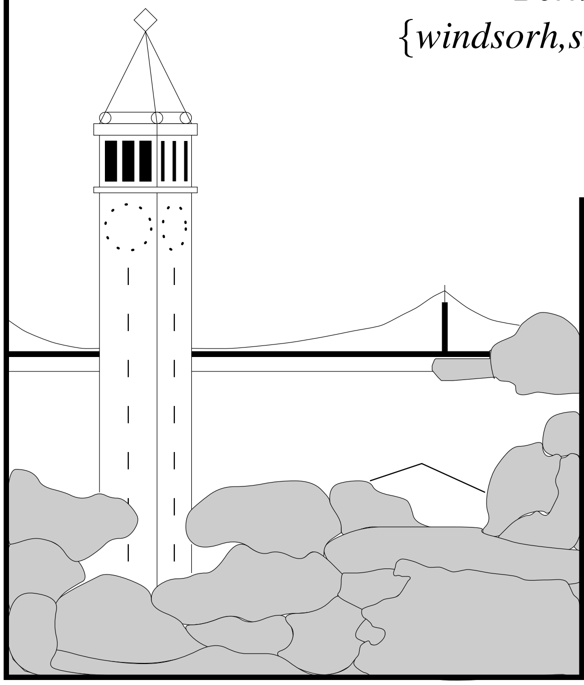
Characteristics of I/O Traffic in Personal Computer and Server Workloads

Windsor W. Hsu[†⋆]
Alan Jay Smith[⋆]

[†]*Storage Systems Department*
Almaden Research Center
IBM Research Division
San Jose, CA 95120
windsor@almaden.ibm.com

[⋆]*Computer Science Division*
EECS Department
University of California
Berkeley, CA 94720
{windsorh,smith}@cs.berkeley.edu



Report No. UCB/CSD-02-1179

April 2002

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Characteristics of I/O Traffic in Personal Computer and Server Workloads

Windsor W. Hsu[†⋆]
Alan Jay Smith[⋆]

[†]Storage Systems Department
Almaden Research Center
IBM Research Division
San Jose, CA 95120
windsor@almaden.ibm.com

[⋆]Computer Science Division
EECS Department
University of California
Berkeley, CA 94720
{windsorh,smith}@cs.berkeley.edu

Abstract

Understanding the characteristics of physical I/O traffic is increasingly important as the performance gap between processor and disk-based storage continues to widen. Moreover, recent advances in technology, coupled with market demands, have led to several new and exciting developments in storage, including network storage, storage utilities, and intelligent self-optimizing storage. In this paper, we empirically examine the I/O traffic of a wide range of real PC and server workloads with the intent of understanding how well they will respond to these new storage developments. As part of our analysis, we compare our results with historical data and reexamine rules of thumb that have been widely used for designing computer systems. Our results show that there is a strong need to focus on improving I/O performance. We find that the I/O traffic is bursty and appears to exhibit self-similar characteristics. In addition, our analysis indicates that there is little cross-correlation in traffic volume among the server workloads, which suggests that aggregating these workloads will likely help to smooth out the traffic and enable more efficient utilization of resources. We also discover that there is a lot of potential for harnessing "free" system resources for purposes such as automatic optimization of disk block layout. In general, the characteristics of the I/O traffic are relatively insensitive to the amount of caching upstream and our qualitative results apply when the upstream cache is increased in size.

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, Intel Corporation, Maxtor Corporation, Microsoft Corporation, Sun Microsystems, Toshiba Corporation and Veritas Software Corporation.

1 Introduction

Processor performance has been increasing at the rate of 60% per year while disk access time, being limited by mechanical delays, has been improving by less than 10% per year [16, 39]. Compounding this widening performance gap between processor and disk storage is the fact that disk capacity has been improving by more than 60% per year [16, 39] so that each disk is responsible for the storage and retrieval of rapidly increasing amounts of data. The overall result of these technology trends, which show no signs of easing, is that computer systems are increasingly bottlenecked by disk-based storage systems. The key step in overcoming this bottleneck is to understand how storage is actually used so that new optimization techniques and algorithms can be designed.

In addition, new paradigms and developments have recently emerged in the storage industry, and determining the real effect of these requires a focused examination of the I/O characteristics of real workloads. First, storage is increasingly placed on some form of general network so that it can be shared and accessed directly by several computers at the same time [48] (*e.g.,* Network Attached Storage (NAS) for file storage and Storage Area Networks (SANs) for block storage). The performance of such network storage hinges on knowing the I/O traffic patterns and optimizing the network for such patterns. Second, consolidating the storage now distributed throughout an organization, for instance to storage utilities or Storage Service Providers (SSPs), is expected to become increasingly popular [28]. Whether such an approach leads to more efficient pooling of resources among different groups of users depends on the characteristics of their workloads, specifically on whether the workloads are independent. In practice, we will need rules of thumb that describe the storage and performance requirements of each group, as well as realistic traffic models. Third, the rapid growth in available processing power in the storage system [14, 19] makes it possible to build intelligent storage systems that can dynamically optimize themselves for the workload [22]. The design of these systems requires a good understanding of how real workloads behave.

In this research, therefore, we empirically examine the storage usage characteristics of real users 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 I/O traffic, specifically, (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 to note any 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 [18]. The insights gained from this research are instrumental to the block reorganization technique outlined in [22].

The rest of this paper is organized as follows. Section 2 contains a brief overview of previous work in characterizing I/O behavior. Section 3 discusses our methodology and describes the traces that we use. In Sections 4-7, we analyze the I/O traffic of our various workloads in detail. Concluding remarks appear in Section 8. Because of the huge amount of data that is involved in this study, we present only a characteristic cross-section in the main text. More detailed graphs and data are presented in Appendix A. Some of the more involved mathematical material appears in Appendix B.

2 Related Work

I/O behavior at the file system level has been characterized in some detail (*e.g.,* [4, 8, 36, 42, 51]). There have also been several studies of the logical I/O characteristics of large database and scientific systems; see [20, 21] for a brief bibliography. These studies provide valuable insights for designing the file system and the database data manager but they are not very useful for understanding what happens at the physical or storage level. Because of the file system cache or the database buffer pool, most of the logical references never reach the physical storage. In addition, the logical I/O behavior does not reflect the effects of file allocation and mapping. Furthermore, many of these studies do not account for system generated traffic such as paging and metadata access, which can account for a significant fraction of the total I/O [42, 45].

Compared to the analysis of I/O behavior at the logical level, physical I/O characterization has received much less attention in the research community. Part of the reason is that storage level characteristics are sensitive to the file system or 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 different systems. Traces collected from large IBM mainframe installations [49] and production VAX/VMS systems [7, 24] 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 [45] and Novel NetWare file servers [17] in academic/research environments. Even though personal computers (PCs) running various flavors of MS Windows are now an integral part of many office activities, there has, to the best of our knowledge, been no published systematic analysis of how storage is used in such systems.

3 Methodology

Trace data can generally be gathered at different levels in the system depending on the purpose of 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. For instance, to study physical storage systems, we could filter a logical trace through models of the file system layer to obtain a trace of the 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 records information about the requests before passing them 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 [42] show that 15% of reads and nearly 30% of writes in Windows NT workloads can be attributed to paging by running programs. In addition, 85% of processes now memory-map files compared with 36% that read files and 22% that write them. From a practical perspective, the approach of starting with a logical trace to evaluate physical storage systems requires that a lot of data be collected, which adds 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 is laid out and how the metadata is referenced. For today's well-tuned systems, each of these components is complicated 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 in general, for the purpose of studying physical storage systems, analyzing traces collected at the physical level is more practical and realistic. This is the method we use in this paper.

In order to make our characterization more useful for subsequent mathematical analyses and modeling by others, we have fitted our data to various functional forms through non-linear regression, which we solved by using the Levenberg-Marquardt method [40]. When appropriate, we also fitted standard probability distributions to our data by using the method of maximum likelihood to obtain parameter estimates and then optimizing these estimates by the Levenberg-Marquardt algorithm [40].

3.1 Trace Collection

The traces analyzed in this study were collected from three different platforms, namely Windows NT, IBM AIX and HP-UX. A different trace facility was used on each platform. The Windows NT traces were collected by using VTrace [29], 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 are mainly interested in the disk activities, which are collected by VTrace through the use of device filters. VTrace takes daily snapshots of the NTFS file system metadata. In addition, it collects data on the file system as well as process and thread activities. We have verified the disk activity collected by VTrace by comparing it with the raw SCSI traffic obtained by a SCSI analyzer. Details of VTrace and the special techniques used to collect the relevant data with minimal intrusion can be found in [29].

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. In most cases, we are able to match up the file object pointer with a file open record to 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 periodic metadata snapshots.

Because VTrace is 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 up 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 will look at this in Section 4.1.

To keep the amount of data collected manageable, process and thread trace records are gathered only for 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 to delete trace records that occur after ten minutes of user idle time. In other words, we use only the trace records that occur from the first user activity after an idle period to the last user activity before an idle period; we assume that there is no activity in the system during the periods when the user is idle. 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 latter 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 power usage.

Both the IBM AIX and HP-UX traces were collected using kernel-level trace facilities built into the operating systems. These trace facilities are completely transparent to the user and adds no noticeable processor load. Among the information collected for each physical I/O are: 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 [23]. The HP-UX trace facility is described in [45].

Design-ation	User Type	System Configuration					Trace Characteristics			
		System	Memory (MB)	File Systems	Storage Used[i] (GB)	# Disks	Duration	Footprint[ii] (GB)	Traffic (GB)	Requests (10^6)
P1	Engineer	333MHz P6	64	1GB FAT[i] 5GB NTFS[i]	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

(a) Personal Systems.

Design-ation	Primary Function	System Configuration					Trace Characteristics			
		System	Memory (MB)	File Systems	Storage Used[i] (GB)	# Disks	Duration	Footprint[ii] (GB)	Traffic (GB)	Requests (10^6)
FS1	File Server (NFS[iii])	HP 9000/720 (50MHz)	32	3 BSD[iii] FFS[iii] (3 GB)	3	3	45 days (4/25/92 - 6/8/92)	1.39	63	9.78
FS2[iv]	File Server (AFS[iii])	IBM RS/6000	-	23 AIX[iii] JFS[iii] (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[iii] (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.[v]	-	-	299	-	18.5	8	32.3 days	4.22	74.6	12.1

[i] Sum of all the file systems and allocated volumes.
[ii] Amount of data referenced at least once (using block size of 512 bytes)
[iii] AFS – Andrew Filesystem, AIX – Advanced Interactive Executive (IBM's flavor of UNIX), BSD – Berkeley System Development Unix, ERP – Enterprise Resource Planning, FFS – Fast Filesystem, JFS – Journal Filesystem, NFS – Network Filesystem, NTFS – NT Filesystem, SMP – Symmetric Multiprocessor
[iv] Only per second I/O statistics were collected.
[v] Excluding FS2.

(b) Servers.

Table 1: Trace Description.

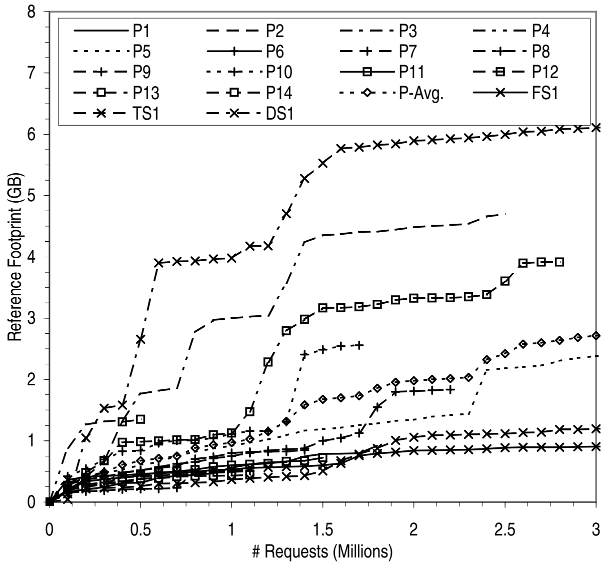


Figure 1: Footprint Vs. Number of References.

3.2 Trace Description

In this study, we use traces collected from both server and personal computer (PC) systems. Table 1 summarizes the characteristics of the traces. The *footprint* of a trace is defined as the amount of data referenced at least once in the trace. Figure 1 plots the trace footprint as a function of the number of references, which is a measure of the trace length. Similar plots for the read footprint and the write footprint are in Figure A-1 in Appendix A.

The PC traces are denoted as P1, P2, ..., P14. The term "P-Avg." represents the arithmetic mean of the results for the PC traces. These traces were collected over a period ranging from about a month to well over nine months on PCs running Windows NT. In this study, we utilize only the first 45 days of the traces. In addition to engineers and graduate students, the users of these systems also include a secretary and several people in senior managerial positions. By having users ranging from managers and a secretary to hard core engineers in our sample, we believe that our traces are illustrative of the PC workloads in many offices, especially those involved in research and development. Note, however, that the traces should not be taken as typical or representative of any other system. Despite this disclaimer, the fact that many of our results correspond to those obtained previously, albeit in somewhat different environments, suggest that our findings are generalizable to a large extent.

The servers examined include two file servers, a time-sharing system and a database server. Throughout this paper, we use the term "S-Avg." to denote the arithmetic mean of the results for the server workloads. The first file server workload (FS1) was taken off a file server for nine clients at the University of California, Berkeley. This system was primarily used for compilation and editing. It is referred to as "Snake" in [45]. The second file server workload (FS2) was taken off an Andrew File System (AFS) server at one of the major development sites of a leading computer storage vendor. The system was the primary server used to support the 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 trace denoted TS1 was gathered on a time-sharing system at an industrial research laboratory. It was mainly used for news, mail, text editing, simulation and compilation. It is referred to as "cello" in [45]. The database server trace (DS1) was collected at one of the largest health insurers nationwide. The system traced was running an Enterprise Resource Planning (ERP) application on top of a commercial database.

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 the qualitative results still apply. To show that our characterization is relatively insensitive to the amount of caching upstream, 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. We denote these filtered traces by adding an "f" to the original designation. For instance, the trace obtained by filtering P1 is denoted as P1f. We also denote the average result for the filtered PC workloads as "Pf-Avg" and that for the filtered server workloads as "Sf-Avg". Following the design of most file systems, we allow a dirty block 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).

In Table 2, we present the faction of I/O activity that is filtered out by such a cache. On average, over 50% 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 hitting in the cache. Furthermore, by delaying the writes, we are able to consolidate them into larger sequential writes. In Table 3 and Figure 2, we present the request size distribution for both the original and the filtered traces. Although the average request size of writes is increased, the request size distributions of the filtered traces track those 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 in the rest of the paper.

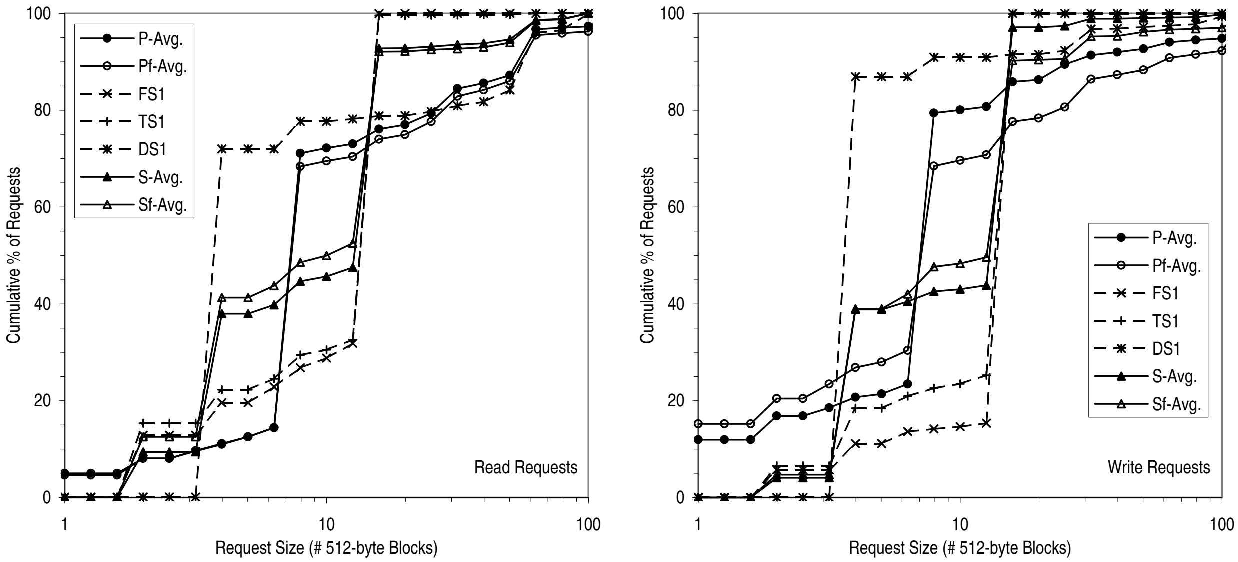


Figure 2: Distribution of Request Size.

	Number of MBs			Number of Requests		
	Read	Write	Overall	Read	Write	Overall
P1	0.575	0.176	0.441	0.618	0.575	0.605
P2	0.503	0.173	0.385	0.547	0.495	0.525
P3	0.583	0.163	0.291	0.632	0.498	0.537
P4	0.301	0.175	0.219	0.358	0.630	0.527
P5	0.369	0.232	0.275	0.438	0.620	0.574
P6	0.831	0.190	0.436	0.821	0.548	0.617
P7	0.546	0.143	0.246	0.551	0.548	0.549
P8	0.592	0.239	0.426	0.629	0.657	0.642
P9	0.484	0.146	0.317	0.488	0.471	0.479
P10	0.216	0.162	0.192	0.316	0.537	0.436
P11	0.515	0.245	0.409	0.520	0.641	0.577
P12	0.416	0.179	0.290	0.450	0.721	0.601
P13	0.557	0.257	0.391	0.585	0.615	0.603
P14	0.356	0.221	0.282	0.415	0.683	0.596
P-Avg.	0.489	0.193	0.329	0.526	0.589	0.562
FS1	0.594	0.573	0.582	0.570	0.681	0.633
TS1	0.583	0.394	0.474	0.546	0.454	0.495
DS1	0.057	0.203	0.122	0.133	0.702	0.488
S-Avg.	0.412	0.390	0.393	0.416	0.612	0.539

Table 2: Fraction of I/O Activity that is Filtered.

