UNIX disk access patterns

Chris Ruemmler and John Wilkes

Computer Systems Laboratory
Hewlett-Packard Laboratories, Palo Alto, CA

HPL–92–152, December 1992

Also published in *USENIX Winter 1993 Technical Conference Proceedings* (San Diego, CA), 25–29 January 1993, pages 405–420.

UNIX disk access patterns

Chris Ruemmler and John Wilkes – Hewlett-Packard Laboratories

ABSTRACT

Disk access patterns are becoming ever more important to understand as the gap between processor and disk performance increases. The study presented here is a detailed characterization of every low-level disk access generated by three quite different systems over a two month period. The contributions of this paper are the detailed information we provide about the disk accesses on these systems (many of our results are significantly different from those reported in the literature, which provide summary data only for file-level access on small-memory systems); and the analysis of a set of optimizations that could be applied at the disk level to improve performance.

Our traces show that the majority of all operations are writes; disk accesses are rarely sequential; 25–50% of all accesses are asynchronous; only 13–41% of accesses are to user data (the rest result from swapping, metadata, and program execution); and I/O activity is very bursty: mean request queue lengths seen by an incoming request range from 1.7 to 8.9 (1.2–1.9 for reads, 2.0–14.8 for writes), while we saw 95th percentile queue lengths as large as 89 entries, and maxima of over 1000.

Using a simulator to analyze the effect of write caching at the disk level, we found that using a small non-volatile cache at each disk allowed writes to be serviced considerably faster than with a regular disk. In particular, short bursts of writes go much faster – and such bursts are common: writes rarely come singly. Adding even 8KB of non-volatile memory per disk could reduce disk traffic by 10–18%, and 90% of metadata write traffic can be absorbed with as little as 0.2MB per disk of non-volatile RAM. Even 128KB of NVRAM cache in each disk can improve write performance by as much as a factor of three. FCFS scheduling for the cached writes gave better performance than a more advanced technique at small cache sizes.

Our results provide quantitative input to people investigating improved file system designs (such as log-based ones), as well as to I/O subsystem and disk controller designers.

Introduction

The I/O gap between processor speed and dynamic disk performance has been growing as VLSI performance (improving at 40–60% per year) outstrips the rate at which disk access times improve (about 7% per year). Unless something is done, new processor technologies will not be able to deliver their full promise. Fixes to this problem have concentrated on ever-larger file buffer caches, and on speeding up disk I/Os through the use of more sophisticated access strategies. Surprisingly, however, there has been very little published on the detailed low-level behavior of disk I/Os in modern systems, which such techniques are attempting to improve. This paper fills part of this void, and also uses the data to provide analyses of some techniques for improving write performance through the use of disk caches.

We captured every disk I/O made by three different HP-UX systems during a four-month period (April 18, 1992 through August 31, 1992). We present here analyses of 63-day contiguous subsets of this data. The systems we traced are described in Table 1: two of them were at HP Laboratories, one (snake) at UC Berkeley.

The most significant results of our analyses of these systems are: the majority of disk accesses (57%) are writes; only 8–12% of write accesses, but 18–33% of reads, are logically sequential at the disk level; 50–75% of all I/Os are synchronous; the majority (67–78%) of writes are to metadata; user-data I/Os represent only 13–41% of the total accesses; 10–18% of all write requests are overwrites of the last block written out; and swap traffic is mostly reads (70–90%).

Name	*Processor*[a]	*MIPS*	*HP-UX Version*	*Physical memory*	*File buffer cache size*	*Fixed storage*	*Users*	*Usage type*
cello	HP 9000/877	76	8.02	96 MB	10/30[b] MB	10.4 GB	20	Timesharing
snake	HP 9000/720	58	8.05	32 MB	5 MB	3.0 GB	200	Server
hplajw	HP 9000/845	23	8.00	32 MB	3 MB	0.3 GB	1	Workstation

a. Each machine uses an HP PA-RISC microprocessor.
b. Cello's file buffer size changed from 10MB to 30MB on April 26, 1992.

Table 1: The three computer systems traced

Disk	Formatted capacity	Track buffer	Cylinders	Data heads	Rotational speed	Average 8KB access	Host interconnect	
							type	speed
HP C2200A	335 MB	none	1449	8	4002 rpm	33.6 ms	HP-IB	1–1.2MB/s[a]
HP C2204A	1.3 GB	none	2x1449[b]	2x16	4002 rpm	30.9 ms	HP-FL	5MB/s
HP C2474S	1.3 GB	128 KB	1935	19	4002 rpm	22.8 ms	SCSI-II	5MB/s
HP 97560	1.3 GB	128 KB	1935	19	4002 rpm	22.8 ms	SCSI-II	10MB/s
Quantum PD425S	407 MB	56 KB	1520	9	3605 rpm	26.3 ms	SCSI-II	5MB/s

a. 1MB/s towards the disk, 1.2MB/s towards the host.
b. A C2204A disk has two 5.25" mechanisms made to look like a single disk drive by the controller.

Table 2: Disk details

The paper is organized as follows. We begin with a short overview of previous work in the area. Then comes a description of our tracing method and details of the systems we traced; it is followed by a detailed analysis of the I/O patterns observed on each of the systems. Then we present the results of both simulations of adding non-volatile write buffers in the disks, and conclude with a summary of our results.

Related work

Most I/O access pattern studies have been performed at the file system level of the operating system rather than at the disk level. Since logging every file system operation (particularly every read and write) generates huge quantities of data, most such studies have produced only summary statistics or made some other compromise such as coalescing multiple I/Os together (e.g., [Ousterhout85, Floyd86, Floyd89]). Many were taken on non-UNIX systems. For example: IBM mainframes [Procar82, Smith85, Kure88, Staelin88, Staelin90, Staelin91, Bozman91]; Cray supercomputers [Miller91]; Sprite (with no timing data on individual I/Os) [Baker91]; DEC VMS [Ramakrishnan92]; TOPS-10 (static analysis only) [Satyanarayanan81].

The UNIX buffer cache means that most accesses never reach the disk, so these studies are not very good models of what happens at the disk. They also ignore the effects of file-system generated traffic, such as for metadata and read-ahead, and the effects of swapping and paging. There have been a few studies of disk traffic, but most have had flaws of one kind or another. For example: poor measurement technology (60 Hz timer) [Johnson87]; short trace periods (75 minutes at a time, no detailed reporting of data, 2ms timer granularity) [Muller91]; limited use patterns [Carson90]. Raymie Stata had earlier used the same tracing technology as this study to look at the I/Os in a time-sharing UNIX environment [Stata90]. He discovered skewed device utilization and small average device queue lengths with large bursts.

We were interested in pursuing this path further, and gathering detailed statistics without the limitations of others' work. The next section details how we did so.

Trace gathering

We traced three different Hewlett-Packard computer systems (described in Table 1). All were running release 8 of the HP-UX operating system [Clegg86], which uses a version of the BSD fast file system [McKusick84]. The systems had several different types of disks attached to them, whose properties are summarized in Table 2.

Trace collection method

All of our data were obtained using a kernel-level trace facility built into HP-UX. The tracing is completely transparent to the users and adds no noticeable processor load to the system. We logged the trace data to dedicated disks to avoid perturbing the system being measured (the traffic to these disks is excluded from our study). Channel contention is minimal: the logging only generates about one write every 7 seconds.

Each trace record contained the following data about a single physical I/O:

- timings, to 1μs resolution, of enqueue time (when the disk driver first sees the request); start time (when the request is sent to the disk) and completion time (when the request returns from the disk);
- disk number, partition and device driver type;
- start address (in 1KB fragments);
- transfer size (in bytes);
- the drive's request queue length upon arrival at the disk driver, including the current request;
- flags for read/write, asynchronous/synchronous, block/character mode;
- the type of block accessed (inode, directory, indirect block, data, superblock, cylinder group bitmap)

The tracing was driven by a daemon spawned from `init`; killing the daemon once a day caused a new

snake						
ID	disk type	partition	size	number of I/Os		reads
A	Quantum PD425S	/ (root)	313 MB	5 848 567	46.6%	40%
		(swap)	94 MB	50 694	0.4%	91%
B	HP 97560	/usr1	1.3 GB	2 862 620	22.8%	48%
C	HP 97560	/usr2	1.3 GB	3 793 004	30.2%	43%
		Total	3.0 GB	12 554 885	100.0%	43%

hplajw						
ID	disk type	partition	size	number of I/Os		reads
A	HP 2200A	/ (root)	278 MB	346 553	83.0%	36%
		(swap)	24 MB	19 625	4.7%	71%
		(swap)	16 MB	17 940	4.3%	68%
B[a]	HP 2200A	(swap)	16 MB	31 113	7.5%	82%
		Total	334 MB	416 262	99.5%[b]	42%

a. The remaining portion of this drive is the area where the trace data was collected.
b. The percentages do not add up to 100% due to raw disk accesses to the boot partition on disk A.

