaufmann Publishers, Inc., San Francisco, CA (1996).
2. E. Grochowski, "IBM Magnetic Hard Disk Drive Technology," Hitachi Global Storage Technologies, http://www.hgst.com/hdd/technolo/grochows/grocho01.htm.
3. W. W. Hsu, A. J. Smith, and H. C. Young, "Projecting the Performance of Decision Support Workloads on Systems with Smart Storage (SmartSTOR)," *Proceedings of IEEE International Conference on Parallel and Distributed Systems*

(ICPADS), Iwate, Japan, July 2000, IEEE, New York (2000), pp. 417–425.
4. W. W. Hsu, A. J. Smith, and H. C. Young, "The Automatic Improvement of Locality in Storage Systems," Technical Report, Computer Science Division, University of California, Berkeley (2003, in preparation). Available as Chapter 4 of Reference 38.
5. W. W. Hsu and A. J. Smith, "The Real Effect of I/O Optimizations and Disk Improvements," Technical Report, Computer Science Division, University of California, Berkeley (2003, in preparation). Available as Chapter 3 of Reference 38.
6. W. W. Hsu and A. J. Smith, "Characteristics of I/O Traffic in Personal Computer and Server Workloads," Technical Report CSD-02-1179, Computer Science Division, University of California, Berkeley (April 2002). Available at http://sunsite.berkeley.edu/TechRepPages/CSD-02-1179.
7. M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout, "Measurements of a Distributed File System," *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, Pacific Grove, CA, October 1991, ACM, New York (1991), pp. 198–212.
8. W. Vogels, "File System Usage in Windows NT 4.0," *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, ACM, New York (1999), pp. 93–109.
9. D. Roselli, J. R. Lorch, and T. E. Anderson, "A Comparison of File System Workloads," *Proceedings of USENIX Annual Technical Conference*, San Diego, CA, June 2000, USENIX Association, Berkeley, CA (2000), pp. 41–54.
10. W. W. Hsu, A. J. Smith, and H. C. Young, "Characteristics of Production Database Workloads and the TPC Benchmarks," *IBM Systems Journal* **40**, No. 3, 781–802 (2001).
11. W. W. Hsu, A. J. Smith, and H. C. Young, "I/O Reference Behavior of Production Database Workloads and the TPC Benchmarks—An Analysis at the Logical Level," *ACM Transactions on Database Systems* **26**, 96–143 (March 2001).
12. A. J. Smith, "Disk Cache—Miss Ratio Analysis and Design Considerations," *ACM Transactions on Computer Systems* **3**, 161–203 (August 1985).
13. P. Biswas, K. K. Ramakrishnan, and D. Towsley, "Trace Driven Analysis of Write Caching Policies for Disks," *Proceedings of ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Santa Clara, CA, May 1993, ACM, New York (1993), pp. 13–23.
14. R. Karedla, J. S. Love, and B. G. Wherry, "Caching Strategies to Improve Disk System Performance," *Computer* **27**, 38–46 (March 1994).
15. C. Ruemmler and J. Wilkes, "UNIX Disk Access Patterns," *Proceedings of USENIX Winter Conference*, San Diego, CA, January 1993, USENIX Association, Berkeley, CA (1993), pp. 405–420.
16. J. R. Heath and S. A. R. Houser, "Analysis of Disk Workloads in Network File Server Environments," *Proceedings of Computer Measurement Group (CMG) Conference*, Nashville, TN, December 1995, Computer Measurement Group (1995), pp. 313–322.
17. J. R. Lorch and A. J. Smith, "The VTrace Tool: Building a System Tracer for Windows NT and Windows 2000," *MSDN Magazine* **15**, 86–102 (October 2000).
18. IBM Corporation, *AIX Versions 3.2 and 4 Performance Tuning Guide*, 5th Edition (1996).
19. M. Rosenblum, E. Bugnion, S. A. Herrod, E. Witchel, and A. Gupta, "The Impact of Architectural Trends on Operating System Performance," *Proceedings of 15th ACM Symposium on Operating Systems Principles (SOSP)*, Copper Mountain, CO, December 1995, ACM, New York (1995), pp. 285–298.
20. G. M. Amdahl, "Storage and I/O Parameters and Systems Potential," *Proceedings of IEEE International Computer Group Conference (Memories, Terminals, and Peripherals)*, Washington, DC, June 1970, IEEE, New York (1970), pp. 371–372.
21. D. Bhandarkar and J. Ding, "Performance Characterization of the Pentium Pro Processor," *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA)*, San Antonio, TX, February 1997, IEEE, New York (1997), pp. 288–297.
22. A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood, "DBMSs on a Modern Processor: Where Does Time Go," *Proceedings of International Conference on Very Large Data Bases (VLDB)*, Edinburgh, Scotland, September 1999, Morgan Kaufman Publishers, San Francisco, CA (1999), pp. 266–277.
23. K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker, "Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads," *Proceedings of ACM International Symposium on Computer Architecture (ISCA)*, Barcelona, Spain, June 1998, ACM, New York (1998), pp. 15–26.
24. B. McNutt, "MVS DASD Survey: Results and Trends," *Proceedings of Computer Measurement Group (CMG) Conference*, Nashville, TN, December 1995, Computer Measurement Group (1995), pp. 658–667.
25. J. B. Major, "Processor, I/O Path, and DASD Configuration Capacity," *IBM Systems Journal* **20**, No. 1, 63–85 (1981).
26. Y. Chen, W. W. Hsu, and H. C. Young, "Logging RAID—An Approach to Fast, Reliable, and Low-Cost Disk Arrays," *Proceedings of European Conference on Parallel Computing (EuroPar)*, Munich, Germany, August 2000, Springer, New York (2000), pp. 1302–1312.
27. B. B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freeman, New York (1982).
28. S. D. Gribble, G. S. Manku, D. Roselli, E. A. Brewer, T. J. Gibson, and E. L. Miller, "Self-Similarity in File Systems," *Proceedings of ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Madison, WI, June 1998, ACM, New York (1998), pp. 141–150.
29. M. E. Gómez and V. Santonja, "Analysis of Self-Similarity in I/O Workload Using Structural Modeling," *Proceedings of Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, College Park, MD, October 1999, IEEE, New York (1999), pp. 234–242.
30. W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)," *IEEE/ACM Transactions on Networking* **2**, 1–15 (February 1994).
31. W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," *IEEE/ACM Transactions on Networking* **5**, 71–86 (February 1997).
32. R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes, "Idleness Is Not Sloth," *Proceedings of USENIX Technical Conference*, New Orleans, LA, January 1995, USENIX Association, Berkeley, CA (1995), pp. 201–212.
33. J. Ousterhout and F. Douglis, "Beating the I/O Bottleneck: A Case for Log-Structured File Systems," *Operating Systems Review* **23**, 11–28 (January 1989).
34. M. Russinovich, "Inside the Cache Manager," *Windows & .NET Magazine*, October 1998, http://www.winnetmag.com/.
35. K. K. Ramakrishnan, P. Biswas, and R. Karedla, "Analysis of File I/O Traces in Commercial Computing Environments,"