	All Requests				Read Requests				Write Requests			
	Avg.	Std. Dev.	Min.	Max.	Avg.	Std. Dev.	Min.	Max.	Avg.	Std. Dev.	Min.	Max.
P1	19.1	26.6	1	128	17.7	22	1	128	22.4	35.4	1	128
P2	17.2	27.4	1	1538	19.1	24.4	1	128	14.6	30.9	1	1538
P3	15.5	24.8	1	128	15.5	19.4	1	128	15.5	26.8	1	128
P4	21.7	33.8	1	128	20.4	30.3	1	128	22.5	35.8	1	128
P5	16.3	25	1	298	20.8	28.3	1	129	14.8	23.6	1	298
P6	15.7	23.7	1	128	23.1	25.5	1	128	14.7	23.2	1	128
P7	18.5	30.3	1	128	19.1	23.9	1	128	18.4	31.9	1	128
P8	17.4	25.8	1	128	16.8	20.9	1	128	18.2	30.9	1	128
P9	14.7	21.1	1	128	15.4	20.2	1	128	13.9	21.8	1	128
P10	19.6	30.7	1	128	23.7	32.8	1	128	15.7	28	1	128
P11	15.2	23.1	1	128	19.4	24.7	1	128	11.7	21.1	1	128
P12	25.3	58.6	1	512	27.5	54.6	1	512	23.6	61.4	1	512
P13	12.3	18.2	1	180	14.5	18.8	1	128	11	17.7	1	180
P14	16.1	28.1	1	1539	20.6	31.2	1	128	14	26.2	1	1539
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
FS1	13.5	5.08	2	512	12.5	5.47	2	64	14.2	4.65	2	512
TS1	12.9	7.77	2	512	12.4	6.52	2	224	13.3	8.62	2	512
DS1	11.9	21.9	1	512	17.4	27.1	1	512	8.55	17.3	1	256
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

Table 3: Request Size (Number of 512-Byte Blocks).

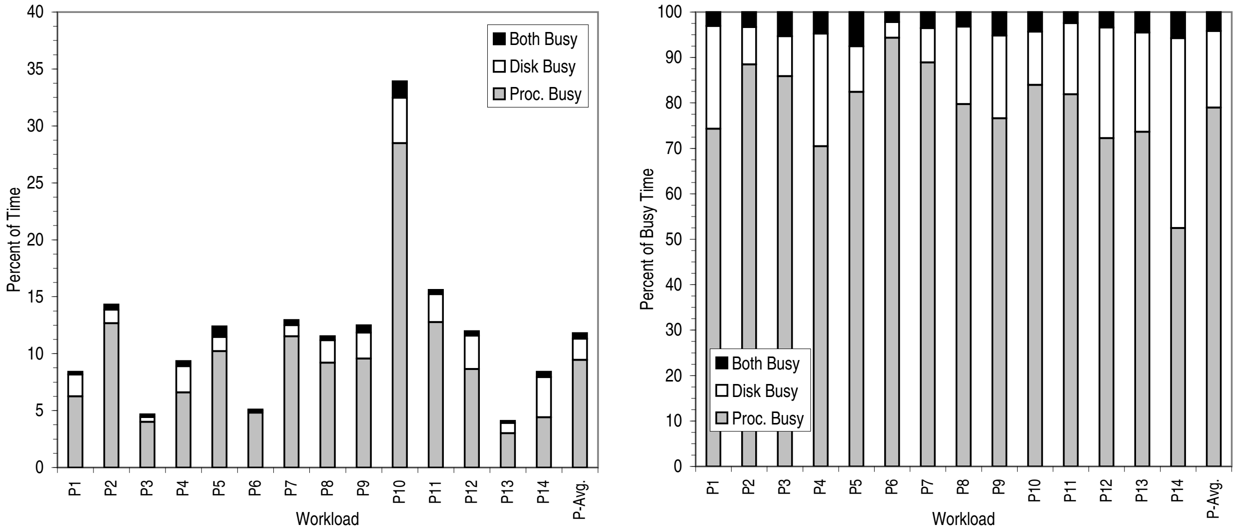


Figure 3: Disk and Processor Busy Time.

4 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 is the I/O component in the overall workload, how many I/Os are generated, and how fast do the requests arrive.

4.1 Overall Significance of I/O

In Figure 3, we present the percent 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. The processor busy time is obtained 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 we only have trace data for the periods during which user input activity occurs at least once every ten minutes. In other words, we consider 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% of the time while the disk is busy for only about 2.5% of the time. This low level of busy time is misleading, however, because the user is interested in response time; CPU idle generally represents user think time, and would occur in any case in a single user environment. Thus we cannot conclude that the processor and I/O system are "fast enough". What the results do suggest is that *there is a lot of idle time for performing background tasks, even without having to deliberately leave the computer on when the user is away*. In other words, significant resources are available without requiring additional power consumption. 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 [32]. In addition, the idle resources can be used to optimize the system so that it will perform better in future for the foreground task (*e.g.,* [22]). We will characterize the disk idle periods in detail in Section 5.3.

I/O is known to be a major component of server workloads (*e.g.,* [43]). But if processors continue to increase in performance according to Moore's Law (60% per year) as many believe they will, *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 3, this technique appears to be relatively ineffective for the PC workloads since *only a small fraction (20% on average) of the disk busy time is overlapped with computation*. In Figure 4, 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% 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

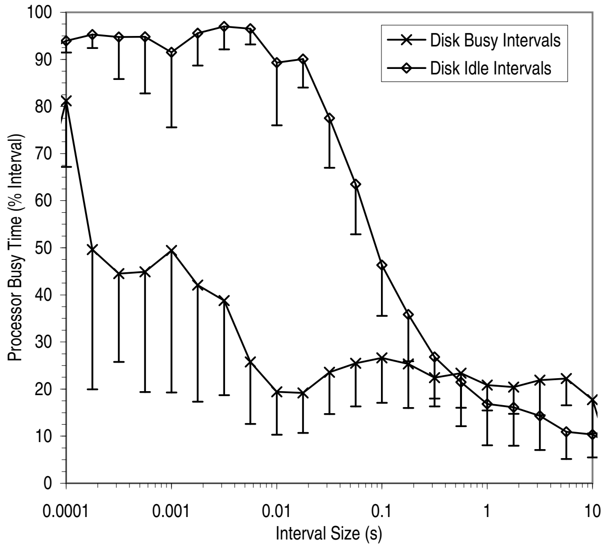


Figure 4: Processor Busy Time during Disk Busy/Idle Intervals. Bars indicate standard deviation (To reduce clutter, we show only the deviation in one direction).

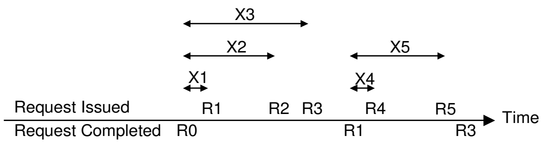


Figure 5: Intervals between Issuance of I/O Requests and Most Recent Request Completion.

all the disks are idle but the processor utilization drops to less than 50% when one or more of the disks are busy. Such results imply that *little processing can be overlapped with I/O so that I/O response time is important for these kinds of workloads.*

That only a small amount of processing is overlapped with I/O suggests that there is effectively little multiprocessing in the PC workloads. Such predominantly single-process 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 can be issued. For instance, in the timeline in Figure 5, 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 is completed earlier, R1, R2 and R3 will be dragged forward and issued earlier. If this in turn causes R1 to be finished earlier, R4 and R5 will be similarly moved forward in time. In Figure 6, we plot the percent of time the processor is busy during the interval between when an I/O request is issued and the most recent completion of

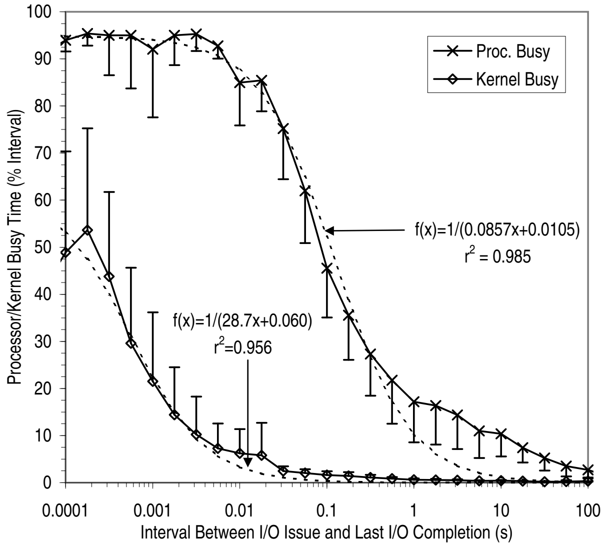


Figure 6: Processor/Kernel Busy Time during Intervals between Issuance of I/Os and Most Recent Request Completion. Bars indicate standard deviation (To reduce clutter, we show only the deviation in one direction).

an I/O request (the $x's$ in Figure 5). We are interested in the processor busy time during such intervals to model what happens when the processing time is reduced through faster processors.

From Figure 6, we find that for the PC workloads, the processor utilization during the intervals between 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 was in the traced system, we would scale only the system processing time by n, leaving the user think time unchanged. Specifically, we would replace an interval of length x by one of $x[1 - f(x) + f(x)/n]$. We believe that for the PC workloads, this is considerably more realistic than simply scaling the inter-arrival time between I/O requests by n, as is commonly done. In Figure 6, we also plot the percent of time that the kernel is busy during the intervals between when an I/O request is issued and the previous I/O completion. We consider the kernel to be busy if the kernel process (process ID = 2 in Windows NT) is allocated the CPU. 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.

This workload model is based on the assumption that I/Os tend to be synchronous, meaning that the system has to wait for I/Os to be completed before it can continue with its processing. As shown in Table 4, this is a reasonable assumption, especially for the PC workloads where, despite

	Read	Write	Overall
P1	0.974	0.667	0.887
P2	0.970	0.627	0.825
P3	0.931	0.701	0.770
P4	0.829	0.731	0.768
P5	0.927	0.776	0.814
P6	0.967	0.849	0.864
P7	0.878	0.723	0.758
P8	0.968	0.835	0.909
P9	0.800	0.605	0.699
P10	0.763	0.749	0.756
P11	0.926	0.705	0.805
P12	0.961	0.566	0.736
P13	0.610	0.695	0.664
P14	0.733	0.714	0.720
P-Avg.	0.874	0.710	0.784
FS1	0.854	0.254	0.505
TS1	0.835	0.671	0.744
DS1	-	-	-
S-Avg.	0.845	0.462	0.624

Table 4: Fraction of I/O Requests that are Synchronous.

			P-Avg.	Pf-Avg.	FS1	TS1	DS1	S-Avg.	Sf-Avg.
# I/O Requests	Average	Read	24.6	12.4	92.7	190	344	209	137
		Write	37.0	14.5	129	246	564	313	113
		Total	61.6	26.9	222	436	908	522	251
	Max.	Read	81.5	48.2	286	577	725	530	446
		Write	102	30.0	355	393	833	527	162
		Total	183	78	641	970	1558	1056	609
I/O Traffic (MB)	Average	Read	234	131	568	1152	3017	1579	1161
		Write	295	236	895	1604	2407	1635	1090
		Total	529	368	1462	2756	5425	3214	2250
	Max.	Read	973	701	1677	3613	4508	3266	2731
		Write	1084	856	2446	2573	5159	3393	2403
		Total	2057	1557	4124	6186	9667	6659	5134

Table 5: Daily I/O Traffic.

the fact that Windows NT provides a common convenient interface for performing both synchronous and asynchronous I/O, nearly 80% of the I/O requests are flagged as synchronous on average. Metadata updates account for most, but not all, of the synchronous writes. Excluding metadata writes, about half of the writes are synchronous. In the FS1 and TS1 traces, 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. The trace DS1 does not contain such information.

4.2 Amdahl's Factor and Access Density

Table 5 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 the disk blocks referenced for later analysis and optimization (*e.g., *to optimize disk block placement [22]) is very feasible.* For instance, for the database server workload, logging eight bytes of information per request will create just over 12MB 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 [3]. More specifically, Amdahls' rule of thumb states that a typical data processing system requires approximately 1Mb/s of I/O bandwidth for every million instructions per second (MIPS) of processing power. This rule of thumb dates back to the sixties 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 [20]. Due to the advent of caching, however, the ratio of physical I/O bandwidth to MIPS was found to be on the order of 0.05. [20] It would be interesting and very helpful to system designers to see if the same figure for Amdahl's factor applies to the current set of workloads.

To this end, we calculated the ratio of I/O intensity, *i.e.,* the rate of I/O activity, to processor speed for our workloads. Unlike the traces in [20] 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 includes the relatively idle periods in the workloads. Therefore, in calculating the I/O intensity normalized by processor speed in Table 6, 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 Table 7. Notice from Table 6 that 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 8), the bandwidth figures for the filtered traces are just slightly lower than those for the original workloads. Our focus in this section is on determining a rough estimate for how intense the I/O is in our various workloads. The effect of filtering the workloads is not large enough to significantly affect any of our findings.

From Table 6, the server workloads turn out to be 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 0.007Mb/s/MHz on average. In

	Avg. Number of Mbs of I/O				Avg. Number of I/Os			
	Per Second	/s /MHz	/s /MIPS	/s /GB	Per Second	/s /MHz	/s /MIPS	/s /GB
P1	0.588	0.00177	0.00177	0.0980	5.80	0.0174	0.0174	0.967
P2	0.557	0.00278	0.00278	0.116	7.25	0.0363	0.0363	1.51
P3	0.811	0.00180	0.00180	0.135	6.42	0.0143	0.0143	1.07
P4	6.84	0.0152	0.01520	1.14	61.0	0.135	0.135	10.2
P5	3.50	0.00778	0.00778	0.583	14.7	0.0326	0.0326	2.45
P6	0.106	0.000639	0.000639	0.0212	1.44	0.00866	0.00866	0.287
P7	2.84	0.0107	0.0107	0.711	28.5	0.107	0.107	7.13
P8	1.08	0.00361	0.00361	0.270	8.65	0.0288	0.0288	2.16
P9	1.11	0.00671	0.00671	0.371	15.4	0.0929	0.0929	5.14
P10	5.71	0.0215	0.0215	1.36	44.8	0.168	0.168	10.7
P11	0.852	0.00243	0.00243	0.213	10.9	0.0310	0.0310	2.72
P12	4.63	0.0116	0.0116	0.771	22.8	0.0570	0.0570	3.80
P13	0.385	0.00193	0.00193	0.0963	8.03	0.0401	0.0401	2.01
P14	4.14	0.00919	0.00919	0.517	51.8	0.115	0.115	6.47
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
FS1	1.26	0.0252	0.0503	0.419	26.8	0.536	1.07	8.94
TS1	1.99	0.0311	0.0621	0.191	39.0	0.610	1.22	3.75
DS1	6.11	0.0204	0.0407	0.117	72.4	0.241	0.482	1.38
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

Table 6: Intensity of I/O during the Busiest One-Hour Period.

	0.1s	1s	10s	1min	10min	1hr	Trace Len.
P1	115	55.3	27.3	8.63	2.13	0.588	0.0366
P2	64.8	26.8	20.9	7.54	2.35	0.557	0.0234
P3	50.3	27.2	15.9	13.1	4.19	0.811	0.0129
P4	121	99.5	80.0	56.1	34.6	6.84	0.0893
P5	40.2	28.0	26.3	17.6	13.2	3.50	0.0674
P6	45.0	23.3	8.51	2.81	0.44	0.106	0.00549
P7	61.3	47.1	18.5	10.4	4.78	2.84	0.0463
P8	51.9	36.4	19.8	11.8	3.60	1.08	0.0204
P9	50.0	27.0	11.1	5.99	3.71	1.11	0.0306
P10	85.0	75.0	48.5	34.9	17.1	5.71	0.0358
P11	133	46.4	29.0	12.7	2.06	0.852	0.0266
P12	90.0	48.7	26.2	20.1	10.7	4.63	0.0139
P13	45.0	21.5	7.77	4.39	1.26	0.385	0.0148
P14	71.6	51.5	32.5	29.0	12.4	4.14	0.0476
P-Avg.	73.1	43.8	26.6	16.8	8.04	2.37	0.0337
Pf-Avg.	109	45	24.0	15.0	6.66	1.92	0.0237
FS1	382	41.1	26.1	11.9	2.05	1.26	0.133
TS1	264	96.3	14.9	10.8	4.88	1.99	0.260
DS1	156	108	91.9	85.1	19.7	6.11	0.515
S-Avg.	267	81.7	44.3	35.9	8.87	3.12	0.302
Sf-Avg.	262	76	42.9	32.0	7.71	2.98	0.213

Table 7: I/O Intensity (Mb/s) Averaged over Various Time Intervals, showing the peak or maximum value observed for each interval size.

order to determine an order of magnitude figure for the ratio of I/O bandwidth 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 [6]. For the server workloads, we use a CPI value of two in view of results in [2, 25]. Based on this estimate 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 [20]. *The figure for the PC workloads is seven times lower at about 0.007 bits of I/O per instruction.*

Interestingly, surveys of large data processing mainframe installations between 1980 and 1993 [33] found the number of physical I/Os per second per MIPS to be decreasing by just over 10% per year to 9.0 in 1993. 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 this turned out not to be true. Data reported in [33] show that the average I/O request size for the surveyed mainframe installations was about 9KB, which is just slightly larger than the 8KB for our server workloads (Table 8). 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 the inconsistent methods used to calculate MIPS. The mainframe surveys used utilized MIPS [30] or the processing power actually consumed by the workload. This is computed by factoring in the processor utilization when the workload is running. Our calculations are based on the MIPS rating of the system, which is what we have available to us. Ultimately though, we believe that intrinsic workload differences account for a major portion of the discrepancy between our results and those from the mainframe surveys.