cello						
ID	disk type	partition	size	number of I/Os		reads
A	HP C2474S	/ (root)	958 MB	3 488 934	11.9%	59%
		/usr/local/lib/news	105 MB	3 511 772	12.0%	44%
		(swap1)	126 MB	243 932	0.8%	80%
		(swap2)	48 MB	59 423	0.2%	70%
B	HP 2204A	/users	1.3 GB	4 162 026	14.2%	47%
C	HP 2204A	/usr/local/src	1.3 GB	1 027 978	3.5%	85%
D	HP 2204A	/usr/spool/news	1.3 GB	14 372 382	49.0%	36%
E	HP 2204A	/usr/spool/news/in.batches	105 MB	479 441	1.6%	13%
		/nfs/export/refdbms	29 MB	21 531	0.1%	84%
		/backup	946 MB	185 845	0.6%	22%
		/tmp	126 MB	623 663	2.1%	4%
		(swap3)	75 MB	212 324	0.7%	80%
F	HP 2204A	/nobackup	1.3 GB	668 687	2.3%	71%
G	HP 2204A	/nobackup-2	1.3 GB	68 431	0.2%	79%
H	HP 2204A	/mount/oldroot	1.3 GB	224 908	0.8%	88%
		Subtotal for swap partitions	249 MB	515 679	1.8%	79%
		Subtotal for the news partitions	1.4 GB	18 363 595	62.6%	37%
		Subtotal excluding news and swap	8.8 GB	10 472 003	35.6%	54%
		Grand total	10.4 GB	29 351 277	100.0%	44%

Table 3: Summary of the traces gathered; cello and hplajw were traced from 92.4.18 to 92.6.20; snake from 92.4.25 to 92.6.27

trace file to be started (the kernel's buffering scheme meant that no events were lost). Each time this happened, we also collected data on the physical memory size, the cache size, system process identifiers, mounted disks, and the swap configuration.

Traced systems

Cello is a timesharing system used by a small group of researchers at Hewlett-Packard Laboratories to do simulation, compilation, editing, and mail. A *news* feed that was updated continuously throughout the day resulted in the majority (63%) of the I/Os in the system, and these I/Os have a higher-than-usual amount of writes (63%). The other partitions vary, with the mean being 46% writes. Because of the large activity directed to the news partitions, the system as a whole does more writes (56%) than reads.

Snake acted as a file server for an HP-UX cluster [Bartlett88] of nine clients at the University of California, Berkeley. Each client was an Hewlett-Packard 9000/720 workstation with 24MB of main memory, 66MB of local swap space, and a 4MB file buffer cache. There was no local file system storage on any of the clients; all the machines in the cluster saw a single common file system with complete single-system semantics. The cluster had accounts for professors, staff, graduate students, and computer science classes. The main use of the system was for compilation and editing. This cluster was new in January 1992, so many of the disk accesses were for the creation of new files. Over the tracing period, the */usr1* disk gained 243 MB and */usr2* gained 120 MB of data.

Finally, the personal workstation (hplajw) was used by the second author of this paper. The main uses of the system were electronic mail and editing papers. There was not much disk activity on this system: the file buffer cache was doing its job well.

Cello and hplajw were traced from 92.4.18 to 92.6.20; snake from 92.4.25 to 92.6.27. We also use a common week-long subset of the data for some analyses; this period runs from 92.5.30 to 92.6.6. All the numbers and graphs in this paper are derived from either the full or the week-long traces: we say explicitly if the shorter ones are being used. Each trace (full or short) starts at 0:00 hours on a Saturday.

The file system configurations for the three systems are given in Table 3. The total numbers of I/O requests logged over the tracing period discussed in this paper were: 29.4M (cello), 12.6M (snake) and 0.4M (hplajw).

The swap partitions are used as a backing store for the virtual memory system. In general, there is little swap activity (0.4% on snake, 1.8% on cello): these systems are reasonably well equipped with memory, or local swap space in the case of snake's diskless clients. The exception is hplajw, on which 16.5% of I/Os are for paging because of memory pressure from simultaneous execution of the X windowing system, FrameMaker, GNU Emacs, and a bibliography database program.

Analysis

This section presents our detailed analyses of the trace data. Although it represents measurements from a single file system design (the HP-UX/4.3BSD fast file system), we believe this data will be of use to other file system designers – particularly in providing upper bounds on the amount of disk traffic that might be saved by different approaches to designing file systems.

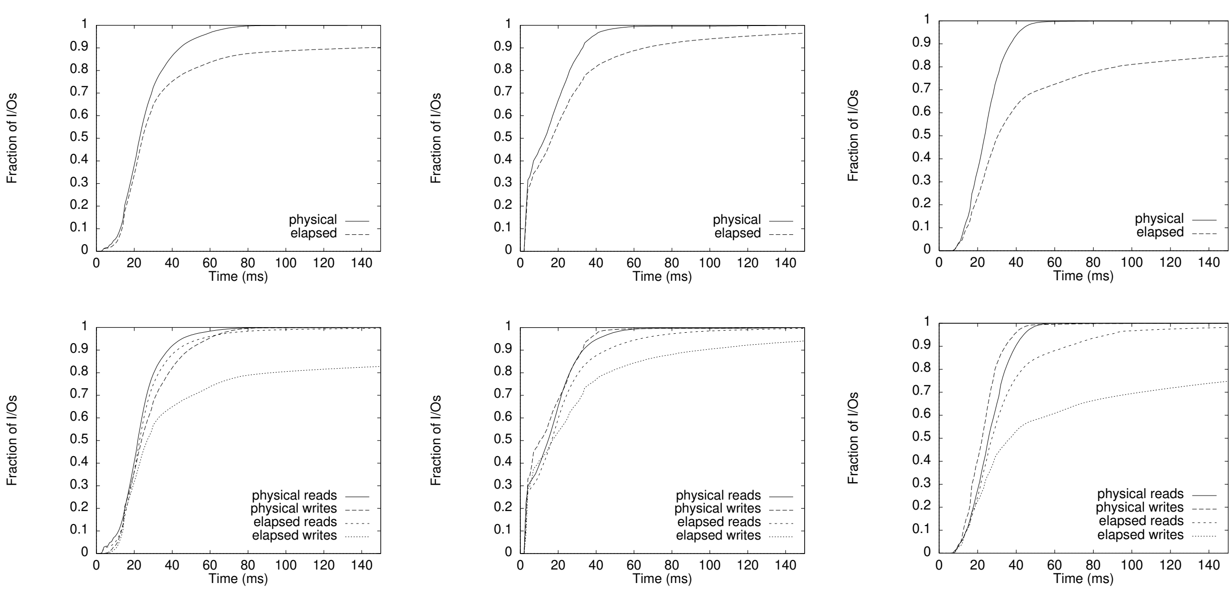
For example, we know that HP-UX is very enthusiastic about flushing metadata to disk to make the file system very robust against power failures.[1] This means that the metadata traffic we measured represents close to an upper bound on how much a metadata-logging file system might be able to suppress.

I/O timings

Figure 1 shows the distribution of both elapsed and physical I/O times for the three systems. The *physical time* is the time between the disk driver dispatching a request to the disk and the I/O completion event – i.e., approximately the time that the disk is busy; the *elapsed time* includes queueing delays. The values are shown in Figure 1 and Table 4. Large differences between these two times indicate that many I/Os are being queued up in the device driver waiting for previous I/Os to complete.

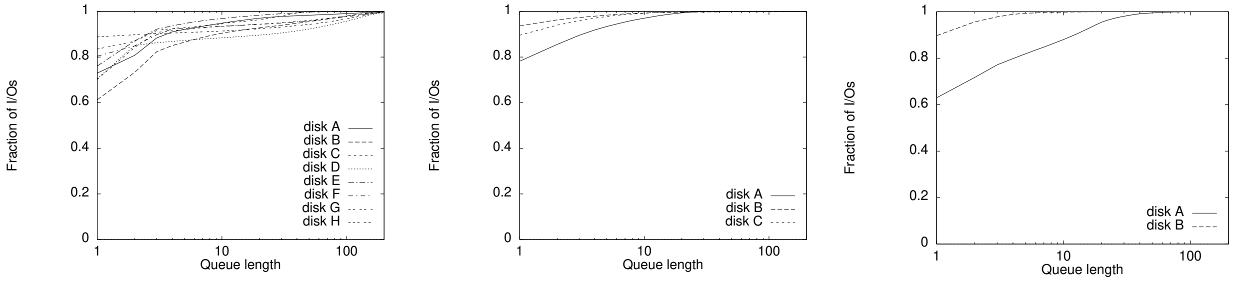
Typical causes for the difference in times include high system loads, bursty I/O activity, or an uneven distribution of load between the disks. Table 4 shows that the disparity in I/O times between elapsed and physical times is much larger for writes than for reads. This suggests that writes are very bursty. One cause is

1 Other people might add "or crashes" here, but we've never experienced a system crash in 6 years of running HP-UX on over twenty PA-RISC machines.



a. Cello **b**. Snake **c**. Hplajw

Figure 1: Distributions of physical and elapsed I/O times; see Table 4 for the mean values



a. Cello: mean = 8.9; stddev = 36.0 **b.** Snake: mean = 1.7, stddev = 3.5 **c.** Hplajw: mean = 4.1; stddev = 7.8

Figure 2: Queue length distributions for each disk in each system

the syncer daemon, which pushes delayed (buffered) writes out to disk every 30 seconds.

Snake has by far the fastest physical I/O times: 33% of them take less than 5ms. This is due to the use of disks with aggressive read-ahead and immediate write-reporting (the drive acknowledges some writes as soon as the data are in its buffer). More on this later.