Proceedings of ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Newport, RI, June 1992, ACM, New York (1992), pp. 78–90.
36. M. D. Dahlin, *Serverless Network File Systems*, Ph.D. thesis, University of California, Berkeley, CA (December 1995).
37. P. J. Denning, "The Working Set Model for Program Behaviour," *Communications of the ACM* **11**, No. 5, 323–333 (May 1968).
38. W. W. Hsu, "Dynamic Locality Improvement Techniques for Increasing Effective Storage Performance, Ph.D. thesis, University of California, Berkeley, CA (December 2002). Available as Technical Report CSD-03-1223, Computer Science Division University of California, Berkeley (January 2003).

Accepted for publication December 16, 2002.

Windsor W. Hsu *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (windsor@almaden.ibm.com).* Dr. Hsu received the B.S. degree in electrical engineering and computer sciences, and the M.S. and Ph.D. degrees in computer science from the University of California, Berkeley. Since 1996, he has been with the Computer Science Department at the IBM Almaden Research Center. His research interests include computer architecture and the performance analysis and modeling of computer systems. Dr. Hsu has received the IBM Supplemental Patent Issue Award (2002), IBM Invention Achievement Award (2001, 2000), IBM Fellowship (1995–1998), UC Regents' Fellowship (1994–1995), UC Chancellor's Scholarship (1993–1994), and UC Regents' Scholarship (1991–1994).

Alan Jay Smith *Computer Science Division, University of California, Berkeley, California 94720 (smith@cs.berkeley.edu).* Dr. Smith received the B.S. degree in electrical engineering from the Massachusetts Institute of Technology, and the M.S. and Ph.D. degrees in computer science from Stanford University. He was an NSF Graduate Fellow. He is currently a professor in the Computer Science Division of the Department of Electrical Engineering and Computer Sciences (EECS), University of California, Berkeley, where he has been on the faculty since 1974; he was vice chairman of the EECS department from July 1982 to June 1984. His research interests include the analysis and modeling of computer systems and devices, computer architecture, and operating systems. Dr. Smith is a Fellow of the IEEE, of the ACM, and of the AAAS (American Association for the Advancement of Science), and is a member of IFIP Working Group 7.3, the Computer Measurement Group, Eta Kappa Nu, Tau Beta Pi, and Sigma Xi. He is on the Board of Directors (1993–2005) and was Chairman (1991–1993) of the ACM Special Interest Group on Computer Architecture (SIGARCH), was Chairman (1983–1987) of the ACM Special Interest Group on Operating Systems (SIGOPS), was on the Board of Directors (1985–1989) of the ACM Special Interest Group on Measurement and Evaluation (SIGMETRICS), was an ACM National Lecturer (1985–1986) and an IEEE Distinguished Visitor (1986–1987), was an associate editor of the ACM *Transactions on Computer Systems* (TOCS) (1982–1993), is a subject area editor of the *Journal of Parallel and Distributed Computing* and is on the editorial board of the *Journal of Microprocessors and Microsystems*. He was program chairman for the Sigmetrics '89/Performance '89 Conference, program co-chair for the second (1990), sixth (1994), and ninth (1997) Hot Chips conferences, and has served on numerous program committees.