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 6 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 [33]. The survey of large data processing mainframe installations cited above found the access density to be decreasing by about 10% per year to 2.1 I/Os per second per GB of storage in 1993. Notice from Table 6 that the access density for DS1 appears to be consistent with the mainframe survey results. However, the access density for FS1 and TS1 is about two to four times higher. The PC workloads have, on average, an access density of 4 I/Os per second per GB of storage, which is on the order of

	All Requests				Read Requests				Write Requests			
	Avg.	Std. Dev.	Min.	Max.	Avg.	Std. Dev.	Min.	Max.	Avg.	Std. Dev.	Min.	Max.
P1	26	32.9	1	128	20.4	22.9	1	128	42.3	48.5	1	128
P2	19.7	30	1	1536	16.5	20.7	1	128	44.3	63.1	1	1536
P3	32.3	43.5	1	128	18.6	28.5	1	128	38.9	47.7	1	128
P4	28.7	40.2	1	128	15.5	21.1	1	128	29.8	41.2	1	128
P5	61	58.9	1	129	96.4	53.1	1	129	13	18.7	1	128
P6	18.9	29.4	1	128	27.6	29	1	128	16.8	29.1	1	128
P7	25.5	36.1	1	128	21.9	25	1	128	27.4	40.5	1	128
P8	32	42.6	1	128	21.2	31.5	1	128	55.3	52.9	1	128
P9	18.5	27.7	1	128	19.3	27.9	1	128	16	26.7	1	128
P10	32.6	44.4	1	128	37	46.4	1	128	23.1	37.8	1	128
P11	20.1	29	1	128	21.6	27.3	1	128	17.4	31.5	1	128
P12	51.9	120	1	512	96.4	144	1	512	38.2	107	1	512
P13	12.3	18.8	1	128	14.6	20.9	1	128	10.5	16.7	1	128
P14	20.5	38.6	1	128	13.7	29.1	1	128	72	58.3	1	128
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
FS1	12	5.52	2	18	11.6	5.61	2	18	14.4	4.15	2	16
TS1	13	9.87	2	512	12.6	5.52	2	64	14.9	19.9	2	512
DS1	21.6	35.3	1	128	25.4	38.6	1	108	19	32.5	1	128
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

Table 8: Request Size (Number of 512-Byte Blocks) during the Busiest One-Hour Period.

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

Table 9: Projected Processing Power and Storage Needed to Drive Various Types of I/O Interconnect to 50% Utilization.

the figure for the server workloads even though the server workloads are several years older. Such results suggest 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 6 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 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 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% utilization. The results are summarized in Table 9. Note that all the modern I/O interconnects offer Gb/s bandwidth. Some of them, specifically

Inter-Arrival Time (s)	P-Avg.	Pf-Avg.	FS1	TS1	DS1	S-Avg.	Sf-Avg.
1st Moment	3.25	7.23	0.398	0.194	0.0903	0.227	0.561
2nd Moment	7.79E+05	1.86E+06	368	23.1	2.00	131	363
3rd Moment	6.46E+11	1.60E+12	2.02E+07	8.09E+03	67.4	6.74E+06	1.88E+07

Table 10: First, Second and Third Moments of the I/O Inter-Arrival Time.

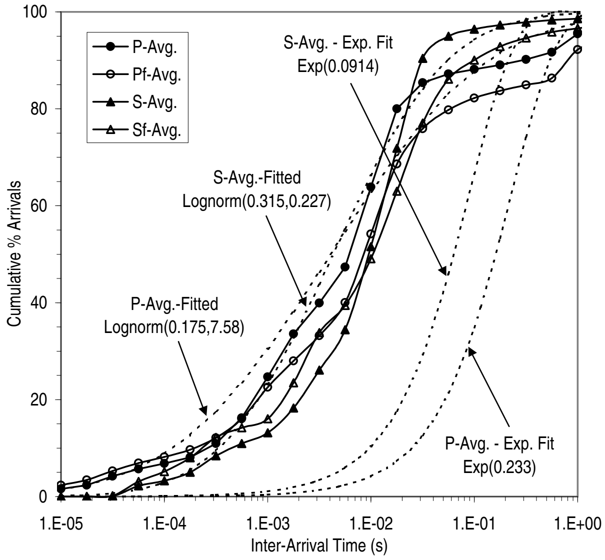


Figure 7: Distribution of I/O Inter-Arrival Time.

ethernet and fiber channel, have newer versions with even higher data rates. For the kinds of workloads that we have, 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 [21]). 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 connection between the disks and the host does not become the bottleneck.

4.3 Request Arrival Rate

In Table 10, we present the first, second and third moments of the inter-arrival time distribution. The distribution is plotted in Figure 7. Since the distribution of I/O inter-arrival time 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 $Lognorm(\mu, \sigma)$ where μ and σ are respectively the mean and standard deviation) is a reasonably*

	# 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.
P1	0.377	1.73	0.937	1	24	0.273	1.66	0.802	1	23	0.104	1.36	0.407	0	10
P2	0.421	1.9	1.01	2	13	0.28	1.93	0.864	1	12	0.141	1.45	0.473	0	6
P3	0.553	2.52	1.51	2	20	0.177	2.34	0.856	0	14	0.376	2.41	1.23	1	20
P4	0.796	2.67	1.96	3	74	0.332	2.15	1.1	1	27	0.464	2.33	1.55	1	74
P5	0.304	1.92	0.958	1	22	0.0985	1.97	0.601	0	20	0.206	1.71	0.704	1	22
P6	0.27	1.52	0.684	1	10	0.0169	1.36	0.181	0	8	0.253	1.52	0.66	1	8
P7	0.47	2.09	1.26	2	55	0.139	1.92	0.766	0	54	0.331	1.91	0.967	1	22
P8	0.365	1.96	1.07	1	26	0.196	1.65	0.699	1	14	0.168	1.82	0.673	0	16
P9	0.718	2.77	2.41	2	73	0.233	1.72	0.837	1	24	0.484	3.3	2.27	1	73
P10	0.573	2.33	1.81	2	60	0.252	1.62	0.766	1	19	0.321	2.53	1.62	1	60
P11	0.454	2.22	1.29	1	37	0.251	2.06	0.948	1	17	0.204	1.73	0.728	1	35
P12	0.341	1.99	1.06	1	19	0.201	2.37	0.897	0	17	0.14	1.35	0.464	1	8
P13	0.664	2.26	1.47	2	24	0.393	2.33	1.17	1	17	0.272	1.7	0.859	1	24
P14	0.541	2.11	1.28	2	23	0.184	1.62	0.677	1	17	0.358	1.98	1.05	1	23
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
FS1	1.49	4.19	4.62	3	181	0.186	1.38	0.538	1	13	1.3	4.74	4.56	3	181
TS1	9.98	27.2	41.1	12	1530	0.214	1.42	0.574	1	20	9.76	36.4	41.1	11	1530
DS1	3.13	8.68	15.9	5	257	0.203	1.95	0.904	1	9	2.93	8.93	15.7	5	256
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

Table 11: Queue Depth on Arrival.

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 towards the larger values, meaning that there exists long intervals with no I/O arrivals. The long tail of the inter-arrival 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 intervals that have far more arrivals than their 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 8 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 11 and Figure A-2 in Appendix A, 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% 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 11). In addition, the maximum

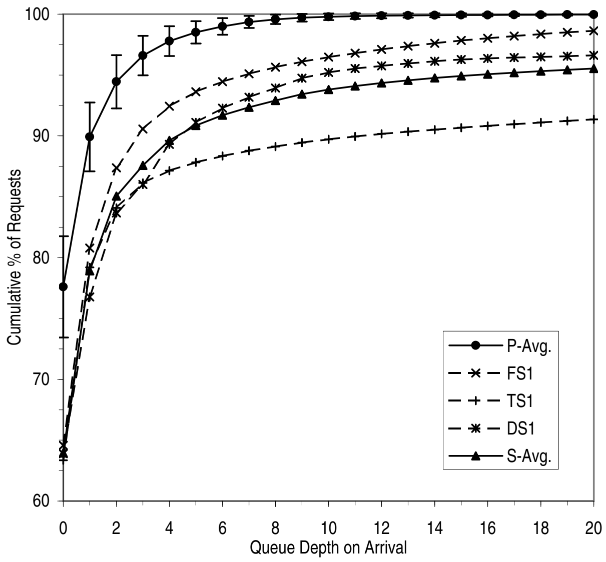


Figure 8: Distribution of Queue Depth on Arrival. Bars indicate standard deviation.

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 will examine this in greater detail in [18]. Observe further from Table 11 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 4). The PC workloads appear to have a significantly shallower write queue than the other workloads but in most cases, there are still enough outstanding write requests to benefit from request scheduling.

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, which is typically not handed hundreds of requests at a time. Some of the differences among the workloads could be the result of collecting the traces at different levels on the different platforms. In general, the operating system and/or the disk device driver will queue up the requests and attempt to schedule them based on some simple performance model of the storage system (*e.g.,* minimize seek distance). There is a tendency for the operating system and/or device driver to hold back the requests and issue only a small number of them at any one time so as to avoid overloading the storage system. In reality, modern storage systems, specifically modern disks, have the ability to do more elaborate and effective [54] request scheduling based on whether a request will hit in the disk cache, and on the seek and rotational positions.

5 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 (Section 4.3), 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 will tend to cancel out one another. 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 will examine the dependence among our workloads in Section 5.1.

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 [9, 50])

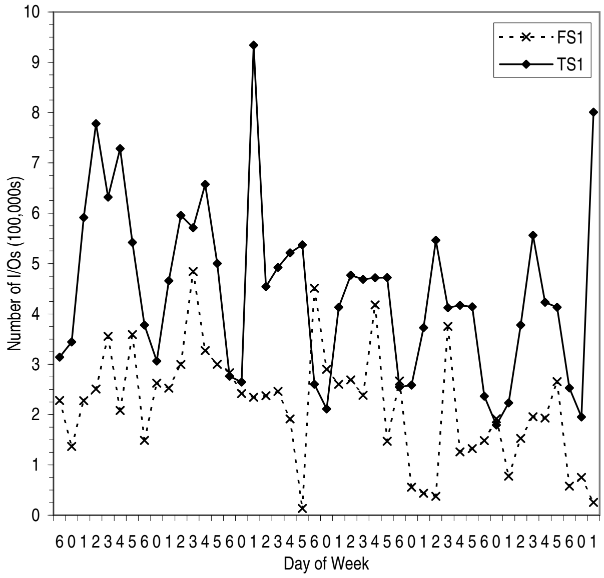


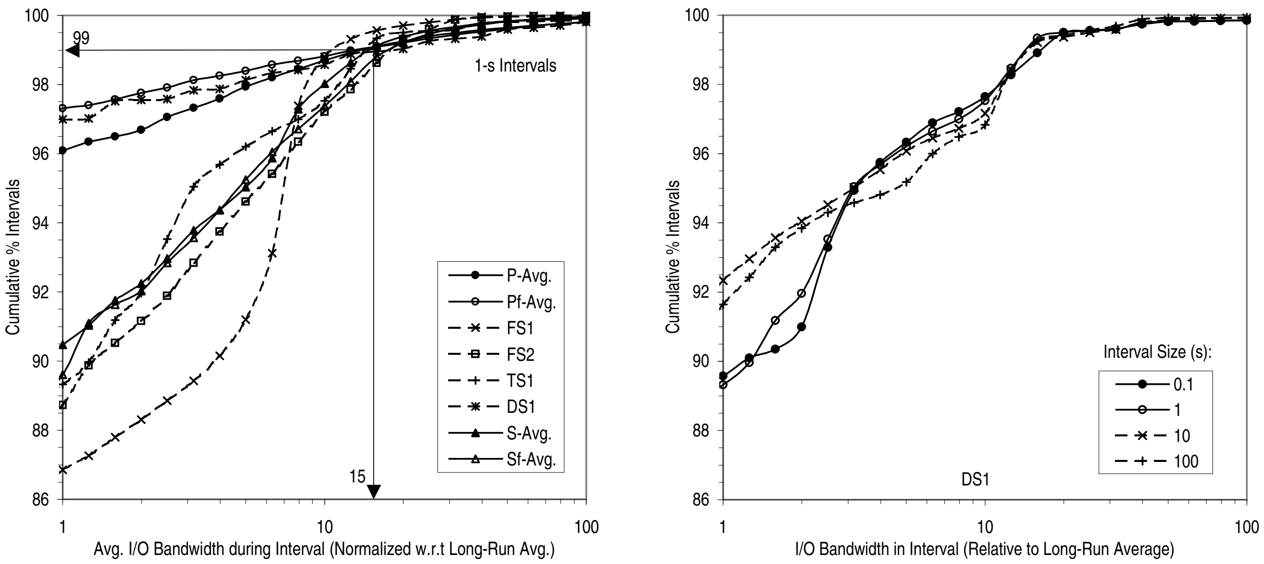
Figure 9: Daily Volume of I/O Activity.

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 [22]). 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 will analyze the autocorrelation of I/O traffic to determine whether they are long-range dependent or self-similar in Section 5.2. In Section 5.3, we characterize in detail the idle periods to help in the design of mechanisms that try to exploit idle time.

5.1 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 Figure 9, we plot the daily volume of I/O activity for FS1 and TS1 as a function of the day of week (0 = Sunday). If the two workloads are positively correlated, we should see the peaks in the two workloads appearing on the same day 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 should be no relation between the volume of activity for 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 X(i) and Y(i) where i=0,1,2...n-1 is defined as



(a) (b)

Figure 10: Distribution of I/O Traffic Averaged over Various Time Intervals.

$$r_{XY} = \frac{\sum_i (X(i) - \overline{X})(Y(i) - \overline{Y})}{\sqrt{\sum_i (X(i) - \overline{X})^2}\sqrt{\sum_i (Y(i) - \overline{Y})^2}} \tag{1}$$

The possible values of R_{XY} range from -1 to 1 with -1 indicating perfect negative correlation between the two workloads, 0 no correlation, and 1 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 Tables A-1 - A-4 in Appendix A.

To summarize the dependence among a set of workloads W, we define the average cross-correlation as $\overline{r_{XY}}$ where $X \in W$, $Y \in W$ and $X \neq Y$. In Figure A-3, we plot the average cross-correlation for the PC workloads as a function of the time interval used to aggregate the arrival process. In the same figure, we also plot 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 working 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.

5.2 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. In Section 4.2, we computed the access density and found that the server workloads average about 5 I/Os or about 30KB 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 10(a) plots the distribution of I/O traffic averaged over one-second intervals and normalized to the average bandwidth over the entire trace. The plot shows that *to satisfy the bandwidth requirement for 99% of the 1-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% of the intervals, which means that *to satisfy requirements beyond 99% of the time will require disproportionately more resources.*

In analyzing the data, we noticed 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 10(b), the distributions for time intervals of 0.1s, 1s, 10s, 100s for the database server are very similar. This scale-invariant characteristic is apparent in Figure A-5, which shows the traffic variation over time for different time scales for the time sharing and database server workloads. The topmost plot shows the throughput averaged over time intervals of 0.3s. In the second plot, we zoom out by a factor of ten so that each data point is the average traffic volume over a three-second interval. The third plot zooms out further by a factor of ten. Observe that rescaling the time series does not smooth out the burstiness. Instead the three plots look similar. It turns out that for these workloads, such plots look similar for time scales ranging from tens of milliseconds to tens of seconds.

Many of the statistical methods used in this section assume that the arrival process is stationary. In order to avoid potential non-stationarity, 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% 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% of other one-hour periods in the trace.

5.2.1 Definition of Self-Similarity

The phenomenon where a certain property of an object is preserved with respect to scaling in space and/or time is described by self-similarity and fractals [31]. Let X be the incremental process of a process Y, *i.e.,* $X(i) = Y(i+1) - Y(i)$. In our case, 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

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

is the aggregated sequence obtained by dividing the original series into blocks of size m and averaging over each block, and k is the index that labels each block. 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 interested reader is referred to [5] for a more detailed treatment.

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. The interested reader is referred to Appendix B for details about how we estimate the degree

	P-Avg.	Pf-Avg.	FS1	FS2	TS1	DS1	S-Avg.	Sf-Avg.
H	0.81	0.79	0.88	0.92	0.91	0.91	0.90	0.80
μ (KB/s)	188	91.6	108	229	1000	445	367	1000
σ² (KB/s)²	769080	528538	122544	345964	1256360	627261	528439	1256360

Table 12: Hurst Parameter, Mean and Variance of the Per-Second Traffic Arrival Rate during the High-Traffic Period.

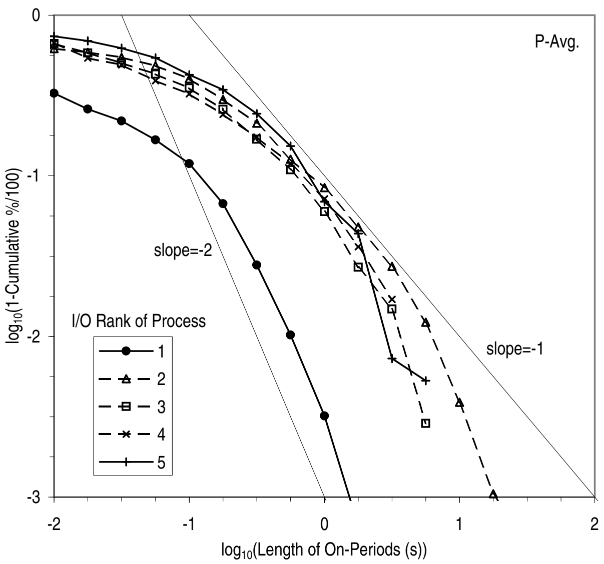
of self-similarity for our various workloads. Here, we simply summarize the Hurst parameter values we obtained (Table 12) and state the finding that *for time scales ranging from tens of milliseconds to tens and sometimes even hundreds of seconds, the I/O traffic is well-represented by a self-similar process.* Note that filtering the workloads does not affect the self-similar nature of their I/O traffic.