The I/O loads on the disks on these systems are quite moderate – less than one request per second per disk. Nonetheless, the queue lengths can become quite large, as can be seen from Figures 2 and 3. Over 70% of the time, requests arrive at an idle disk. Some disks see queue lengths of 5 or more 15% of the time requests are added, and 5% of the time, experience queue lengths around 80 (cello) or 20 (hplajw). The maximum queue lengths seen are much larger: over 1000 requests on cello, over 100 on hplajw and snake. This suggests that improved request-scheduling algorithms may be beneficial [Seltzer90, Jacobson91].

The bursty nature of the arrival rate is also shown by Figure 4, which shows the overall arrival rates of requests, and by Figure 6, which shows request inter-arrival times. Many of the I/O operations are issued less than 20ms apart; 10–20% less than 1ms apart.

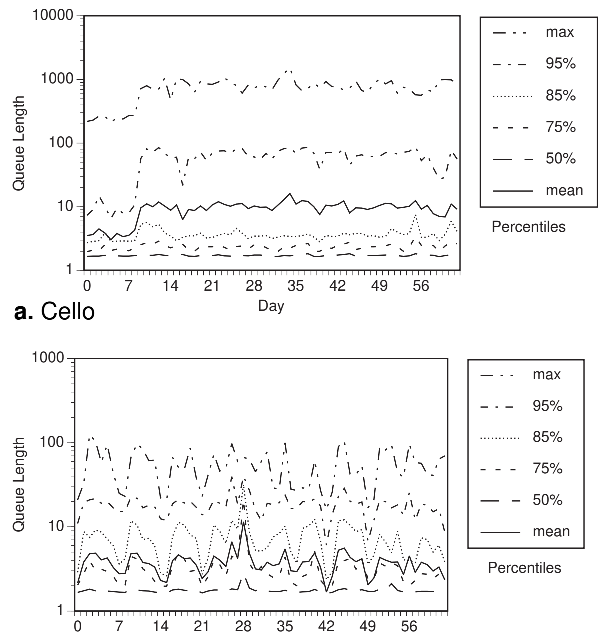
The I/O load on both snake and cello is significantly skewed, as Figure 5 shows: one disk receives most of the I/O operations on each system.

system	*I/O type*	*physical*	*elapsed*
cello	*reads*	23.6	27.4
	writes	27.7	272.0
	total	25.9	164.0
snake	*reads*	17.0	22.3
	writes	14.9	42.2
	total	15.8	33.7
hplajw	*reads*	27.5	39.2
	writes	24.1	142.0
	total	25.5	98.5

Table 4: Mean I/O request response times in ms

I/O types

Some previous studies (e.g., [Ousterhout85]) assumed that almost all accesses to the disk were for user data and therefore neglected to measure metadata and swap accesses. Our data (presented in Table 6)

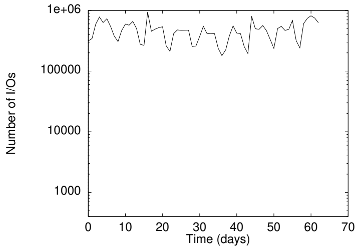


a. Cello **b.** Hplajw

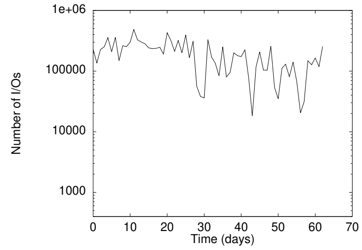
Figure 3: Mean queue length distributions versus time (daily values over 92.4.18–92.6.20)

percentile	*cello*			*snake*	*hplajw*
	A	***B***	***D***	***A***	***A***
% disk idle	73.0%	61.3%	70.5%	78.1%	63.0%
80%	2	3	2	2	5
90%	4	10	25	4	13
95%	11	43	89	7	20
99%	98	144	177	18	39
100%	606	1520	1070	111	124

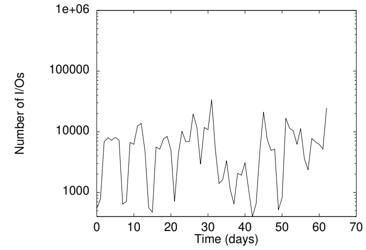
Table 5: Per-disk queue length distributions



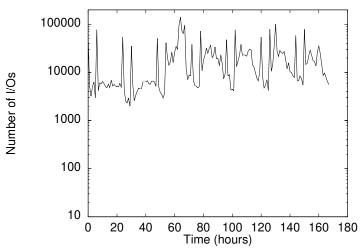
a. Daily I/O rate on cello.
Mean = 465 964 I/Os/day, or 5.4/s.



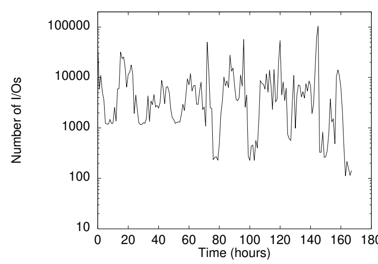
b. Daily I/O rate on snake.
Mean = 199 284/day, or 2.3/s.



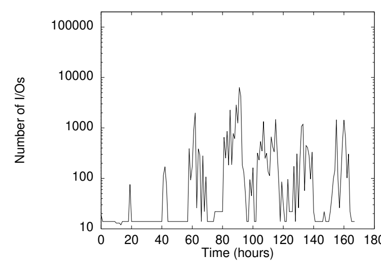
c. Daily I/O rate on hplajw.
Mean = 6607/day, or 0.076/s.



d. Hourly I/O rate on cello.
Mean = 19 422/hour.

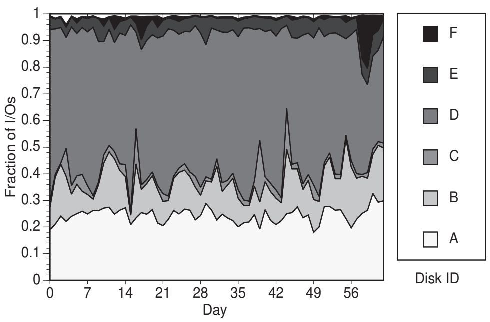


e. Hourly I/O rate on snake.
Mean = 7 158/hour.

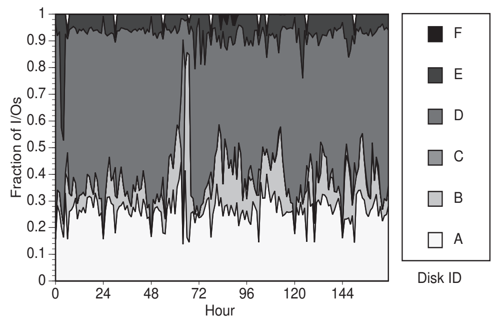


f. Hourly I/O rate on hplajw.
Mean = 265/hour.

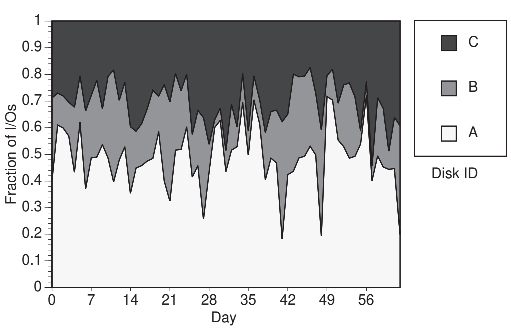
Figure 4: I/O rates for all three systems. All traces being on a Saturday; the hourly data spans 92.5.30–92.6.6



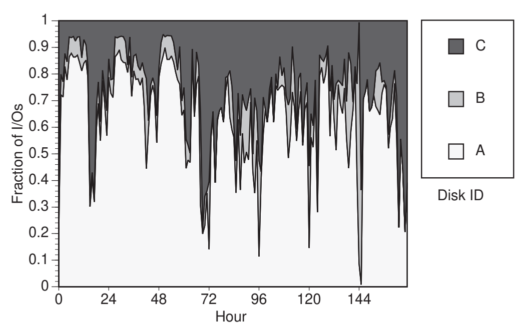
a. Cello



b. Cello (92.5.30 to 92.6.6)



c. Snake



d. Snake (92.5.30 to 92.6.6)

Figure 5: I/O load distribution across the disks for cello and snake. Disks **G** and **H** on cello are omitted, because they saw almost no I/O traffic; hplajw is omitted because almost all I/O was to the root disk.

system	operation	I/O type				Raw disk I/Os
		user data	metadata	swap	unknown	
cello	reads	51.4% (66%)	40.0% (100%)	3.2% (100%)	5.5% (100%)	8.6%
	writes	32.0% (55%)	67.3% (72%)	0.7% (100%)	0.0% (100%)	0.7%
	total	40.6% (61%)	55.2% (81%)	1.8% (100%)	2.4% (100%)	4.2%
snake	reads	5.7% (86%)	17.4% (100%)	0.7% (100%)	76.2% (83%)	34.0%
	writes	19.5% (46%)	78.1% (18%)	0.1% (100%)	2.3% (45%)	0.1%
	total	13.4% (53%)	51.5% (30%)	0.4% (100%)	34.7% (82%)	15.0%
hplajw	reads	23.6% (60%)	10.7% (100%)	29.2% (100%)	36.4% (100%)	65.7%
	writes	21.7% (26%)	71.2% (52%)	7.1% (100%)	0.0% (100%)	7.1%
	total	22.5% (41%)	45.5% (57%)	16.5% (100%)	15.4% (100%)	31.9%

Table 6: distribution of I/Os by type

In this table, *user data* means file contents, *metadata* includes inode, directory, indirect blocks, superblock, and other bookkeeping accesses, *swap* corresponds to virtual memory and swapping traffic, and *unknown* represents blocks classified as such by the tracing system (they are demand-paged executables and, on snake, diskless-cluster I/Os made on behalf of its clients). The percentages under "I/O type" sum to 100% in each row.

The amount of raw (character-mode, or non-file-system) traffic is also shown as a percentage of the entire I/Os. Raw accesses are made up of the swap traffic and the demand-paging of executables from the *unknown* category. On hplajw, there was also a small amount of traffic to the boot partition in this mode.

Numbers in parentheses represent the percentage of that kind of I/O that was synchronous at the file system level (i.e., did not explicitly have the *asynchronous-I/O* flag attached to the I/O request).

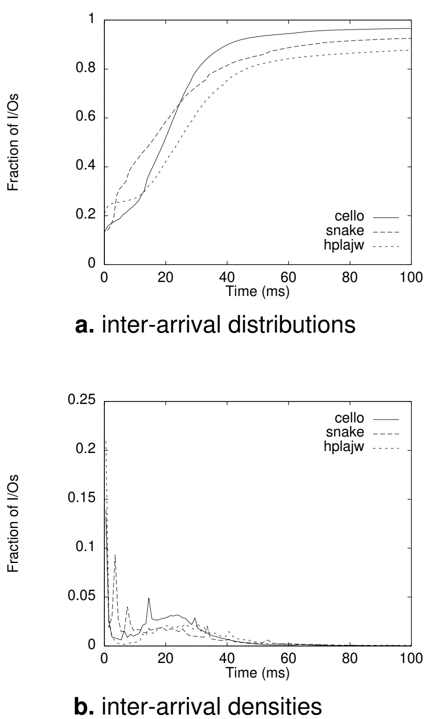


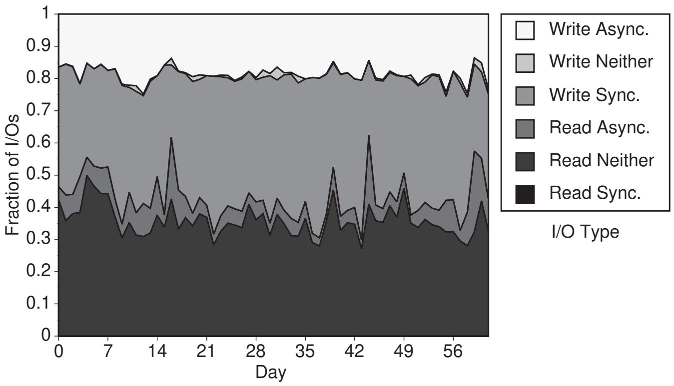
Figure 6: System-wide distribution and density plots of I/O request inter-arrival times. Cello mean: 185ms; snake mean: 434ms; hplajw mean: 13072ms

suggest that these assumptions are incorrect. In every trace, the user data is only 13–41% of all the I/Os generated in each system. The metadata percentage is especially high for writes, where 67–78% of the writes are to metadata blocks.

The ratio of reads to writes varies over time (Figure 8). The spikes in the hourly graph correspond to nightly backups. Overall, reads represent 44% (cello), 43% (snake) and 42% (hplajw) of all disk accesses. This is a surprisingly large fraction, given the sizes of the buffer caches on these systems – cello in particular. When cello's buffer cache was increased from 10MB to 30MB, the fraction of reads declined from 49.6% to 43.1%: a surprisingly small reduction. This suggests that predictions that almost all reads would be absorbed by large buffer caches [Ousterhout85, Ousterhout89, Rosenblum92] may be overly optimistic.

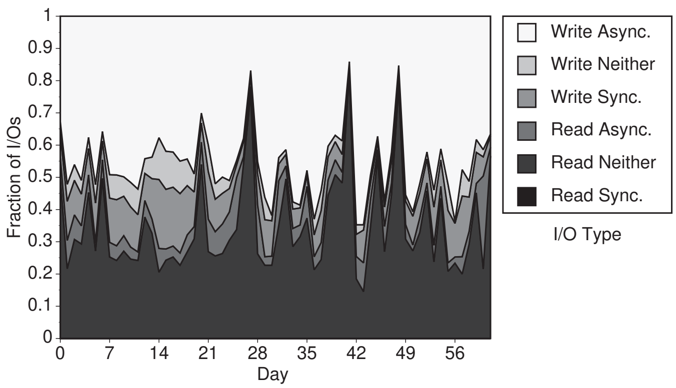
The HP-UX file system generates both *synchronous* and *asynchronous* requests. Synchronous I/O operations cause the invoking process to wait until the I/O operation has occurred before proceeding, so delaying synchronous operations can increase the latency associated with an I/O operation. Asynchronous operations proceed in the background, not tied directly to any process. Requests not explicitly flagged either way are treated as synchronous at the file system level, asynchronous at the disk level (this distinction is only important for writes).

Most read requests are implicitly synchronous (Table 6 and Figure 7), except for user data, where 14–



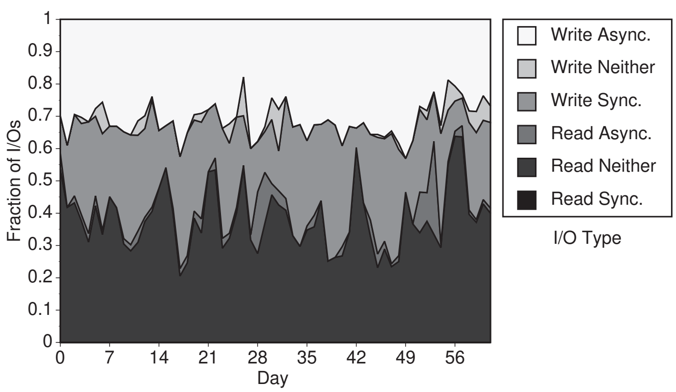
a. Cello

Mean values	*reads*	*writes*	*both*
asynchronous	7.7%	18.7%	26.4%
neither	36.4%	0.8%	37.2%
synchronous	0.0%	36.4%	36.4%
total	44.1%	55.9%	100.0%



b. Snake

Mean values	*reads*	*writes*	*both*
asynchronous	5.9%	42.6%	48.5%
neither	37.9%	4.6%	42.5%
synchronous	0.0%	9.0%	9.0%
total	43.8%	56.2%	100.0%



c. Hplajw

Mean values	*reads*	*writes*	*both*
asynchronous	4.0%	28.8%	32.8%
neither	38.4%	4.4%	42.8%
synchronous	0.0%	24.4%	24.4%
total	42.4%	57.6%	100.0%

Figure 7: I/Os on each system classified by type, expressed as a fraction of the total I/Os on that system.The labels *synchronous* and *asynchronous* indicate that one of these flags was associated by the file system with the request; *neither* indicates that the file system did not explicitly mark the request in either way. The flags do not occur together.

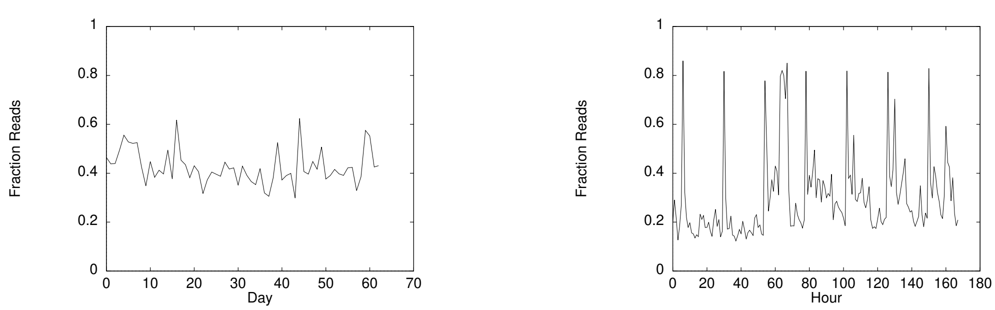
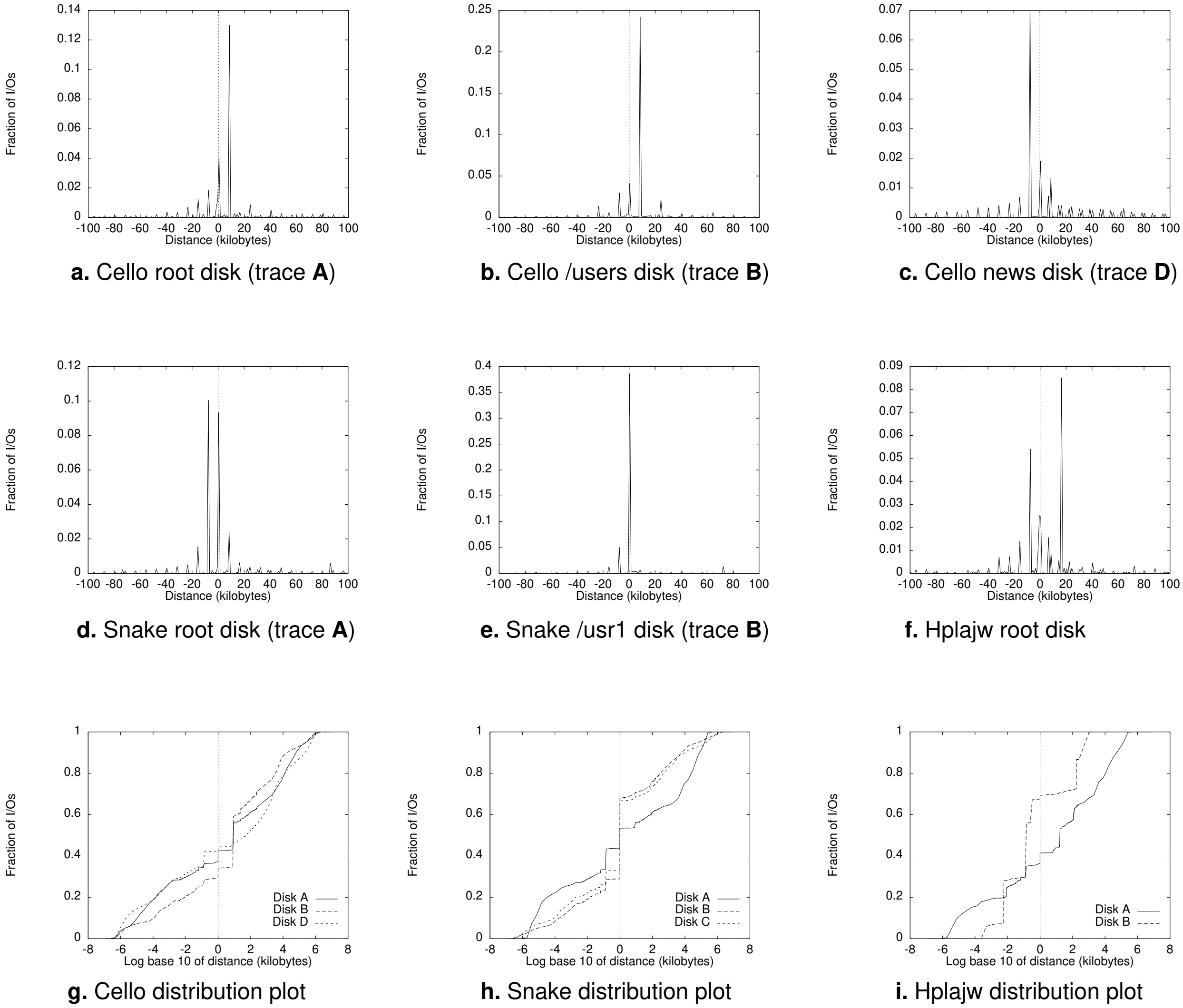