5.2.2 Implications of Self-Similar I/O Traffic

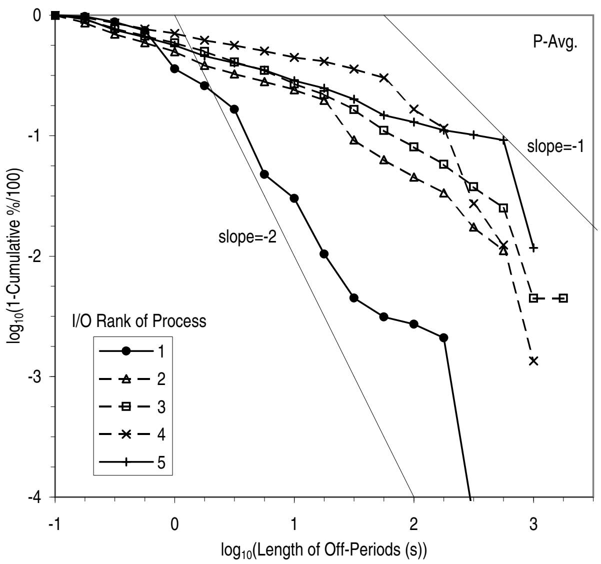
That the I/O traffic is self-similar implies that the burstiness exists over a wide range of time scales and that attempts at evening out the traffic temporally will tend to not remove all the variability. In addition, the I/O system may experience concentrated periods of congestion with associated increase in queuing time and that resource (*e.g.,* buffer, channel) 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 has to be taken into account when designing storage systems, especially when we wish to isolate multiple workloads 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 describing 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 12 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 [15] and I/O traffic [13] 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 than the Poisson model(*e.g.,* [27]). However, analytical modeling with self-similar inputs has not been well developed yet. (See [38] for some



(a) On-Periods.



(b) Off-Periods.

Figure 11: Length of On/Off Periods for the Five Most I/O-Active Processes.

recent results on modeling network traffic with self-similar processes). This, coupled with the complexity of storage systems today, means that most of the analysis has to be performed through simulations. Generating I/O traffic that is consistent with the self-similar characteristic observed in real workloads is therefore extremely important and useful. In Appendix B, we present a recipe that can be used to generate self-similar traffic using the parameters in Table 12.

5.2.3 Underpinnings of Self-Similar I/O Traffic

We have seen that I/O traffic 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 try to 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

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

Such a random variable can give rise to extremely large values with non-negligible 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[1] [53]. In this section, we break down each workload into the I/O traffic generated by the individual processes. As in [13], 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 on periods for the process. This analysis has been shown to be relatively insensitive to the threshold value used to distinguish the on and off periods [53].

Taking logarithm on both sides of Equation 3, we get

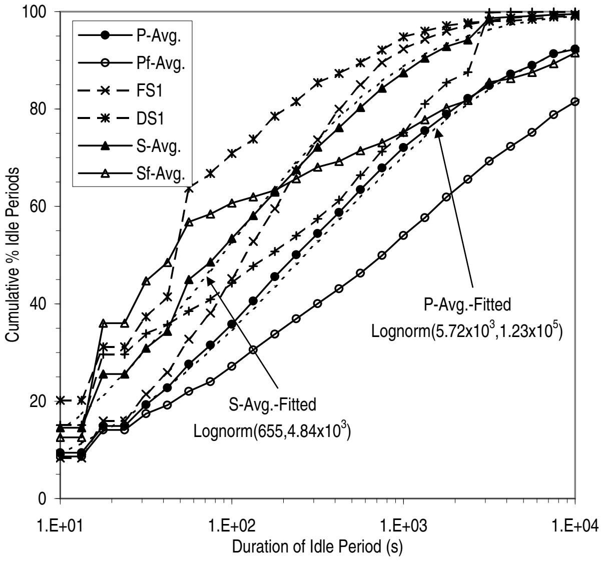
$$logP(X > x) \sim log(c) - \alpha log(x), as\ x \to \infty. \quad (4)$$

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" [26]. In Figure 11, 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 [13] 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.

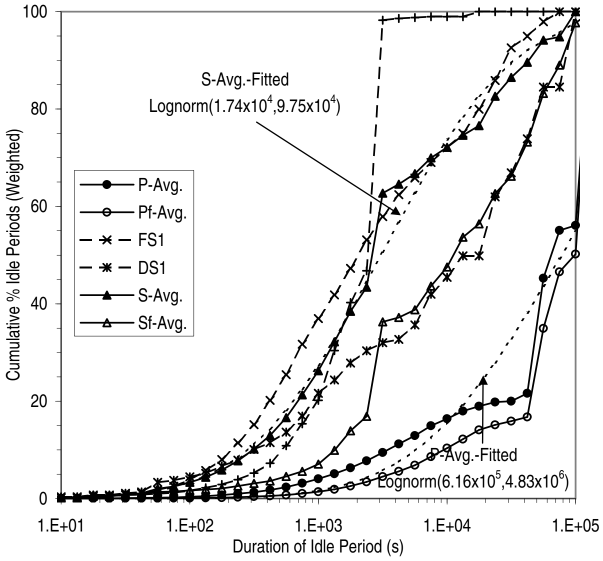
5.3 The Relative Lulls

As discussed earlier, when the I/O load is not constant but varies over time, there may be opportunities to use the

[1] Not Poisson; assumptions of Palm-Khintchine theorem are not satisfied.



(a)



(b)

Figure 12: Distribution of Idle Period Duration. For the weighted distribution in (b), an idle period of duration s is counted s times, *i.e.,* it is the distribution of idle time.

relatively idle periods to do some useful work. The reader is referred to [12] for an overview of idle-time processing and a general taxonomy of idle-time detection and prediction algorithms. Here, we characterize in detail the idle periods, focusing on specific metrics that will be helpful in designing mechanisms that try to exploit idle time.

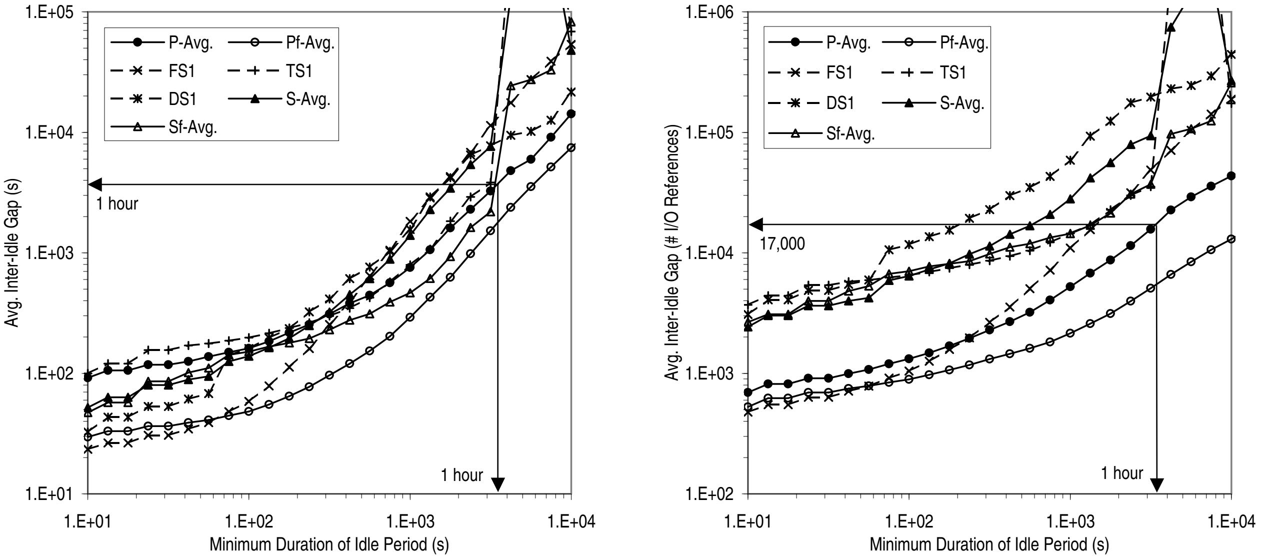
We consider an interval to be idle if the average number of I/Os per second during the interval is less than some value k. The term idle period refers to a sequence of intervals that are idle. The duration of an idle period is simply the product of the number of idle intervals it contains and the interval 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 [18], 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.

Figure 12 presents the distribution of idle period duration for our workloads. 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 hundred seconds), long idle periods account for most of the idle time. This is consistent with previous results and implies that *a system that exploits idle time can get most of the potential benefit by simply focusing on the long idle periods* [12].

5.3.1 Inter-idle Gap

An important consideration in attempting to make use of idle periods 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 also determines the effectiveness and even the feasibility of exploiting the idle periods. For instance, a log-structured file system or array [34, 44] 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 [22], the inter-idle gap, *i.e.,* the time span between suitably long idle periods, determines the amount of reference data that has to be accumulated on the disk.

In Figure 13, 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 17,000 references. These results indicate that *in the personal systems environment, there are long idle periods that occur frequently enough to be interesting for offline optimizations such as block reorganization* [22]. As we would expect, the server workloads have longer busy periods separated by shorter idle



(a) Measured in seconds.

(b) Measured in Number of I/O References.

Figure 13: Average Duration of Busy Periods.

periods. *In these server environments, we have to be more meticulous in using the available idle time. For instance, we may have to divide the task of analyzing the reference patterns to optimize block placement [22] into several finer-grained steps that can be scheduled whenever a short idle period presents itself.*

5.3.2 Idle Length Prediction

In some cases, there is a recovery cost associated with stopping an offline task before it is completed. Therefore, it is important to be able to predict how long an idle period will last so that the system can decide whether a task should be initiated. In Figure A-4 in Appendix A, we plot the autocorrelation of the sequence of idle period duration at different lags. For all the workloads, there is little correlation between the length of one idle period and the lengths of the immediately preceding periods. In other words, *how long the system will remain idle is not predictable from the lengths of its recent idle periods.* This is in stark contrast to the strong correlation that has previously been observed for a personal Unix workstation [12]. In that study, the idle period was taken to be an interval during which there was no I/O activity. 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 and hence the strong autocorrelation.

In Figure 14, 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,

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

where $l(\cdot)$ is the probability distribution of the idle period duration, *i.e.,* $l(j)$ 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, *i.e.,* $L(j) = \sum_{i=1}^{j} l(i)$. Observe from Figure 14 that E[I(x)] is generally increasing. In other 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 duration as the idle duration increases. 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 (Figure A-6 in Appendix A). The hazard rate is simply the probability that an idle period ends with a duration $\leq k + r$ 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 chance that a task initiated now and requiring r units of time will not be completed before the system becomes busy again. More formally,

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

We find that the hazard rate is generally declining as the

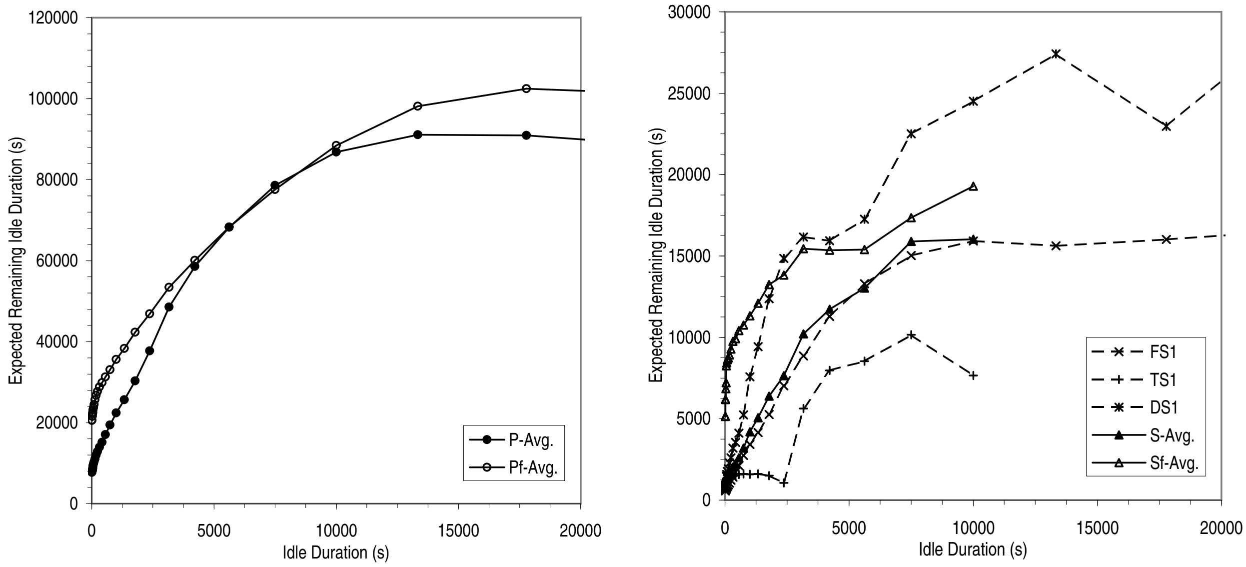


Figure 14: Remaining Idle Duration.

idle period duration increases. *This result again supports the idea of predicting idle period duration by conditioning on the amount of time the system has already been idle.* In addition, the hazard rate increases with r, which is what we would expect. In other words, the chances that a task will not be completed before the system becomes busy again increases with the length of the task.

6 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 is 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 is unlikely to be updated again but is likely to be read. The latter is useful when it is highly likely that the data will be updated several more times before it is read. In cases where the data is being both updated and read, replication may not be useful. Thus the read-write composition of the traffic, together with the flow of data from writes to reads, is an extremely important workload characteristic. This is the focus of this section.

6.1 Read/Write Ratio

A wide range of read/write ratio has been reported in the literature. In addition to intrinsic workload differences, the read-to-write ratio also depends a lot on how much of the reads and writes have been filtered by caching, and on the kinds of I/Os (*e.g.,* user data, paging, file system metadata) 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 30s) flushing the dirty blocks in the file cache to disk so as to limit the amount of data that will 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 [46]. Therefore, more of the reads than writes are filtered by the file system cache. The file system also adds metadata writes which can account for more than half of the physical writes (more than 72% in [45] and more than 53% in our PC workloads). Therefore, 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 desktop workstation workloads [42], and the ratio of logical read to write operations has been found to be between 3 and 4.5 in various office environments [41]. At the physical level, the read/write ratio has been observed to range from about 0.4 to 1 for Novell NetWare file servers [17] and from about 0.7 to 0.8 for several HP-UX systems [45]. These figures are comparable to the read/write ratio we obtained, which are presented in Table 13. Observe that *consistently across all the server workloads and the PC workloads, on average, writes account for about 60% 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 [33]. Analysis

	Requests # Read Requests / # Write Requests	Traffic MB Read / MB Written	Footprint # Unique Blocks Read / # Unique Blocks Written
P1	2.51	1.99	1.05
P2	1.37	1.79	1.55
P3	0.429	0.430	0.563
P4	0.606	0.550	0.585
P5	0.338	0.475	1.02
P6	0.147	0.231	0.322
P7	0.288	0.299	0.399
P8	1.23	1.14	0.941
P9	0.925	1.02	1.38
P10	0.937	1.41	2.17
P11	0.831	1.38	0.787
P12	0.758	0.883	0.904
P13	0.566	0.744	1.40
P14	0.481	0.710	0.770
P-Avg.	0.816	0.932	0.988
Pf-Avg.	0.965	0.607	0.888
FS1	0.718	0.633	1.50
TS1	0.794	0.740	1.15
DS1	0.607	1.24	1.06
S-Avg.	0.706	0.870	1.24
Sf-Avg.	1.12	0.843	1.19

Table 13: Read/Write Ratio.

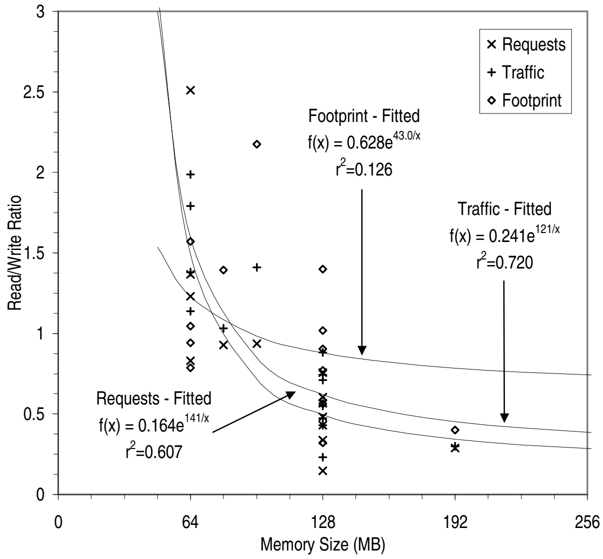
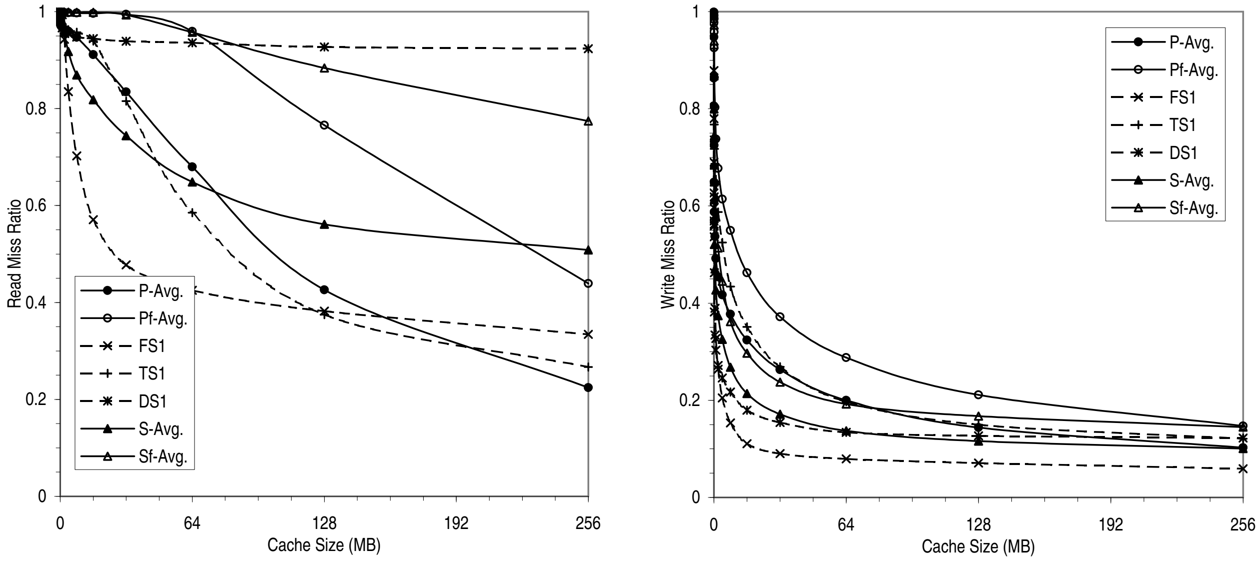


Figure 15: Read/Write Ratio as Function of Memory Size.