Figure 8: Fraction of I/Os on cello that are reads as a function of time (daily: 92.4.18 to 92.6.20; hourly: 92.5.30 to 92.6.6). Note the reduction in read fraction after cello's buffer cache was increased in size on 92.4.26.



a. Cello root disk (trace **A**) **b.** Cello /users disk (trace **B**) **c.** Cello news disk (trace **D**)

d. Snake root disk (trace **A**) **e.** Snake /usr1 disk (trace **B**) **f.** Hplajw root disk

g. Cello distribution plot **h.** Snake distribution plot **i.** Hplajw distribution plot

Figure 9: Density and distribution plots of the distance in KB between the end of one request and the start of the next. In the distribution plots, the X-axis is given by

$$x = \text{sign}(d) \times \log_{10}(|d|)$$

of the distance *d*. The large peaks at –8 KB correspond to block overwrites. The traces run from 92.5.30 to 92.6.6.

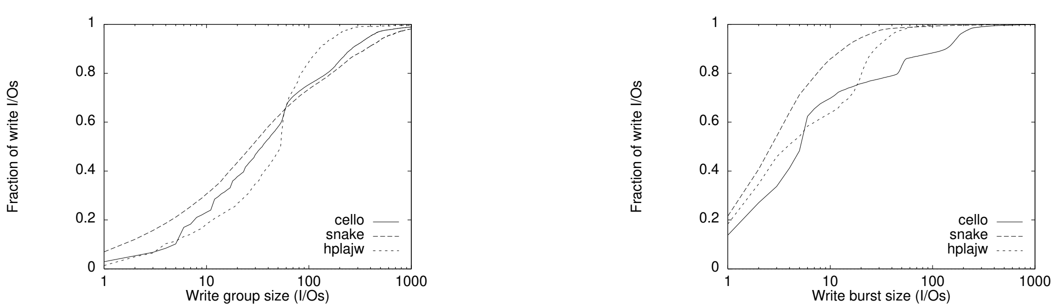
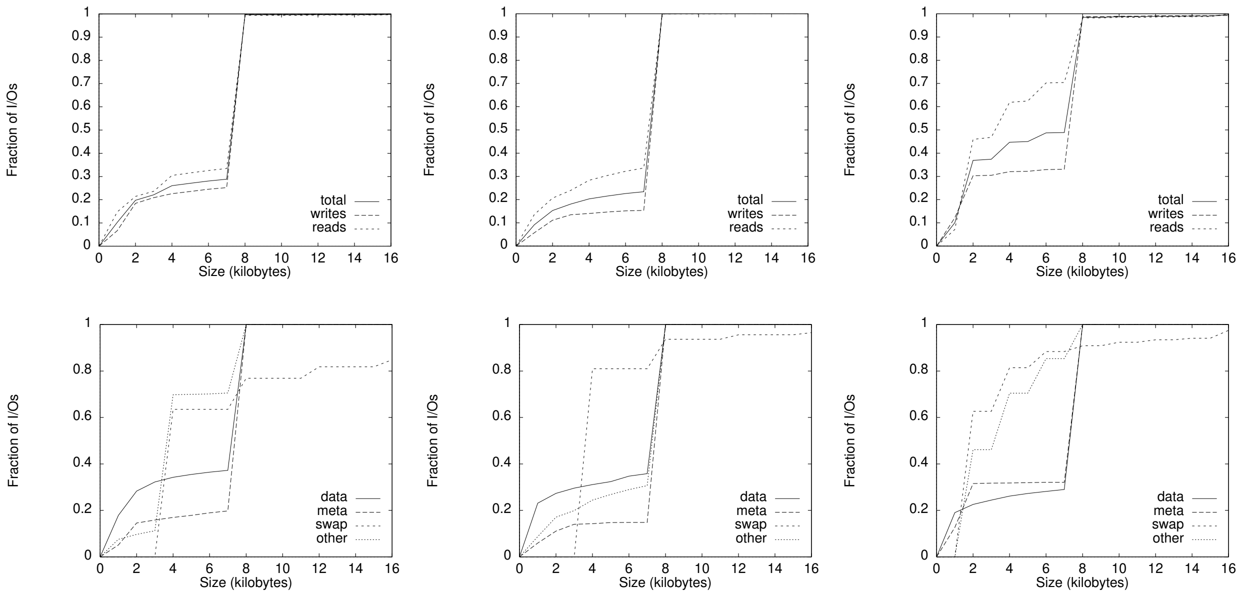


Figure 10: Distributions of *per-disk* write group sizes and write burst sizes



a. Cello **b.** Snake **c.** Hplajw

Figure 11: Distribution of transfer sizes for all systems

40% of the accesses are asynchronous read-aheads. Writes are almost all explicitly flagged as synchronous or asynchronous; again, it is user data that has the highest amount (45–74%) of asynchronous activity, except on snake, where almost all metadata writes (82%) are asynchronous. All swap and paging traffic is synchronous.

Write groups and write bursts

Looking just at the stream of write requests, we found that many of the writes occurred in *groups,* with no intervening reads. Writes rarely occur on their own: Figure 10 shows that only 2–10% of writes occur singly; 50% of them happen in groups of 20 or more (snake) to 50 or more (hplajw). Write *bursts* are almost as common. (A burst includes all those I/Os that occur closer than 30ms to their predecessor.) Most writes (60–75%) occur in bursts. On cello, 10% of the writes occur in bursts of more than 100 requests.

I/O placement

Previous studies have shown that I/Os at the file system level are highly sequential [Ousterhout85, Baker91]. But our data (plotted in Figure 9) shows that by the time these requests reach the disk they are much less so.

We define requests to be *logically sequential* if they are at adjacent disk addresses or disk addresses spaced by the file system interleave factor. There is a wide range of logical sequentiality: from 4% (on the cello news disk), 9% (hplajw root disk) to 38% (snake /usr1 disk). The means for the three systems are shown in Table 7, expressed as percentages of all I/Os.

system	*reads*	*writes*
cello	7.7%[a]	6.3%
snake	14.1%	4.7%
hplajw	4.5%	5.0%

a. 15.4% without the news disk.

Table 7: Fraction of I/Os that are logically sequential

I/O sizes

An I/O device can be accessed either through the block device or the character device. When a file is accessed via the block device, all I/O is done in multiples of the fragment size (1 KB) up to the block size (8 KB). Disks accessed via the character device (e.g., for demand paging of executables or swapping) have no such upper bound, although they are always multiples of the machine's physical page size: 2 KB for hplajw, 4 KB for the other two.

As Table 6 shows, most all accesses go through the file system, except on hplajw, where there is a large amount of swap and paging traffic (32% of the requests). Figure 11 shows how the distribution of I/O sizes varies across the systems we traced as a function of the kind of I/O being performed. As expected, file system writes are up to 8 KB in size, while swap accesses can be larger than this.

Block overwrites

We observed a great deal of block overwriting: the same block (typically a metadata block) would be written to disk over and over again. One cause of this is a file that is growing by small amounts: each time the file is extended, HP-UX posts the new inode metadata to disk – metadata is essentially held in a write-through cache.

Figure 12 plots the time between overwrites of the same block. On the root disks, 25% (hplajw and cello) to 38% (snake) of updated blocks are overwritten in less than 1 second; 45% of the blocks are overwritten in 30 seconds (cello); 18% of the blocks are overwritten at 30-second intervals (snake – presumably the syncer daemon); and over 85% of all blocks written are overwritten in an hour or less – 98% for snake.

A similar picture is told by the block access distributions shown in Figure 13. Up to 30% of the writes are directed to just 10 blocks, and 65–100% of the writes go to the most popular 1000 blocks; 1% of the blocks receive over 90% of the writes.

Together, these figures suggest that caching only a small percentage of the blocks in non-volatile memory could eliminate a large fraction of the overwrites.

Immediate reporting

The disks on snake use a technique known as *immediate reporting* for some of their writes. Our studies show that enabling it reduces the mean write time from 20.9 ms to 13.2 ms.

HP-UX's immediate reporting is intended to provide faster write throughput for isolated writes and sequential transfers. It operates as follows. An *eligible* write command is acknowledged as complete by the disk drive as soon as the data has reached the drive's volatile buffer memory. Eligible writes are those that are explicitly or implicitly asynchronous, and those that are physically immediately after the write that the drive is currently processing. The disk's write buffer acts solely as a FIFO: no request reordering occurs.

Since the data that is immediately-reported is vulnerable to power failures until it is written to disk, HP-UX disables immediate reporting for write requests explicitly flagged as synchronous.

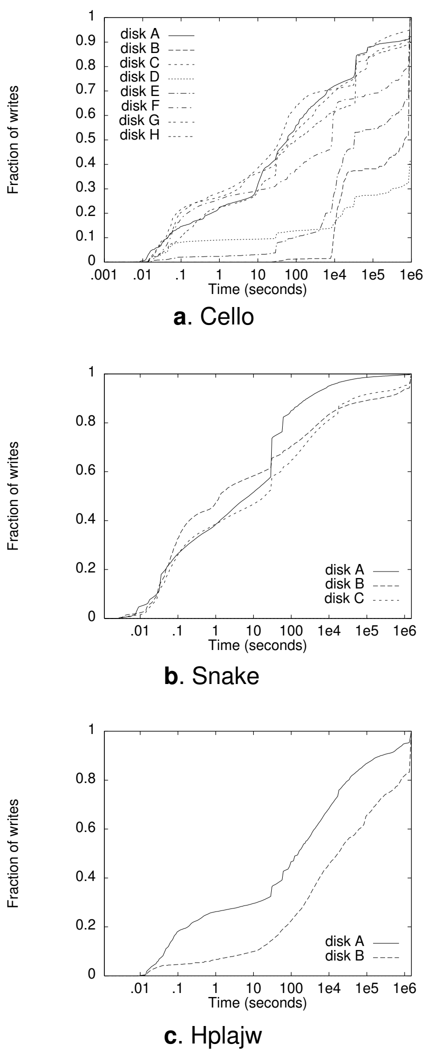


Figure 12: Distribution of 8 KB-block overwrite delays

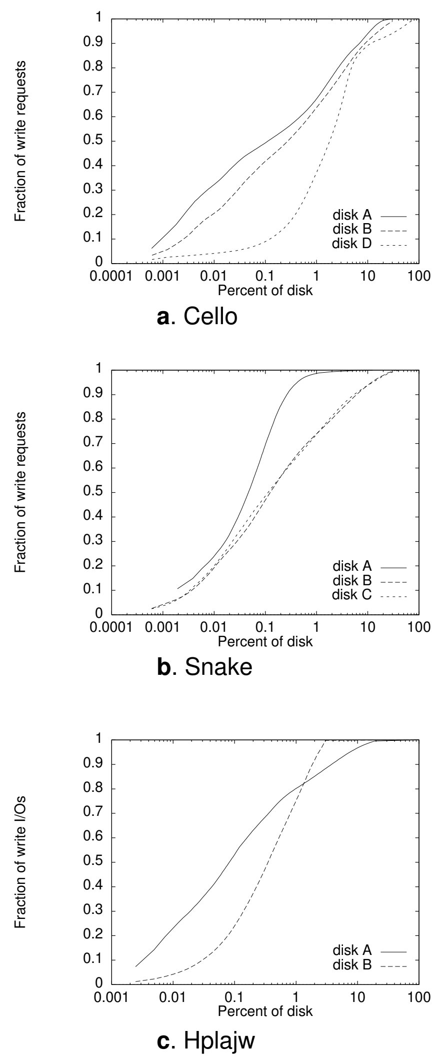


Figure 13: Distribution of writes by 8 KB block number; blocks are sorted by write access count

When immediate reporting is enabled, writes are faster, and take only 3–7ms for an 8 KB block. Figure 1 shows that 45% of all writes occur in less than 7ms. This is 53% of the writes eligible for immediate reporting (determined from Figure 7 and Table 3). The minimum physical request time is around 3ms, made up of 1.6ms of SCSI bus data transfer time, plus about the same again in disk, driver and SCSI channel overheads.

The main benefit of immediate reporting is that sequential, back-to-back writes can proceed at full disk data rates, without the need for block interleaving. (On the HP97560 disks on snake, this means 2.2 MB/sec.) However, only 4.7–6.3% I/Os are sequential writes, so the benefit is not as great as might be hoped. Perhaps caching, which allows request reordering in the disk, could help alleviate this problem. To investigate this, we turned to simulation studies driven by our trace data.

Simulation studies

We constructed a simulation model to explore various design alternatives in the I/O subsystem. We report here on our results with adding non-volatile RAM (NVRAM) caches. We begin with a description of the model itself, and the calibration we performed on it. We then present the results of applying different caching strategies.

The simulation model

We modelled the disk I/O subsystem in (excruciating) detail, including transfer times over the SCSI bus from the host, bus contention, controller overhead, seek times as a (measured) function of distance, read and write settling times, rotation position, track- and cylinder-switch times, zone bit recording for those disks that had it, media transfer rate, and placement of sparing areas on the disks.

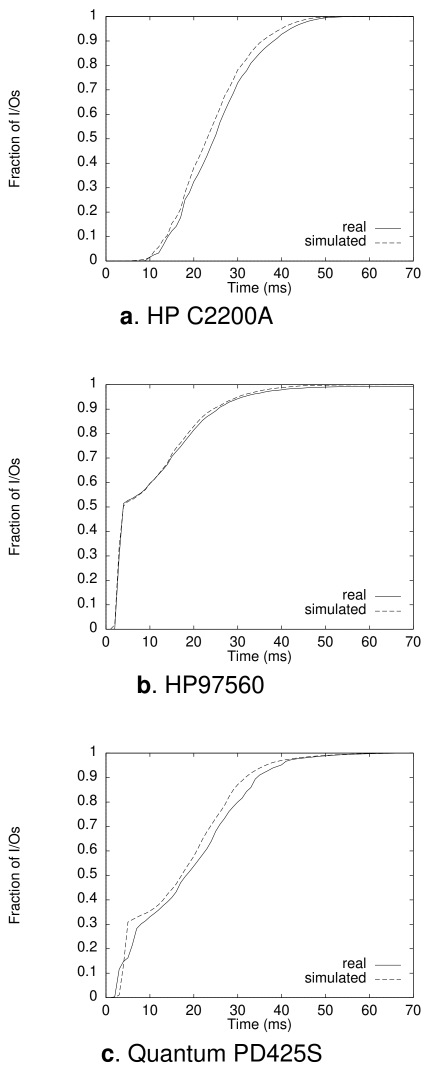


Figure 14: Measured and modelled physical disk I/O times over the period 92.5.30–92.6.6

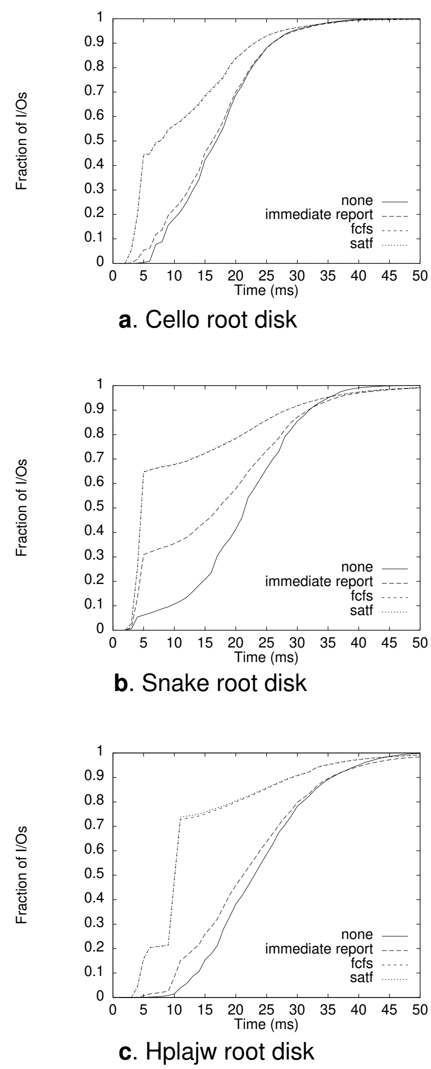


Figure 15: Distributions of physical I/O times for different disk caching policies with 128 KB of cache, over the period 92.5.30–92.6.6