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 about 10 [20, 21].

Observe from Table 13 that for the PC workloads, the read/write ratio does appear 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 15, 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, *i.e.,* as $x \to \infty$, there will be about 6 writes for every read. Such results support to 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 [36, 37, 44]. However, that the read ratio remains relatively consistent across all our workloads, which span a time period of eight years, suggests that workload changes may have a counter effect. 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 16, 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 are not filtered 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. Note that 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 exhibit 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 non-volatile memory, write requests can similarly be very significantly reduced. In fact, for practically all the workloads, a small cache of 1MB 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. In addition, the effective latency of writes can often be reduced by writing data asynchronously or in a log [44, 52] or by using write-ahead logging [35]. Recent results (*e.g.,* [10]) also 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.*



(a) Read Miss Ratio.

(b) Write Miss Ratio.

Figure 16: Miss Ratio with LRU Write-Back Cache (512B Blocks).

6.2 Working Set Overlap

The working set $W(t, \tau)$ is defined as the set of blocks referenced within the last τ units of time [11]. More formally,

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

where $Count(b, t - \tau, t)$ denotes the number of times block b is referenced between $t - \tau$ and t. In Figure 17, 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, τ=1 day). To understand the interaction between reads and writes, we differentiate the blocks referenced into those that are read, written, and both read and written. Specifically,

$$W_{read}(t, \tau) = \{b : ReadCount(b, t - \tau, t) >= 1\} \tag{8}$$

$$W_{written}(t, \tau) = \{b : WriteCount(b, t - \tau, t) >= 1\} \tag{9}$$

$$W_{both}(t, \tau) = W_{read}(t, \tau) \cap W_{written}(t, \tau) \tag{10}$$

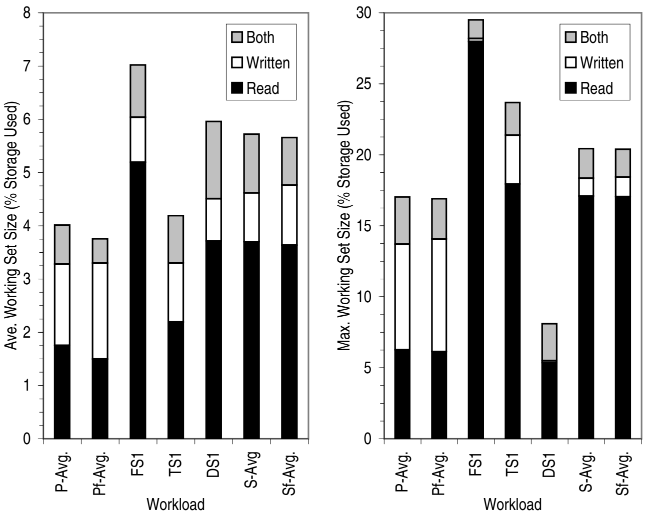


Figure 17: Daily Working Set Size.

Observe that on average, the daily working set for the various workloads range from just over 4% (PC workloads) to about 7% 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 can be several times larger than the average. Notice further from Figure 17 that the read working

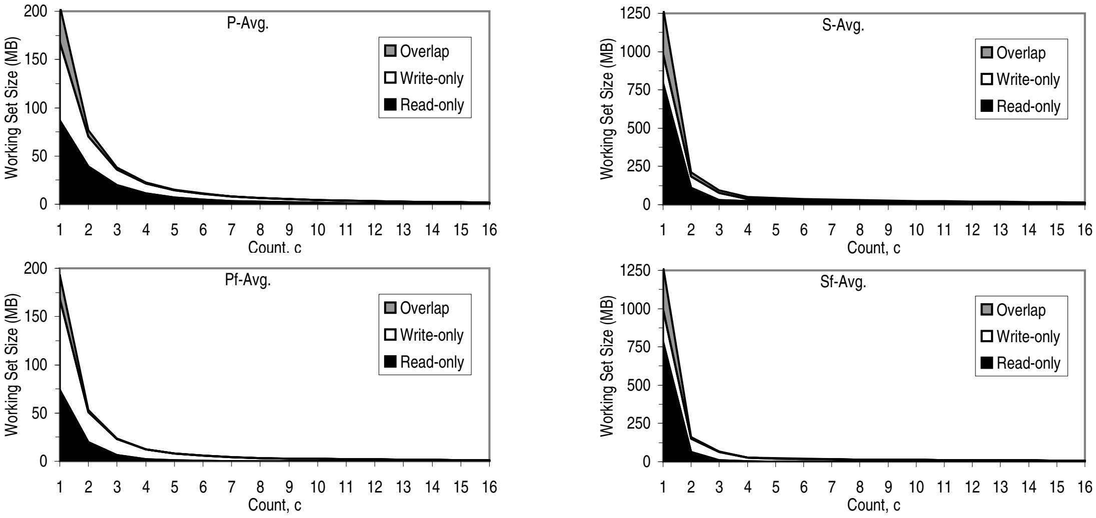


Figure 18: Average Daily Generalized Working Set Size.

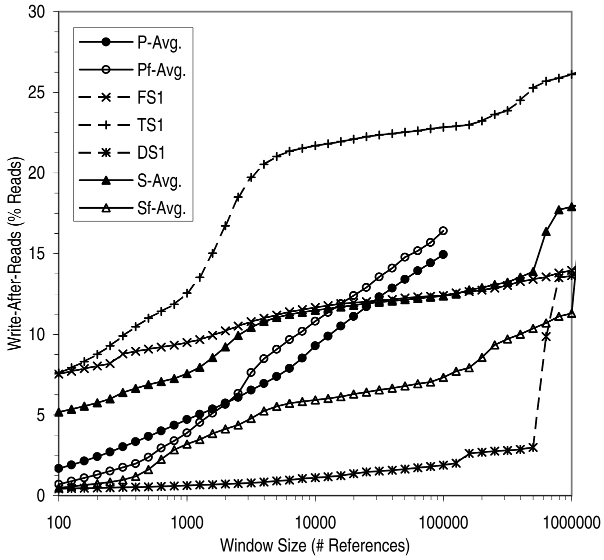


Figure 19: Write after Read (WAR).

set is larger than the write working set for all the workloads, especially for the server workloads. In addition, the working set of blocks that are both read and written is small, representing less than 25% 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, 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 [11] is simply the special case where $c = 1$. Figure 18 presents the average daily generalized working set size for our workloads. The figure shows that for all the workloads, *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 [22].* Notice also that *the amount of data that is both actively read and updated is clearly very small.* We will examine this further by looking at the dependencies between reads and writes in the next section.

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

In Figure 19, we plot the percentage of reads for which there is a write within τ references that constitute a WAR. We refer to τ as the window size. Observe that even for a large window size of 100,000 references, less than 25% 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 reorganized or replicated based on their read access patterns, write performance will not be significantly affected.* Notice from Figures 20 and 21 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. Therefore, 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 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.

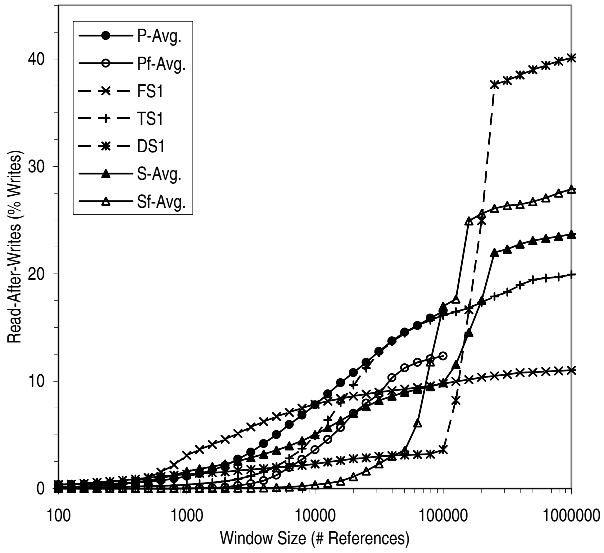


Figure 20: Read after Write (RAW).

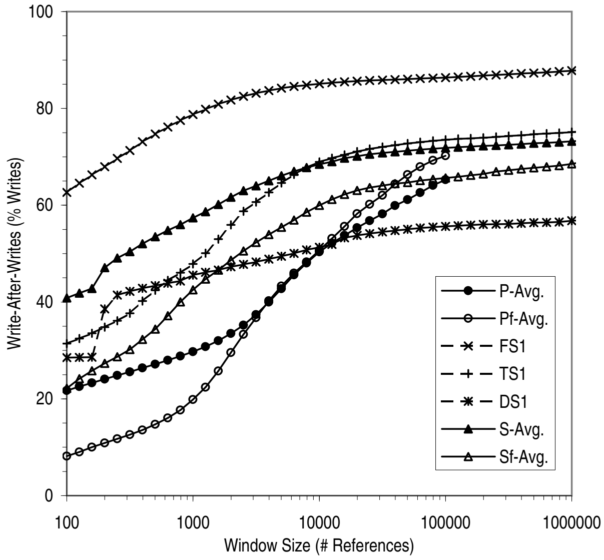


Figure 21: Write after Write (WAW).

7 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 include:

- 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% 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 by using non-volatile memory, write requests can similarly be very significantly reduced. In fact, for practically all the workloads, a small write-back cache of 1MB eliminates more than half the writes. 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-examined Amdahl's rule of thumb for a balanced system and found 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 of the world's largest corporations [20]. The figure for the PC workloads is seven times lower at about 0.007 bits of physical I/O per instruction.

- 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 also show that to satisfy I/O bandwidth requirements 99% of the time, we will need to provision for 15 times the long-run average bandwidth. Going beyond 99% of the time will 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 the personal systems environment, there are idle periods that are both long enough and that occur frequently enough to be interesting for offline optimizations such as block reorganization. In the server environment, we 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 a short idle period presents itself. Our results suggest that the length of an idle period can be predicted more accurately by conditioning 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 for it to be feasible to collect a daily trace of the blocks referenced for later analysis and optimization. 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.

8 Acknowledgements

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.

References

[1] P. Abry and D. Veitch, "Wavelet analysis of long-range-dependent traffic," *IEEE Transactions on Information Theory*, 44, 1, pp. 2–15, 1998.

[2] 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), Sept. 1999.

[3] G. M. Amdahl, "Storage and I/O parameters and systems potential," *Proceedings of IEEE International Computer Group Conference (Memories, Terminals, and Peripherals)*, (Washington, DC), pp. 371–372, June 1970.

[4] 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), pp. 198–212, Oct. 1991.

[5] J. Beran, *Statistics for Long-Memory Processes*, Chapman & Hall, New York, NY, 1994.

[6] D. Bhandarkar and J. Ding, "Performance characterization of the Pentium Pro processor," *Proceedings of Third International Symposium on High-Performance Computer Architecture (HPCA)*, (San Antonio, Texas), pp. 288–297, Feb. 1997.

[7] 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), pp. 13–23, May 1993.

[8] G. P. Bozman, H. H. Ghannad, and E. D. Weinberger, "A trace-driven study of CMS file references," *IBM Journal of Research and Development*, 35, 5/6, pp. 815–828, Sept./Nov. 1991.

[9] 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), Aug. 2000.

[10] M. Dahlin, *Serverless Network File Systems*, PhD thesis, University of California, Berkeley, Dec. 1995.

[11] P. J. Denning, "The working set model for program behaviour," *Communications of the ACM*, 11, 5, May 1968.

[12] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes, "Idleness is not sloth," *Proceedings of USENIX Technical Conference*, (New Orleans, LA), pp. 201–212, Jan. 1995. Extended version available as Technical Report HPL-96-140, HP Laboratories, Palo Alto, CA, Oct. 1996.

[13] M. E. Gómez and V. Santonja, "Analysis of self-similarity in I/O workload using structural modeling," *Proceedings of 7th Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, (College Park, MD), pp. 234–242, Oct. 1999.

[14] J. Gray, "Put EVERYTHING in the storage device." Talk at NASD Workshop on Storage Embedded Computing, June 1998. http://www.nsic.org/nasd/1998-jun/gray.pdf.

[15] 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), pp. 141–150, June 1998.

[16] E. Grochowski, "IBM leadership in disk storage technology," 2000. http://www.storage.ibm.com/technolo/grochows/grocho01.htm.

[17] 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), pp. 313–322, Dec. 1995.

[18] 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, 2002. In preparation.

[19] 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 Seventh International Conference on Parallel and Distributed Systems (ICPADS)*, (Iwate, Japan), pp. 417–425, July 2000.

[20] W. W. Hsu, A. J. Smith, and H. C. Young, “Analysis of the characteristics of production database workloads and comparison with the TPC benchmarks,” *IBM Systems Journal*, 40, 3, 2001.
[21] 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, 1, Mar. 2001.
[22] W. W. Hsu, A. J. Smith, and H. C. Young, “The automatic improvement of locality in storage systems.” Research Report, IBM Almaden Research Center, San Jose, CA, 2002. In preparation.
[23] IBM Corporation, *AIX Versions 3.2 and 4 Performance Tuning Guide*, 5th ed., Apr. 1996.
[24] R. Karedla, J. S. Love, and B. G. Wherry, “Caching strategies to improve disk system performance,” *Computer*, 27, 3, pp. 38–46, Mar. 1994.
[25] K. Keeton, D. Patterson, Y. He, R. Raphael, and W. Baker, “Performance characterization of a quad Pentium Pro SMP using OLTP workloads,” *Proceedings of ACM SIGARCH International Symposium on Computer Architecture (ISCA)*, (Barcelona, Spain), pp. 15–26, June 1998.
[26] M. Kratz and S. Resnick, “The QQ - estimator and heavy tails,” *Stochastic Models*, 12, pp. 699–724, 1996.
[27] 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, pp. 1–15, Feb. 1994.
[28] F. Les, “Outsourced network storage,” *PC Magazine (What’s In Storage for You?)*, Mar. 2001. http://www.zdnet.com/pcmag/stories /reviews/0,6755,2692092,00.html.
[29] J. R. Lorch and A. J. Smith, “The VTrace tool: Building a system tracer for Windows NT and Windows 2000,” *MSDN Magazine*, 15, 10, pp. 86–102, Oct. 2000.
[30] J. B. Major, “Processor, I/O path, and DASD configuration capacity,” *IBM Systems Journal*, 20, 1, pp. 63–85, 1981.
[31] B. B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freeman, New York, 1982.
[32] P. McDougall, “The power of peer-to-peer,” *Information Week*, Aug. 2000. http://www.informationweek.com/801/peer.htm.
[33] B. McNutt, “MVS DASD survey: Results and trends,” *Proceedings of Computer Measurement Group (CMG) Conference*, (Nashville, TN), pp. 658–667, Dec. 1995.
[34] J. Menon, “A performance comparison of RAID-5 and log-structured arrays,” *Proceedings of IEEE International Symposium on High Performance Distributed Computing*, (Washington, DC), pp. 167–178, Aug. 1995.
[35] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, “ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging,” *ACM Transactions on Database Systems*, 17, 1, pp. 94–162, Mar. 1992.
[36] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson, “A trace-driven analysis of the UNIX 4.2 BSD file system,” *Proceedings of ACM Symposium on Operating System Principles (SOSP)*, (Orcas Island, WA), pp. 15–24, Dec. 1985.
[37] J. Ousterhout and F. Douglis, “Beating the I/O bottleneck: A case for log-structured file systems,” *Operating Systems Review*, 23, 1, pp. 11–28, Jan. 1989.
[38] K. Park and W. Willinger, eds., *Self-Similar Network Traffic and Performance Evaluation*, John Wiley and Sons Inc., New York, 2000.
[39] D. A. Patterson and K. K. Keeton, “Hardware technology trends and database opportunities.” Keynote speech at *SIGMOD’98*, June 1998. Slides available at http://www.cs.berkeley.edu/~pattrsn /talks/sigmod98-keynote-color.ppt.
[40] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, 1990.
[41] 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), pp. 78–90, June 1992.
[42] D. Roselli, J. R. Lorch, and T. E. Anderson, “A comparison of file system workloads,” *Proceedings of USENIX Annual Technical Conference*, (Berkeley, CA), pp. 41–54, June 2000.
[43] 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), pp. 285–298, Dec. 1995.
[44] M. Rosenblum and J. K. Ousterhout, “The design and implementation of a log-structured file system,” *ACM Transactions on Computer Systems*, 10, 1, pp. 26–52, 1992.
[45] C. Ruemmler and J. Wilkes, “UNIX disk access patterns,” *Proceedings of USENIX Winter Conference*, (San Diego, CA), pp. 405–420, Jan. 1993.
[46] M. Russinovich, “Inside the cache manager,” *Windows & .NET Magazine*, Oct. 1998.
[47] B. Ryu and M. Nandikesan, “Real–time generation of fractal atm traffic: Model.” Technical Report 440-96-06, Center for Telecommunicaions Research, Columbia University, New York, Mar. 1996.
[48] D. Sacks, “Demystifying DAS, SAN, NAS, NAS gateways, Fibre Channel, and iSCSI.” http://www.storage.ibm.com/ibmsan/solutions/ infrastructure/pdf/sto_net.pdf, Mar. 2001.
[49] A. J. Smith, “Disk cache — miss ratio analysis and design considerations,” *ACM Transactions on Computer Systems*, 3, 3, pp. 161–203, Aug. 1985.
[50] D. Stodolsky, M. Holland, W. V. Courtright II, and G. A. Gibson, “Parity-logging disk arrays,” *ACM Transactions on Computer Systems*, 12, 3, pp. 206–235, Aug. 1994.
[51] W. Vogels, “File system usage in Windows NT 4.0,” *Proceedings of 17th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 93–109, Dec. 1999.
[52] R. Y. Wang, T. E. Anderson, and D. A. Patterson, “Virtual log based file system for a programmable disk,” *Proceedings of USENIX Operating Systems Design and Implementation (OSDI)*, (New Orleans, LA), pp. 29–43, Feb. 1999.
[53] 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, 1, pp. 71–86, Feb. 1997.
[54] B. L. Worthington, G. R. Ganger, and Y. N. Patt, “Scheduling algorithms for modern disk drives,” *Proceedings of ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, (Nashville, TN), pp. 241–251, May 1994.