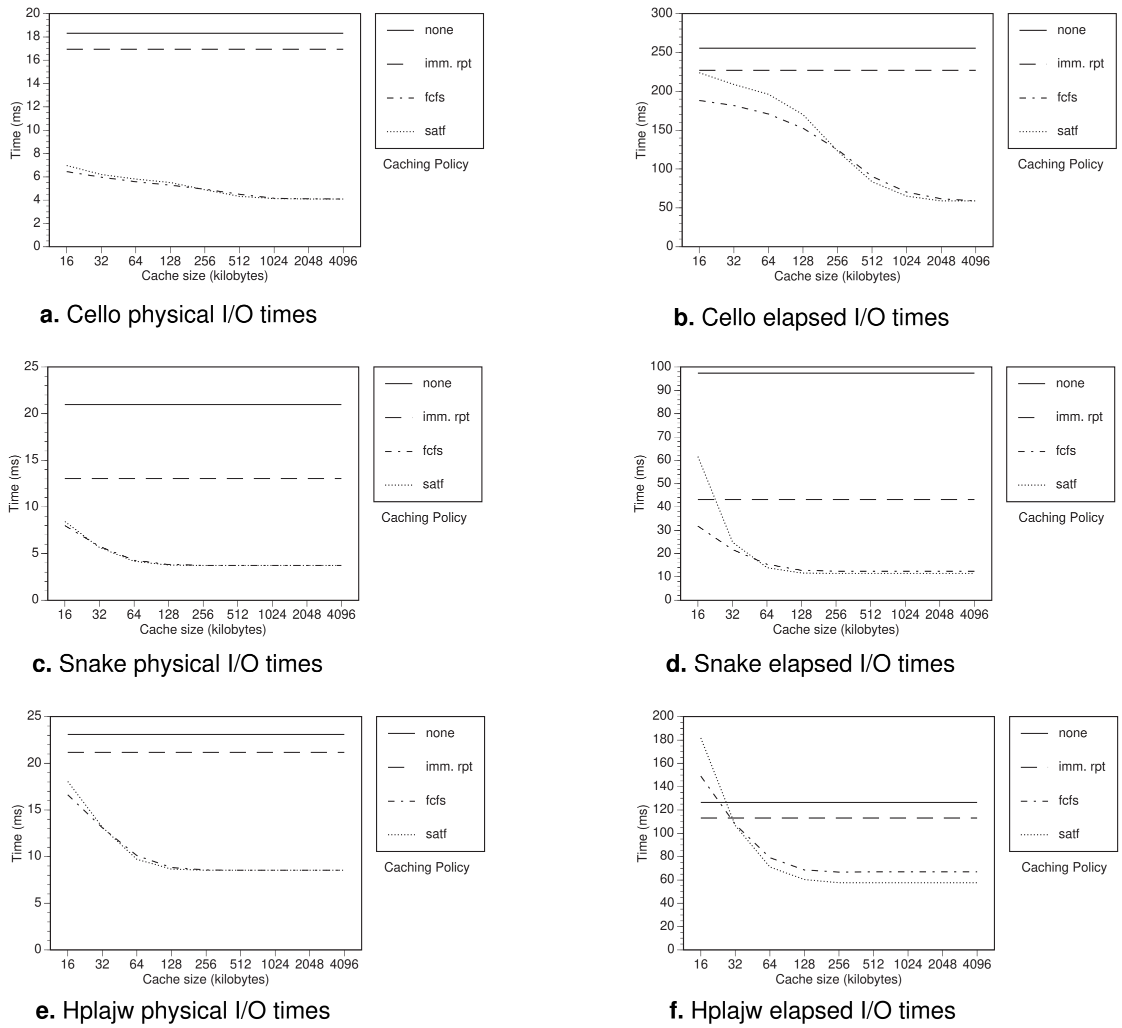


Figure 16: Physical and elapsed times for different cache sizes and caching policies. The root disk from each system is shown; traces are from 92.5.30–92.6.6.

To calibrate our model, we compared the measured I/O times in the traces against the three disks we simulated. The result is the close match shown in Figure 14. We did not attempt to mimic the two-spindle, single-controller HP2204A: instead, we modelled it as an HP2474S (but with read-ahead disabled, since the HP2204 controllers do not provide it). Since our results compare simulator runs, rather than compare simulations against the real trace (other than for the calibration), we believe the results are still useful.

Read-ahead at the disk

We did a limited exploration of the effects of in-disk read-ahead. If read-ahead is enabled, the disk continues to read sequentially into an internal read-ahead cache buffer after a read until either a new request arrives, or the buffer fills up. (This buffer is independent of the write caches we discuss later.) In the best case, sequential reads can proceed at the full disk transfer rate. The added latency for other requests can be made as small as the time for a single sector read (0.2 ms for an HP97560) in all but one case: if the read-ahead crosses a track boundary, the track switch proceeds to completion even if there is a new request to service.[2]

Table 8 shows the effects of disabling or enabling read-ahead for the cello traces. Enabling it improves physical read performance by 10% and elapsed read times by 42%, but has no effect on write times.

[2] A more recent HP disk, the HP C3010, lets the host decide whether such inter-track read-aheads should occur.

read-ahead?	*Reads*		*Writes*	
	elapsed	*physical*	*elapsed*	*physical*
no	33.7ms	16.4ms	255.4ms	18.3ms
yes	19.5ms	14.7ms	255.5ms	18.3ms

Table 8: Effect of read-ahead on I/O times, averaged over all disks on cello; 92.5.30–92.6.6

Non-volatile write caching at the disk

If non-volatile memory were used for the write buffers in the disk, the limitations of immediate reporting could be lifted: both synchronous and non-sequential writes could be cached. In particular, we were interested in how the write-back policy from the cache would affect performance.

Policies we explore here include: no cache at all, immediate reporting, caching with a straight FCFS scheduling algorithm, and caching with a modified shortest access time first scheduling (*SATF*) algorithm. (SATF is a scheduling algorithm that takes both seek and rotation position into account [Seltzer90, Jacobson91]. We modified it to favor writing large blocks out of the cache over small ones since this gave better performance at small cache sizes: it freed up space in the cache faster.)

We gave reads priority over flushing dirty buffers from the cache to disk, given the small number of asynchronous reads we saw in our traces. Each simulated disk was also given a reserved buffer for reads so that these did not have to wait for space in the write buffer. In addition, large writes (>32 KB) were streamed straight to the disk, bypassing the write buffer.

The results are presented in Figure 15 and 16, which show how the I/O times change under the different policies for the traces from the different systems.

We were surprised by two things in this data: first, there was almost no difference in the mean physical I/O times between the FCFS and SATF scheduling disciplines in the disk. In this context, FCFS is really the SCAN algorithm used by the device driver (modified by overwrites, which are absorbed by the cache). With small numbers of requests to choose from, SATF may second-guess the request stream from the host – and get it wrong – in the face of incoming read requests. At larger cache sizes, this effect is compensated for by the increased number of requests SATF can select from.

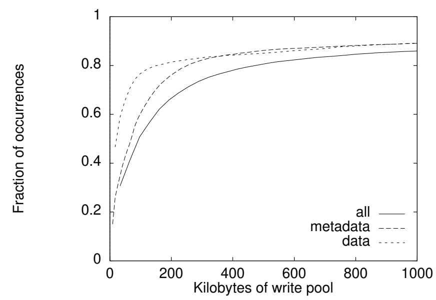
Second, even though the mean physical times were uniformly better when caching was enabled, the elapsed times for small cache sizes were sometimes worse. We tracked this down to the cache buffer replacement policy: the cache slots occupied by a request are not freed until the entire write has finished, so that an incoming write may have to wait for the entire current write to complete. At small cache sizes, this has the effect of increasing the physical times of writes that happen in bursts big enough to fill the cache – thereby accentuating the queueing delays, which occur mostly in these circumstances.

We also found that a very small amount of NVRAM (even as little as 8 KB per disk) at the SCSI controller or in the host would eliminate 10–18% of the write traffic, as a result of the many overwrites. Indeed, on snake, 44–67% of metadata writes are overwritten in a 30 second period: absorbing all of these would reduce the total I/Os by 20%.

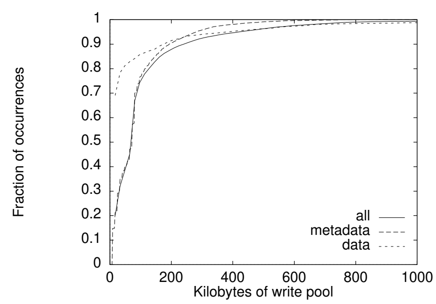
Reads were slowed down a little (less than 4%) when caching was turned on. We surmise that the increased cache efficiency increased the mean seek distance for writes by absorbing many of the overwrites. This meant that a read issued while a cache-flush write was active would occasionally have to wait for a slightly longer seek to complete than it would have done if there had been no caching. Also, reads do not interrupt a write: if this happens, the physical read time will include the time for the write to finish.