Appendix A

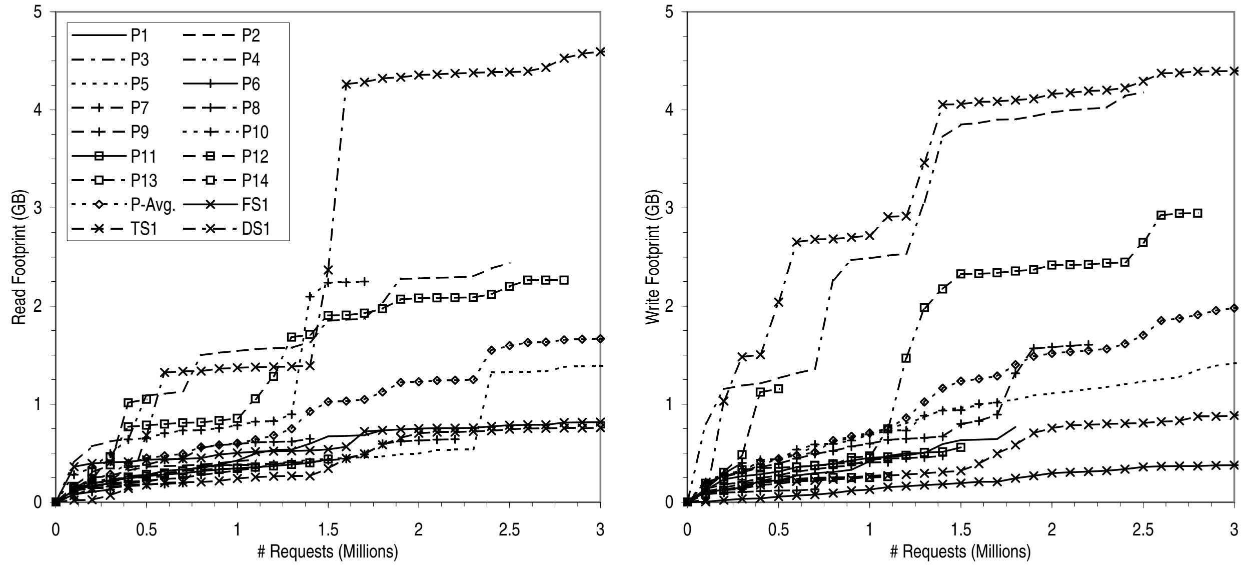
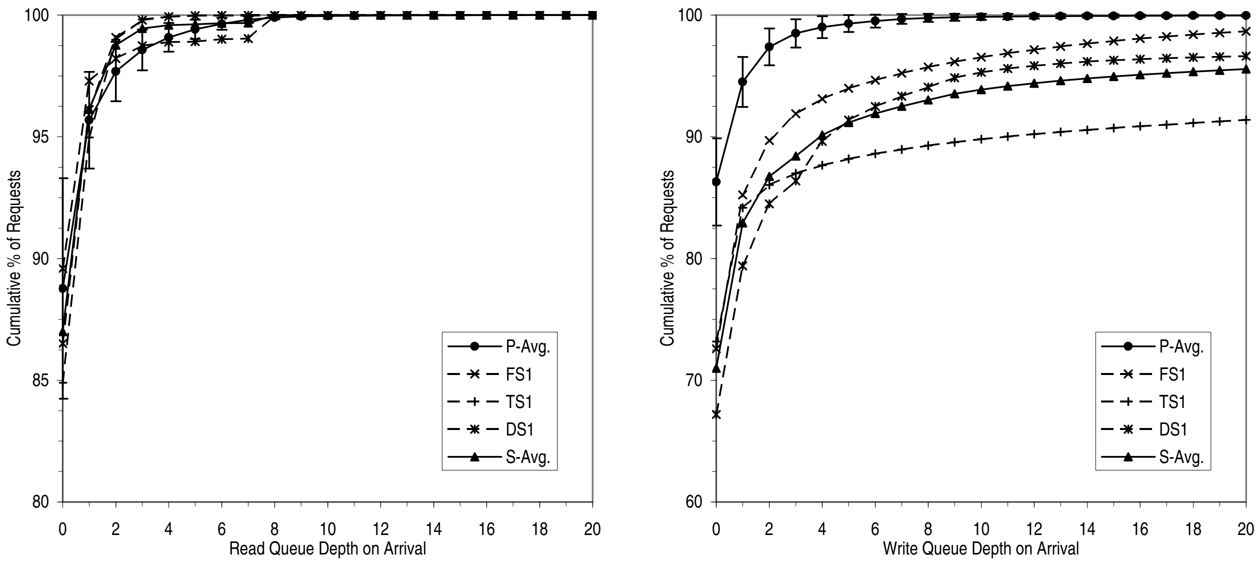


Figure A-1: Footprint Vs. Number of References.



(a) Number of Outstanding Read Requests.

(b) Number of Outstanding Write Requests.

Figure A-2: Average Queue Depth on Arrival. Bars indicate standard deviation.

Average

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	Avg.[i]	FS1	TS1	DS1	Avg.[ii]
P1	1	0.0692	0.0459	0.0784	0.0773	0.027	0.097	0.0798	0.074	0.0393	0.0419	0.0329	0.0262	0.0368	0.0558	0.00962	0.0219	0.00331	-
P2	0.0692	1	0.0244	0.0533	0.0759	0.0251	0.0834	0.0423	0.159	0.0195	0.0285	0.0116	0.0283	0.0544	0.0519	0.00249	0.046	0.00056	-
P3	0.0459	0.0244	1	0.0115	0.0518	0.0263	0.0324	0.0272	0.0371	0.0428	0.0487	0.0132	0.0192	0.0447	0.0327	0.0285	0.0192	0.0175	-
P4	0.0784	0.0533	0.0115	1	0.0399	0.0262	0.0496	0.0484	0.0994	0.0278	0.0593	0.0109	0.0742	0.0446	0.0480	0.0247	0.0376	0.0144	-
P5	0.0773	0.0759	0.0518	0.0399	1	0.0342	0.0939	0.0512	0.0765	0.0281	0.0349	0.0118	0.048	0.04	0.0510	0.021	0.0263	0.00529	-
P6	0.027	0.0251	0.0263	0.0262	0.0342	1	0.0673	0.0333	0.0615	0.0538	0.0352	0.0299	0.0434	0.0528	0.0397	0.0197	0.0201	0.0331	-
P7	0.097	0.0834	0.0324	0.0496	0.0939	0.0673	1	0.105	0.0857	0.0475	0.0532	0.0317	0.0431	0.0722	0.0663	0.0303	0.0315	0.0776	-
P8	0.0798	0.0423	0.0272	0.0484	0.0512	0.0333	0.105	1	0.0509	0.038	0.0294	0.0431	0.0309	0.0362	0.0474	0.015	0.0248	0.0463	-
P9	0.074	0.159	0.0371	0.0994	0.0765	0.0615	0.0857	0.0509	1	0.0497	0.0731	0.0233	0.0366	0.0941	0.0708	0.0288	0.0576	0.0196	-
P10	0.0393	0.0195	0.0428	0.0278	0.0281	0.0538	0.0475	0.038	0.0497	1	0.0353	0.0143	0.0209	0.0429	0.0354	0.00701	0.0149	0.0134	-
P11	0.0419	0.0285	0.0487	0.0593	0.0349	0.0352	0.0532	0.0294	0.0731	0.0353	1	0.0077	0.0311	0.057	0.0412	0.0404	0.0456	0.0164	-
P12	0.0329	0.0116	0.0132	0.0109	0.0118	0.0299	0.0317	0.0431	0.0233	0.0143	0.0077	1	0.0112	0.0149	0.0197	0.000939	0.00489	0.00926	-
P13	0.0262	0.0283	0.0192	0.0742	0.048	0.0434	0.0431	0.0309	0.0366	0.0209	0.0311	0.0112	1	0.0625	0.0366	0.0368	0.0216	0.0246	-
P14	0.0368	0.0544	0.0447	0.0446	0.04	0.0528	0.0722	0.0362	0.0941	0.0429	0.057	0.0149	0.0625	1	0.0502	0.0129	0.0614	0.0775	-
Avg.[i]	0.0558	0.0519	0.0327	0.0480	0.0510	0.0397	0.0663	0.0474	0.0708	0.0354	0.0412	0.0197	0.0366	0.0502	0.0462	-	-	-	-
FS1	0.00962	0.00249	0.0285	0.0247	0.021	0.0197	0.0303	0.015	0.0288	0.00701	0.0404	0.000939	0.0368	0.0129	-	1	0.0242	0.0222	0.0232
TS1	0.0219	0.046	0.0192	0.0376	0.0263	0.0201	0.0315	0.0248	0.0576	0.0149	0.0456	0.00489	0.0216	0.0614	-	0.0242	1	0.042	0.0331
DS1	0.00331	0.00056	0.0175	0.0144	0.00529	0.0331	0.0776	0.0463	0.0196	0.0134	0.0164	0.00926	0.0246	0.0775	-	0.0222	0.042	1	0.0321
Avg.[ii]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.0232	0.0331	0.0321	0.0295

Average

[i] Average of cross correlation with other PC workloads, excluding self.
[ii] Average of cross correlation with other server workloads, excluding self.

Table A-1: Cross-Correlation of Per-Minute Volume of I/O Activity.

Average

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	Avg.[i]	FS1	TS1	DS1	Avg.[ii]
P1	1	0.16	0.108	0.129	0.228	0.118	0.233	0.22	0.16	0.106	0.123	0.08	0.0812	0.0832	0.141	0.0262	0.0505	0.0179	-
P2	0.16	1	0.0807	0.132	0.217	0.0925	0.183	0.126	0.35	0.0486	0.0898	0.0392	0.0858	0.125	0.133	0.00882	0.078	-0.00217	-
P3	0.108	0.0807	1	0.0311	0.141	0.111	0.101	0.0788	0.11	0.113	0.131	0.026	0.0619	0.0944	0.091	0.0619	0.0356	0.0549	-
P4	0.129	0.132	0.0311	1	0.12	0.072	0.115	0.107	0.194	0.061	0.113	0.035	0.176	0.0743	0.105	0.0481	0.0706	0.0321	-
P5	0.228	0.217	0.141	0.12	1	0.137	0.216	0.167	0.211	0.0959	0.107	0.0321	0.147	0.107	0.148	0.0579	0.0506	0.0397	-
P6	0.118	0.0925	0.111	0.072	0.137	1	0.187	0.114	0.183	0.141	0.129	0.0743	0.142	0.145	0.127	0.0524	0.0486	0.0885	-
P7	0.233	0.183	0.101	0.115	0.216	0.187	1	0.255	0.201	0.0971	0.119	0.0667	0.0945	0.136	0.154	0.0608	0.0569	0.141	-
P8	0.22	0.126	0.0788	0.107	0.167	0.114	0.255	1	0.115	0.0825	0.0906	0.0947	0.0832	0.0819	0.124	0.0338	0.0518	0.106	-
P9	0.16	0.35	0.11	0.194	0.211	0.183	0.201	0.115	1	0.108	0.143	0.0441	0.0962	0.173	0.161	0.059	0.108	0.0323	-
P10	0.106	0.0486	0.113	0.061	0.0959	0.141	0.0971	0.0825	0.108	1	0.0771	0.0344	0.0546	0.0914	0.085	0.0184	0.0392	0.0394	-
P11	0.123	0.0898	0.131	0.113	0.107	0.129	0.119	0.0906	0.143	0.0771	1	0.0193	0.108	0.108	0.104	0.0869	0.0993	0.0294	-
P12	0.08	0.0392	0.026	0.035	0.0321	0.0743	0.0667	0.0947	0.0441	0.0344	0.0193	1	0.0229	0.0248	0.046	0.000372	0.00633	0.03	-
P13	0.0812	0.0858	0.0619	0.176	0.147	0.142	0.0945	0.0832	0.0962	0.0546	0.108	0.0229	1	0.145	0.100	0.0815	0.0424	0.0513	-
P14	0.0832	0.125	0.0944	0.0743	0.107	0.145	0.136	0.0819	0.173	0.0914	0.108	0.0248	0.145	1	0.107	0.0267	0.106	0.119	-
Avg.[i]	0.141	0.133	0.091	0.105	0.148	0.127	0.154	0.124	0.161	0.085	0.104	0.046	0.100	0.107	0.116	-	-	-	-
FS1	0.0262	0.00882	0.0619	0.0481	0.0579	0.0524	0.0608	0.0338	0.059	0.0184	0.0869	0.000372	0.0815	0.0267	-	1	0.0462	0.0405	0.0434
TS1	0.0505	0.078	0.0356	0.0706	0.0506	0.0486	0.0569	0.0518	0.108	0.0392	0.0993	0.00633	0.0424	0.106	-	0.0462	1	0.04	0.0431
DS1	0.0179	-0.00217	0.0549	0.0321	0.0397	0.0885	0.141	0.106	0.0323	0.0394	0.0294	0.03	0.0513	0.119	-	0.0405	0.04	1	0.0403
Avg.[ii]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.0434	0.0431	0.0403	0.0422

Average

[i] Average of cross correlation with other PC workloads, excluding self.
[ii] Average of cross correlation with other server workloads, excluding self.

Table A-2: Cross-Correlation of Per-10-Minute Volume of I/O Activity.

Average

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	Avg.[i]	FS1	TS1	DS1	Avg.[ii]
P1	1	0.376	0.298	0.262	0.484	0.329	0.39	0.434	0.292	0.215	0.207	0.126	0.199	0.202	0.293	0.0413	0.147	0.071	-
P2	0.376	1	0.244	0.258	0.479	0.302	0.384	0.344	0.418	0.138	0.186	0.15	0.233	0.342	0.296	0.0272	0.145	0.058	-
P3	0.298	0.244	1	0.106	0.306	0.29	0.181	0.199	0.295	0.206	0.242	0.0813	0.151	0.207	0.216	0.142	0.121	0.062	-
P4	0.262	0.258	0.106	1	0.231	0.168	0.278	0.271	0.323	0.154	0.249	0.0963	0.347	0.165	0.224	0.0812	0.155	0.11	-
P5	0.484	0.479	0.306	0.231	1	0.38	0.372	0.344	0.384	0.227	0.223	0.0619	0.296	0.22	0.308	0.0903	0.131	0.082	-
P6	0.329	0.302	0.29	0.168	0.38	1	0.376	0.258	0.356	0.252	0.213	0.14	0.327	0.291	0.283	0.11	0.142	0.241	-
P7	0.39	0.384	0.181	0.278	0.372	0.376	1	0.454	0.361	0.177	0.171	0.156	0.187	0.241	0.287	0.121	0.119	0.188	-
P8	0.434	0.344	0.199	0.271	0.344	0.258	0.454	1	0.255	0.164	0.183	0.157	0.193	0.187	0.265	0.0764	0.129	0.267	-
P9	0.292	0.418	0.295	0.323	0.384	0.356	0.361	0.255	1	0.263	0.216	0.126	0.197	0.331	0.294	0.088	0.169	0.0909	-
P10	0.215	0.138	0.206	0.154	0.227	0.252	0.177	0.164	0.263	1	0.15	0.0763	0.136	0.209	0.182	0.0247	0.144	0.107	-
P11	0.207	0.186	0.242	0.249	0.223	0.213	0.171	0.183	0.216	0.15	1	0.0297	0.19	0.244	0.193	0.145	0.187	0.0627	-
P12	0.126	0.15	0.0813	0.0963	0.0619	0.14	0.156	0.157	0.126	0.0763	0.0297	1	0.0355	0.0485	0.099	-0.00785	0.0473	0.06	-
P13	0.199	0.233	0.151	0.347	0.296	0.327	0.187	0.193	0.197	0.136	0.19	0.0355	1	0.298	0.215	0.16	0.131	0.12	-
P14	0.202	0.342	0.207	0.165	0.22	0.291	0.241	0.187	0.331	0.209	0.244	0.0485	0.298	1	0.230	0.0355	0.243	0.161	-
Avg.[i]	0.293	0.296	0.216	0.224	0.308	0.283	0.287	0.265	0.294	0.182	0.193	0.099	0.215	0.230	0.242	-	-	-	-
FS1	0.0413	0.0272	0.142	0.0812	0.0903	0.11	0.121	0.0764	0.088	0.0247	0.145	-0.00785	0.16	0.0355	-	1	0.076	0.0832	0.0796
TS1	0.147	0.145	0.121	0.155	0.131	0.142	0.119	0.129	0.169	0.144	0.187	0.0473	0.131	0.243	-	0.076	1	0.0422	0.0591
DS1	0.071	0.058	0.062	0.11	0.082	0.241	0.188	0.267	0.0909	0.107	0.0627	0.06	0.12	0.161	-	0.0832	0.0422	1	0.0627
Avg.[ii]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.0796	0.0591	0.0627	0.0671

Average

[i] Average of cross correlation with other PC workloads, excluding self.
[ii] Average of cross correlation with other server workloads, excluding self.

Table A-3: Cross-Correlation of Hourly Volume of I/O Activity.

Average

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	Avg.[i]	FS1	TS1	DS1	Avg.[ii]
P1	1	0.579	0.241	0.488	0.659	0.647	0.492	0.684	0.544	0.38	0.116	0.205	0.315	0.18	0.425	0.065	0.341	-0.0254	-
P2	0.579	1	0.115	0.319	0.631	0.513	0.628	0.736	0.565	0.317	0.243	0.253	0.537	0.464	0.454	-0.118	0.445	0.184	-
P3	0.241	0.115	1	0.136	0.312	0.26	-0.0361	0.201	0.37	0.354	0.249	-0.0782	0.171	0.12	0.186	0.288	0.401	0.176	-
P4	0.488	0.319	0.136	1	0.323	0.277	0.219	0.516	0.502	0.35	0.434	0.224	0.469	0.203	0.343	0.0703	0.552	-0.466	-
P5	0.659	0.631	0.312	0.323	1	0.592	0.507	0.618	0.566	0.452	0.0851	-0.0555	0.382	0.17	0.403	0.135	0.344	-0.0191	-
P6	0.647	0.513	0.26	0.277	0.592	1	0.569	0.406	0.619	0.426	0.141	0.25	0.591	0.321	0.432	0.0314	0.476	0.414	-
P7	0.492	0.628	-0.0361	0.219	0.507	0.569	1	0.597	0.563	0.162	0.0476	0.373	0.455	0.324	0.377	0.0792	0.204	0.278	-
P8	0.684	0.736	0.201	0.516	0.618	0.406	0.597	1	0.542	0.224	0.132	0.266	0.369	0.22	0.424	-0.0358	0.333	0.23	-
P9	0.544	0.565	0.37	0.502	0.566	0.619	0.563	0.542	1	0.728	0.0909	0.352	0.404	0.376	0.479	0.175	0.629	-0.0133	-
P10	0.38	0.317	0.354	0.35	0.452	0.426	0.162	0.224	0.728	1	0.116	0.0664	0.431	0.584	0.353	0.062	0.472	-0.0131	-
P11	0.116	0.243	0.249	0.434	0.0851	0.141	0.0476	0.132	0.0909	0.116	1	0.0112	0.272	0.387	0.179	0.163	0.518	0.387	-
P12	0.205	0.253	-0.0782	0.224	-0.0555	0.25	0.373	0.266	0.352	0.0664	0.0112	1	0.23	0.11	0.170	-0.163	0.0531	-0.201	-
P13	0.315	0.537	0.171	0.469	0.382	0.591	0.455	0.369	0.404	0.431	0.272	0.23	1	0.586	0.401	0.0261	0.59	0.133	-
P14	0.18	0.464	0.12	0.203	0.17	0.321	0.324	0.22	0.376	0.584	0.387	0.11	0.586	1	0.311	-0.297	0.523	0.13	-
Avg.[i]	0.425	0.454	0.186	0.343	0.403	0.432	0.377	0.424	0.479	0.353	0.179	0.170	0.401	0.311	0.353	-	-	-	-
FS1	0.065	-0.118	0.288	0.0703	0.135	0.0314	0.0792	-0.0358	0.175	0.062	0.163	-0.163	0.0261	-0.297	-	1	0.133	-0.343	-0.105
TS1	0.341	0.445	0.401	0.552	0.344	0.476	0.204	0.333	0.629	0.472	0.518	0.0531	0.59	0.523	-	0.133	1	0.18	0.157
DS1	-0.0254	0.184	0.176	-0.466	-0.0191	0.414	0.278	0.23	-0.0133	-0.0131	0.387	-0.201	0.133	0.13	-	-0.343	0.18	1	-0.0815
Avg.[ii]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-0.105	0.157	-0.0815	-0.0100

Average

[i] Average of cross correlation with other PC workloads, excluding self.
[ii] Average of cross correlation with other server workloads, excluding self.

Table A-4: Cross-Correlation of Daily Volume of I/O Activity.

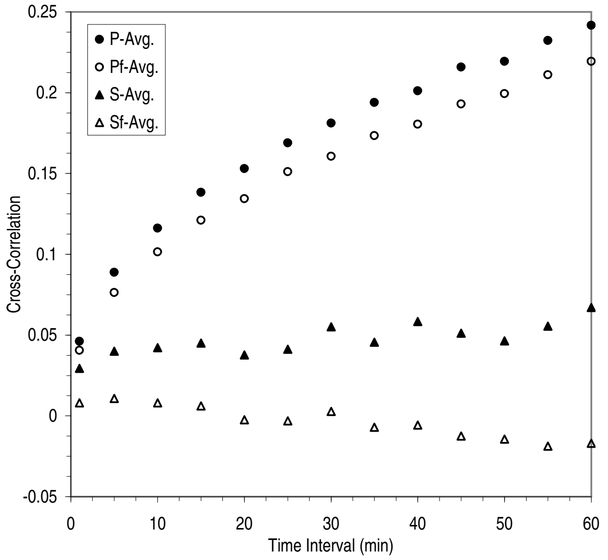


Figure A-3: Cross-Correlation of Volume of I/O Activity vs. Time Interval Used to Aggregate Volume.

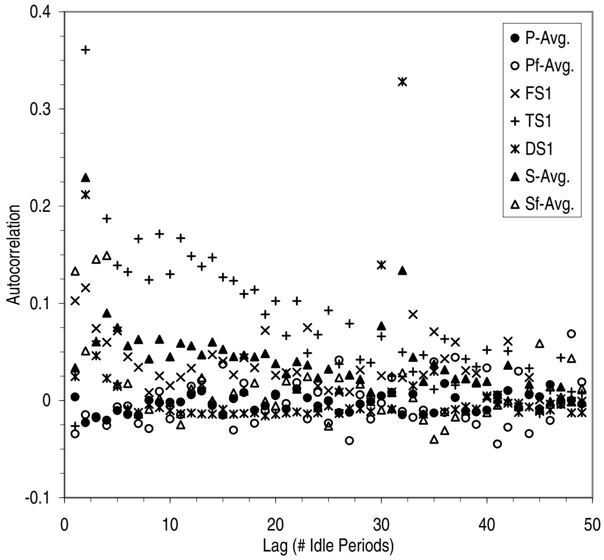
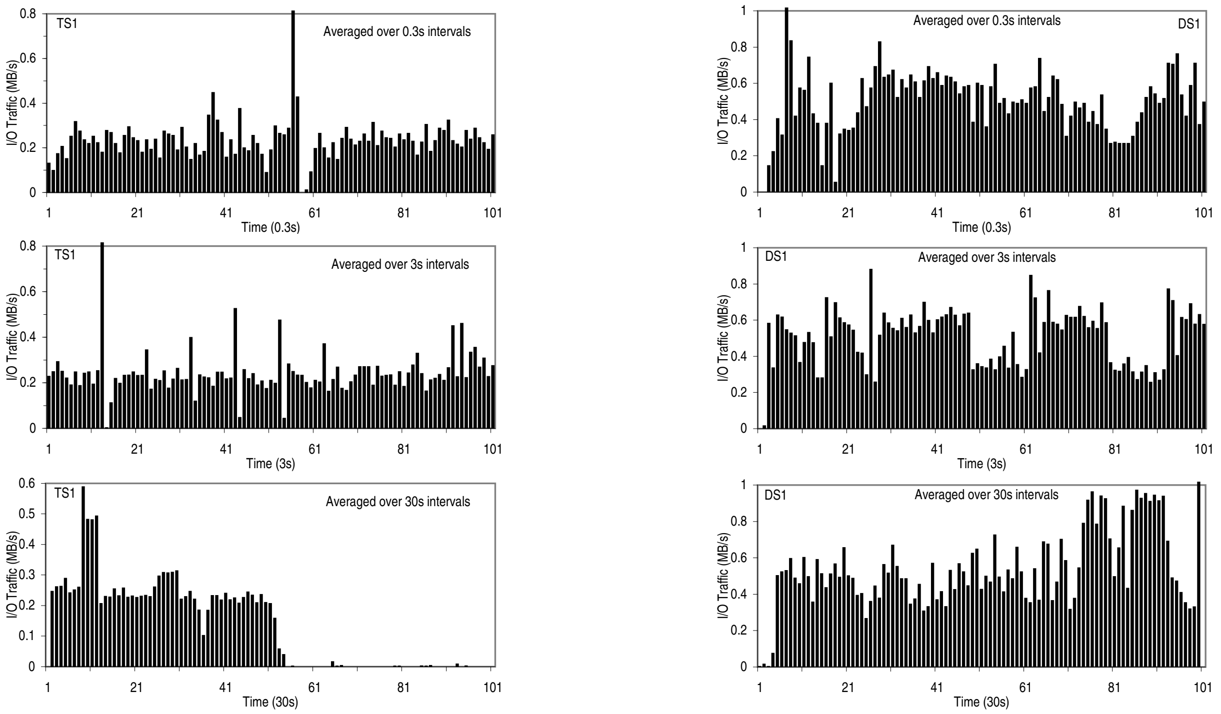


Figure A-4: Autocorrelation of the Sequence of Idle Period Duration.



(a) TS1

(b) DS1

Figure A-5: I/O Traffic at Different Time Scales during the High-Traffic Period (One-hour period that contains more I/O traffic than 95% of other one-hour periods).

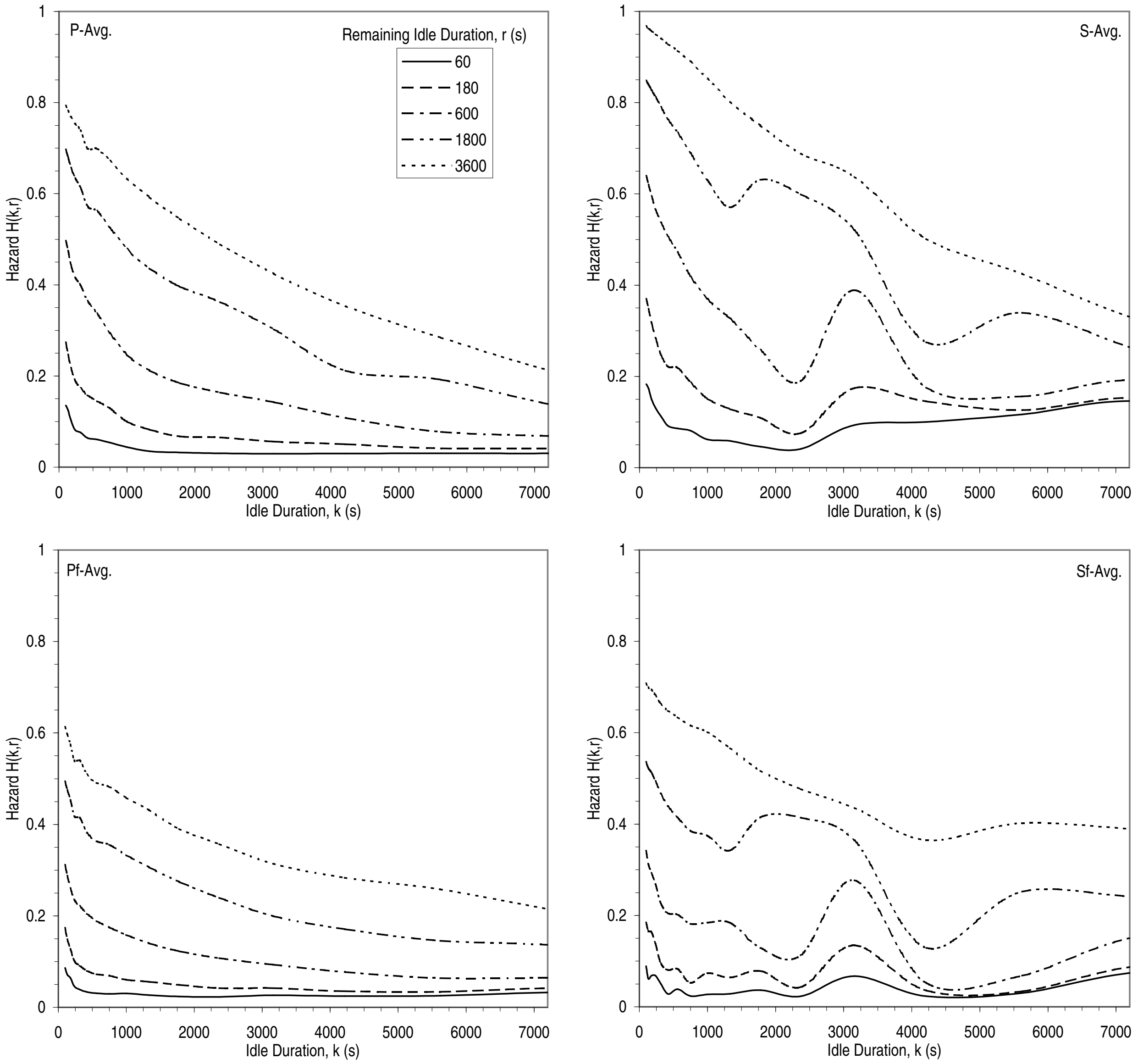


Figure A-6: Hazard Rate for the Distribution of Idle Period Duration.

Appendix B

B-1 Estimating the Degree of Self-Similarity

The degree of self-similarity is expressed using a single parameter, the Hurst parameter H. For a self-similar series, $1/2 < H < 1$, and as $H \rightarrow 1$, the degree of self-similarity increases. For smooth Poisson traffic, H is 1/2. 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. In this paper, we focus on the R/S method and the variance-time plot. Newer inference methods that are more sensitive to different types of scaling phenomena (*e.g.,* [1]) have been developed but are beyond the scope of the current paper.

B-1.1 The R/S Method

One of the manifestations of the self-similar property is that the autocorrelations of the process decay hyperbolically rather than exponentially. This behavior is known as long-range dependence and it provides an explanation for an empirical law known as the Hurst effect [27].

The R/S or rescaled adjusted range statistic for a set of observations $X_k : k = 1, 2, ..., n$ having mean $\overline{X}(n)$ and sample variance $S^2(n)$ is defined by

$$\frac{R(n)}{S(n)} = \frac{1}{S(n)}[max(0, w_1, W_2, ..., W_n) - min(0, W_1, W_2, ..., W_n)] \tag{B-1}$$

where

$$W_k = (X_1 + X_2 + ... + X_k) - k\overline{X}(n),\ k \geq 1.$$

It turns out that

$$E\left[\frac{R(n)}{S(n)}\right] \sim cn^H \tag{B-2}$$

where $H = 0.5$ for short-range dependent processes and $0.5 < H < 1$ for long-range dependent processes. This difference between short and long-range dependent processes is known as the Hurst effect and forms the basis for the R/S method of inferring the Hurst parameter.

Taking logarithm on both sides of Equation B-2,

$$log\left(E\left[\frac{R(n)}{S(n)}\right]\right) \sim Hlog(n) + log(c) \tag{B-3}$$

Therefore, we can estimate H by plotting $log(E[R(n)/S(n)])$ versus $log(n)$ for different values of n. In practice, we divide a set of N observations into K disjoint subsets each of length N/K and compute $log(E[R(n)/S(n)])$ for each of these subsets using logarithmically spaced values of n. The resulting plot of $log(E[R(n)/S(n)])$ versus $log(n)$ is commonly referred to as a pox plot. For a long-range dependent time series, the pox plot should fluctuate in a straight street of slope H, $0.5 < H < 1$ [5].

In Figure B-1, we present the pox plots for our various workloads for the high-traffic period. Observe that the pox plots for all the workloads appear to fluctuate around straight streets with slope ranging from 0.6 to almost 0.9. In other words, *all the workloads exhibit long-range dependence and self-similarity in their I/O traffic patterns.* In Figure B-2, we present the corresponding pox plots for the filtered traces. The same behavior is observed.

B-1.2 Variance-Time Plot

Another manifestation of self-similarity is that the variance of the aggregated process $X^{(m)}$ decrease more slowly than the reciprocal of m, where

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

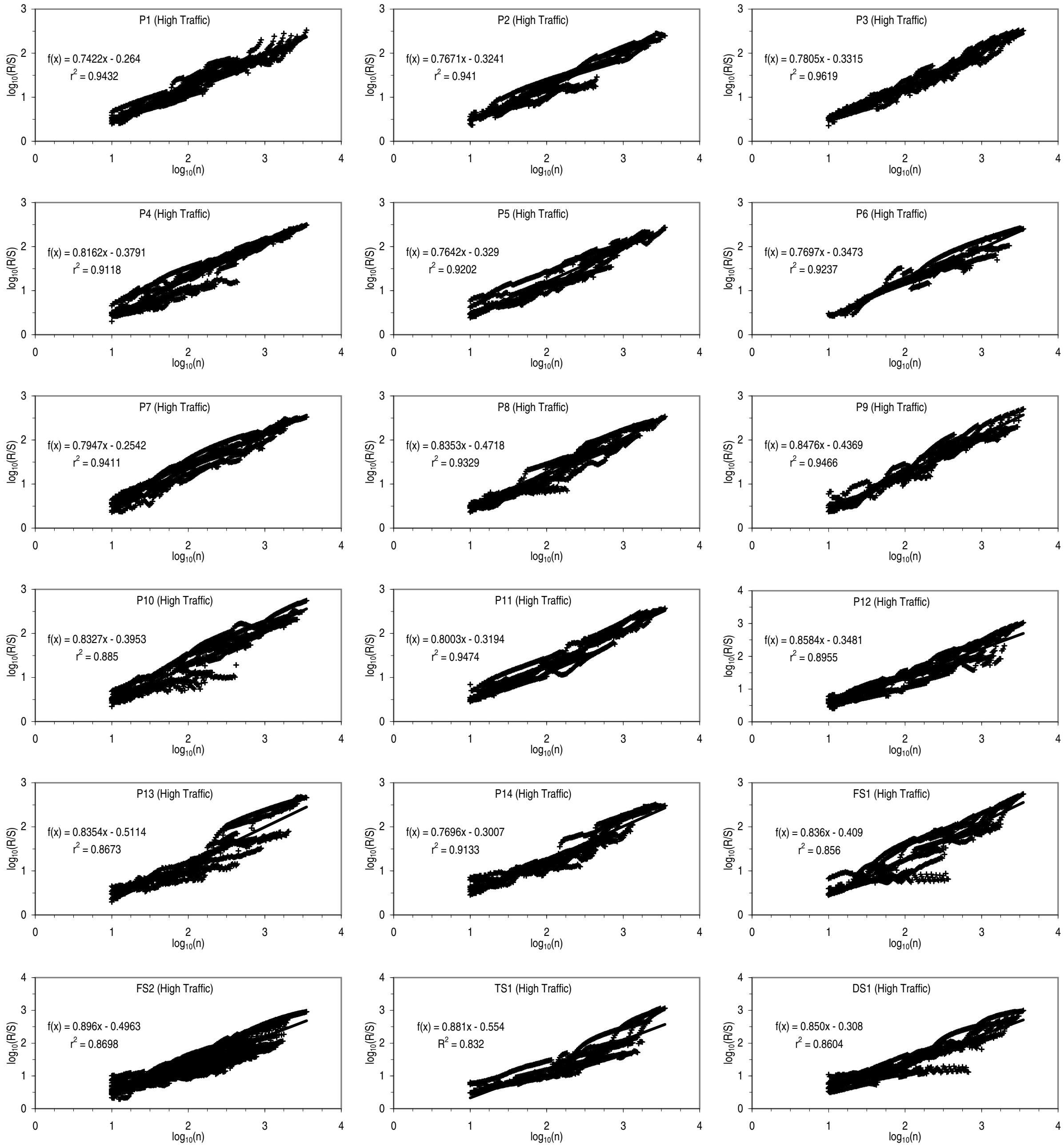


Figure B-1: Pox Plots to Detect Self-Similarity.

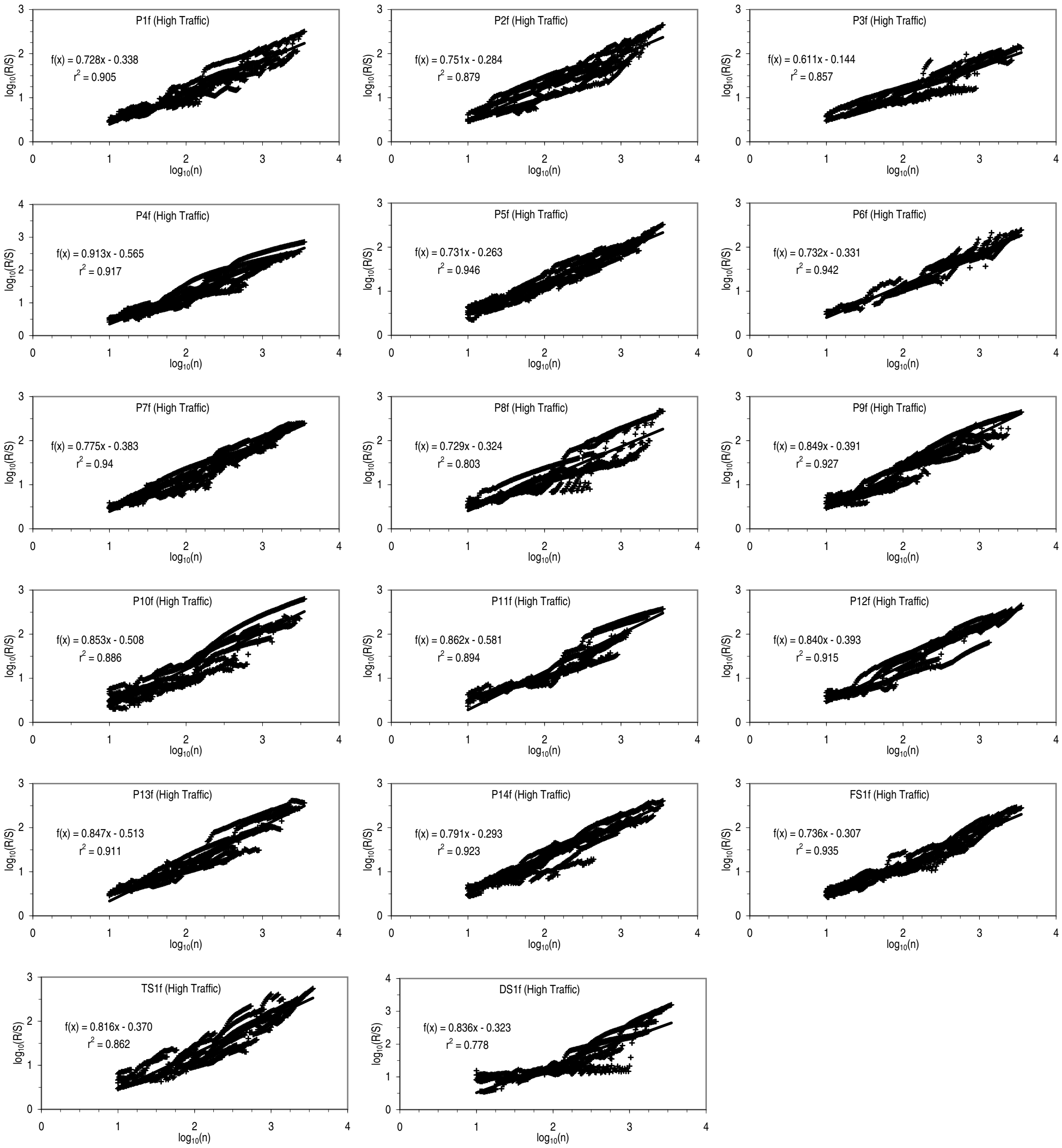
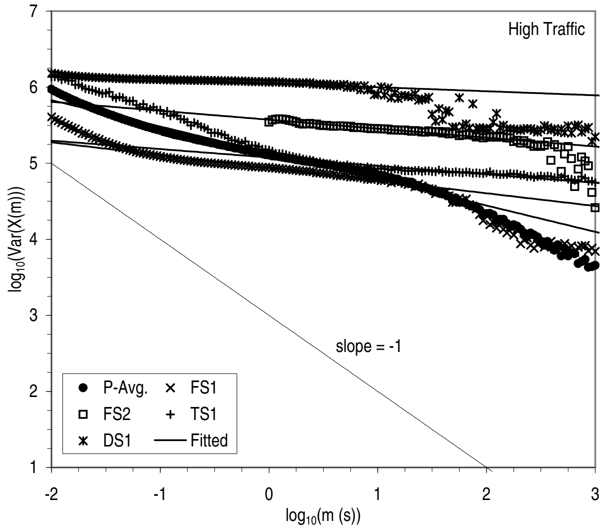
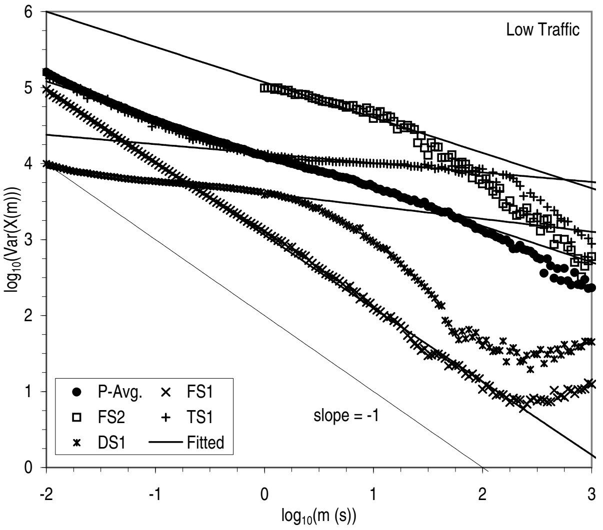


Figure B-2: Pox Plots to Detect Self-Similarity (Filtered Traces).



(a) High-Traffic Period.



(b) Low-Traffic Period.

Figure B-3: Variance-Time Plots to Detect Self-Similarity.

More formally,

$$Var(X^{(m)}) \quad \sim \quad cm^{-\beta},\ 0 < \beta < 1. \tag{B-4}$$

Taking logarithm on both sides,

$$log(Var(X^{(m)})) \quad \sim \quad log(c) - \beta log(m). \tag{B-5}$$

Thus for a self-similar process, the variance-time plot, *i.e.,* the plot of $log(Var(X^{(m)}))$ against $log(m)$, should be a straight line with a slope between -1 and 0. The degree of self-similarity is given by $H = 1 - \beta/2$.

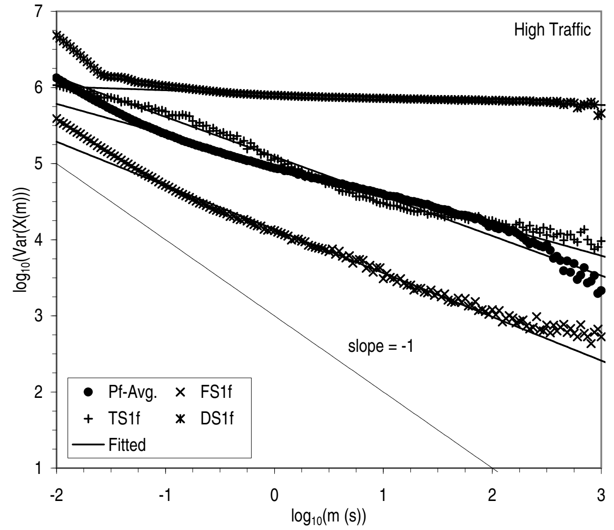
The variance-time plots for our various workloads are presented in Figures B-3 and B-4. Observe that for the high-traffic period, the variance-time plots for all the workloads are very linear with slopes that are more gradual than -1. This indicates that the I/O traffic for the workloads is self-similar in nature. Notice though that *the self-similarity does not span all time scales but appears to break down beginning just beyond 10s for the database server. In other words, 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 but not beyond that.* Interestingly, the filtered traces appear to be self-similar to larger time scales although some of them have a steeper slope, meaning that they are less self-similar.

For the low-traffic period, all the plots again have linear segments with slope of less than -1 but these segments are shorter than in the high-traffic case, particularly in the case of the database server. In addition, the slope of the linear regions is noticeably steeper than for the high-traffic period. This means that *I/O traffic during the low-traffic period is self-similar but less so and over a smaller range of time scales than during the high-traffic period.* As discussed in the main text, the self-similarity could be caused by the superposition of I/O generated by different processes in the system where each process behaves as an independent I/O source with heavy-tailed on periods. During the low-traffic period, we would expect that there are fewer processes running in the system and therefore fewer independent sources of I/O so that the aggregated traffic is less self-similar. This is in line with observations in [13].

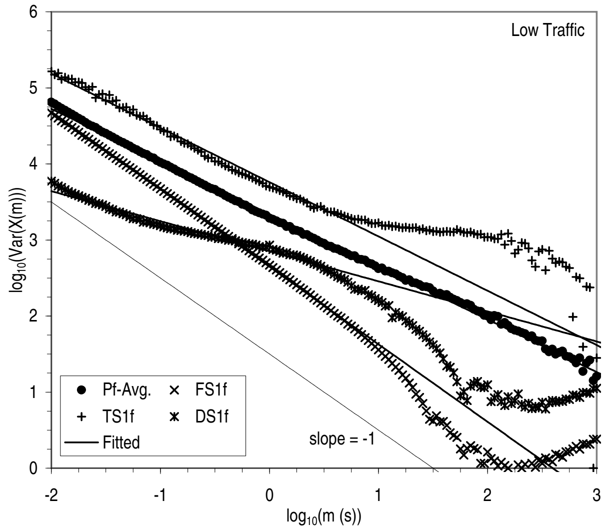
Table B-1 summarizes the Hurst parameter values that we obtained using the R/S method and the variance-time plot. These two methods provide independent estimates of the degree of self-similarity and discrepancies between their results can be expected. In view of this, the figures we obtained are reasonably consistent, which adds confidence to our analysis and results.

B-2 Generating Self-Similar I/O Traffic

There are several ways to generate self-similar traffic but models such as those based on F-ARIMA and Fractional Guassian Noise processes are generally computationally expensive. An alternative traffic generator based on the superposition of inde-



(a) High-Traffic Period.



(b) Low-Traffic Period.

Figure B-4: Variance-Time Plots to Detect Self-Similarity (Filtered Traces).

High Traffic		P-Avg.	Pf-Avg.	FS1	FS2	TS1	DS1	S-Avg.	Sf-Avg.
Var.-Time	Slope of Fitted Line	-0.35	-0.40	-0.17	-0.12	-0.11	-0.053	-0.11	-0.38
	Hurst Parameter	0.83	0.80	0.92	0.94	0.94	0.97	0.94	0.81
Pox	Slope of Fitted Line	0.80	0.79	0.84	0.90	0.88	0.85	0.86	0.80
	Hurst Parameter	0.80	0.79	0.84	0.90	0.88	0.85	0.86	0.80

(a) High Traffic.

High Read Traffic		P-Avg.	Pf-Avg.	FS1	FS2	TS1	DS1	S-Avg.	Sf-Avg.
Var.-Time	Slope of Fitted Line	-0.26	-0.29	-0.20	-0.10	-0.13	-0.10	-0.14	-0.13
	Hurst Parameter	0.87	0.85	0.90	0.95	0.94	0.95	0.93	0.93
Pox	Slope of Fitted Line	0.77	0.74	0.85	0.92	0.79	0.76	0.80	0.77
	Hurst Parameter	0.77	0.74	0.85	0.92	0.79	0.76	0.80	0.77

(b) High Read Traffic.

High Write Traffic		P-Avg.	Pf-Avg.	FS1	FS2	TS1	DS1	S-Avg.	Sf-Avg.
Var.-Time	Slope of Fitted Line	-0.50	-0.55	-0.29	-0.12	-0.28	-0.068	-0.21	-0.49
	Hurst Parameter	0.75	0.73	0.85	0.94	0.86	0.97	0.89	0.76
Pox	Slope of Fitted Line	0.79	0.78	0.81	0.76	0.88	0.82	0.83	0.79
	Hurst Parameter	0.79	0.78	0.81	0.76	0.88	0.82	0.83	0.79

(c) High Write Traffic.

Table B-1: Degree of Self-Similarity.

pendent and identical fractal renewal processes is attractive because it has a physical correspondence to the superposition of I/O traffic generated by different processes, and is relatively easy to construct. The Superposition of Fractal Renewal Processes model is completely characterized by M, the number of fractal renewal processes, and $p(\tau)$, the inter-arrival probability density function. A convenient probability density function is the following where the parameter A serves as a threshold between exponential behavior and power-law behavior:

$$p(\tau) = \begin{cases} \frac{\gamma}{A} e^{\frac{-\gamma\tau}{A}}, & \tau \leq A, \\ \gamma e^{-\gamma} A \gamma \tau^{-(\gamma+1)}, & \tau > A \end{cases} \tag{B-6}$$

The interested reader is referred to [47] for more details about the model.

B-2.1 The Inputs

The inputs to the traffic generator are:

1. H, the Hurst parameter which measures the degree of self-similarity [5].
2. μ, the average number of arrivals during intervals of duration T_s.
3. σ^2, the variance in the number of arrivals during intervals of duration T_s.

B-2.2 Model Setup

The three inputs described above were chosen to be relatively easy to measure and understand. Before we begin to generate the traffic, however, we need to convert the inputs into a more convenient form:

1. Calculate

$$\alpha = 2H - 1 \tag{B-7}$$

2. Calculate

$$\gamma = 2 - \alpha \tag{B-8}$$

3. Calculate

$$\lambda = \frac{\mu}{T_s} \tag{B-9}$$

4. Calculate

$$T_o = \frac{T_s}{(\frac{\sigma^2}{\lambda T_s})^{\frac{1}{\alpha}} - 1} \tag{B-10}$$

5. Calculate

$$A = \left[\frac{T_o^\alpha 2\gamma^2(\gamma - 1)e^\gamma}{(2-\gamma)(3-\gamma)[1 + (\gamma - 1)e^\gamma]^2} \right]^{\frac{1}{\alpha}} \tag{B-11}$$

6. Calculate

$$M = \left\lfloor \frac{A\lambda}{\gamma} \left[1 + \frac{1}{(\gamma - 1)e^\gamma} \right] \right\rfloor \tag{B-12}$$

B-2.3 The Algorithm

Let $T_i^{(j)}$ denote the ith inter-arrival time for process j. The following algorithm calculates the $T_i^{(j)}$ by spawning M independent threads. This multi-threaded approach is useful when actual I/Os are to be issued. For pure simulations or where I/O calls return immediately after they have been issued, a single-threaded version can be easily constructed.

1. Spawn M threads
2. For each thread
3. Generate a random variable U uniformly distributed in [0,1)
4. Calculate

$$V = \frac{1 + (\gamma - 1)e^{\gamma}}{\gamma} U \tag{B-13}$$

5. Calculate

$$\tau_o^{(j)} = \begin{cases} -\gamma^{-1} A \ln[U \frac{\gamma V - 1}{\gamma V - U}], & V \geq 1, \\ A V^{\frac{1}{1-\gamma}}, & V < 1 \end{cases} \tag{B-14}$$

6. Repeat
7. Generate a random variable U uniformly distributed in [0,1)
8. Calculate

$$\tau_i^{(j)} = \begin{cases} -\frac{1}{\gamma} A \ln[U], & U \geq e^{-\gamma}, \\ \frac{1}{e} A U^{\frac{-1}{\gamma}}, & U < e^{-\gamma} \end{cases} \tag{B-15}$$

9. Until enough arrivals are generated