Non-volatile write caching at the host

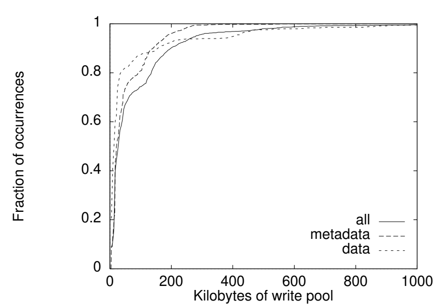
We then determined how the amount of NVRAM affected how many writes we could absorb in each 30 second interval in our traces. (We assumed no background write activity to empty the cache during the 30 second intervals.) We show this data in two ways: Figure 17 shows the distribution of 30-second intervals



a. Cello (92.5.30–92.6.6)



b. Snake (92.5.30–92.6.6)



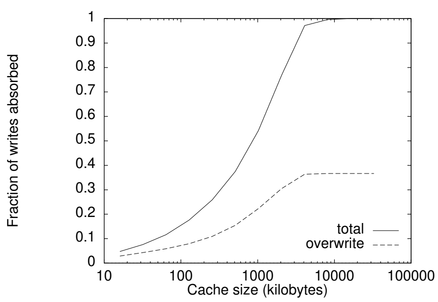
c. Hplajw (92.5.30–92.6.6)

Figure 17: Distributions of the cache sizes needed to absorb all write accesses over 30 second intervals.

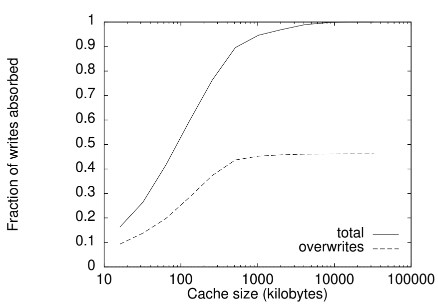
in which a given amount of cache was able to absorb *all* of the writes; Figure 18 shows the fraction of writes absorbed by a given cache size. (The disparity between the "overwrites" and "total" lines in the latter represents the amount of valid data in the NVRAM cache at the end of the 30 second period.)

Two hundred KB of NVRAM can absorb all writes in 65% (cello) to 90% (hplajw) of 30 second intervals. If metadata alone is considered (because it has the highest percentage of synchronous I/Os), all metadata writes can be absorbed in 80% of the intervals with 100 KB (hplajw and snake) to 250 KB (cello) of NVRAM.

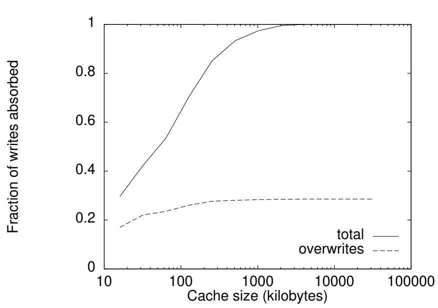
The "total" lines in Figure 18 show the write I/O bandwidth: once the cache is big enough to absorb this, little further improvement is seen. 95% absorption is reached at 700 KB (hplajw), 1 MB (snake) and 4 MB (cello). Overwrites account for 25% (hplajw) to 47% (snake) of all writes.



a. Cello (92.5.30–92.6.6)



b. Snake (92.5.30–92.6.6)



c. Hplajw (92.5.30–92.6.6)

Figure 18: distributions of the number of writes absorbed by given cache sizes over 30 second intervals.

Conclusions

We have provided a great deal of information on three complete, disk-level I/O traces from computer systems with moderately disparate workloads. These results will be of use to others in understanding what file systems do, to evaluate possible changes, and to provide distribution parameters for modelling.

We have also presented the results of simulations of write caching at the disk level, and demonstrated that this is an effective technique, although a new finding is that the write scheduling policy has little effect on the cache efficacy.

Acknowledgments

This work was carried out as part of the DataMesh research project at HP Laboratories. We thank the users and administrators of the systems we traced for their cooperation in allowing us to gather the data. David Jacobson helped us interpret some of our results.

Availability

For researchers wishing greater detail than can be reproduced here, we have made the raw data for the graphs in this paper available via anonymous *ftp* from ftp.hpl.hp.com, in the file pub/wilkes/USENIX.Jan93.tar.

References

[Baker91] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout. Measurements of a distributed file system. *Proceedings of 13th ACM Symposium on Operating Systems Principles* (Asilomar, Pacific Grove, CA). Published as *Operating Systems Review* **25**(5):198–212, 13–16 October 1991.

[Bartlett88] Debra S. Bartlett and Joel D. Tesler. A discless HP-UX file system. *Hewlett-Packard Journal* **39**(5):10–14, October 1988.

[Bozman91] 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):815–28, Sept.–Nov. 1991.

[Carson90] Scott D. Carson. Experimental performance evaluation of the Berkeley file system. Technical report UMIACS–TR–90–5 and CS–TR–2387. Institute for Advanced Computer Studies, University of Maryland, January 1990.

[Clegg86] Frederick W. Clegg, Gary Shiu-Fan Ho, Steven R. Kusmer, and John R. Sontag. The HP-UX operating system on HP Precision Architecture computers. *Hewlett-Packard Journal* **37**(12):4–22, December 1986.

[English92] Robert M. English and Alexander A. Stepanov. Loge: a self-organizing storage device. *USENIX Winter 1992 Technical Conference Proceedings* (San Francisco, CA), pages 237–51, 20–24 January 1992.

[Floyd86] Rick Floyd. Short-term file reference patterns in a UNIX environment. Technical report 177. Computer Science Department, University of Rochester, NY, March 1986.

[Floyd89] Richard A. Floyd and Carla Schlatter Ellis. Directory reference patterns in hierarchical file systems. *IEEE Transactions on Knowledge and Data Engineering* **1**(2):238–47, June 1989.

[Jacobson91] David M. Jacobson and John Wilkes. Disk scheduling algorithms based on rotational position. Technical report HPL–CSP–91–7. Hewlett-Packard Laboratories, 24 February 1991.

[Johnson87] Thomas D. Johnson, Jonathan M. Smith, and Eric S. Wilson. Disk response time measurements. *USENIX Winter 1987 Technical Conference Proceedings* (Washington, DC), pages 147–62, 21–23 January 1987.

[Kure88] Øivind Kure. *Optimization of file migration in distributed systems*. PhD thesis, published as UCB/CSD 88/413. Computer Science Division, Department of Electrical Engineering and Computer Science, UC Berkeley, April 1988.

[McKusick84] Marshall K. McKusick, William N. Joy, Samuel J. Leffler, and Robert S. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems* **2**(3):181–97, August 1984.

[Miller91] Ethan L. Miller and Randy H. Katz. Analyzing the I/O behavior of supercomputer applications. *Digest of papers, 11th IEEE Symposium on Mass Storage Systems* (Monterey, CA), pages 51–9, 7–10 October 1991.

[Muller91] Keith Muller and Joseph Pasquale. A high performance multi-structured file system design. *Proceedings of 13th ACM Symposium on Operating Systems Principles* (Asilomar, Pacific Grove, CA). Published as *Operating Systems Review* **25**(5):56–67, 13–16 October 1991.

[Ousterhout85] John K. Ousterhout, Hervé Da Costa, David Harrison, John A. Kunze, Mike Kupfer, and James G. Thompson. A trace-driven analysis of the UNIX 4.2 BSD file system. *Proceedings of 10th ACM Symposium on Operating Systems Principles* (Orcas Island, WA). Published as *Operating Systems Review* **19**(5):15–24, December 1985.

[Ousterhout89] John Ousterhout and Fred Douglis. Beating the I/O bottleneck: a case for log-structured file systems. *Operating Systems Review* **23**(1):11–27, January 1989.

[Porcar82] Juan M. Porcar. *File migration in distributed computer systems*. PhD thesis, published as Technical report LBL–14763. Physics, Computer Science and Mathematics Division, Lawrence Berkeley Laboratory, UC Berkeley, July 1982.

[Ramakrishnan92] K. K. Ramakrishnan, Prabuddha Biswas, and Ramakrishna Karedla. Analysis of file I/O traces in commercial computing environments. *Proceedings of 1992 ACM SIGMETRICS and PERFORMANCE92 International Conference on Measurement and Modeling of Computer Systems* (Newport, RI). Published as *Performance Evaluation Review* **20**(1):78–90, 1–5 June 1992.

[Rosenblum92] Mendel Rosenblum and John K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, **10**(1):26–52, February 1992.

[Satyanarayanan81] M. Satyanarayanan. A study of file sizes and functional lifetimes. *Proceedings of 8th ACM Symposium on Operating Systems Principles* (Asilomar, Ca). Published as *Operating Systems Review*, **15**(5):96–108, December 1981.

[Seltzer90] Margo Seltzer, Peter Chen, and John Ousterhout. Disk scheduling revisited. *USENIX Winter 1990 Technical Conference Proceedings* (Washington, DC), pages 313–23, 22–26 Jan. 1990.

[Smith85] Alan Jay Smith. Disk cache—miss ratio analysis and design considerations. *ACM Transactions on Computer Systems* **3**(3):161–203, August 1985.

[Staelin88] Carl Staelin. File access patterns. Technical report CS–TR–179–88. Department of Computer Science, Princeton University, September 1988.

[Staelin91] Carl Staelin and Hector Garcia-Molina. Smart filesystems. *USENIX Winter 1991 Technical Conference Proceedings* (Dallas, TX), pages 45–51, 21–25 January 1991.

[Stata90] Raymie Stata. *File systems with multiple file implementations.* Masters thesis, published as a technical report. Dept of Electrical Engineering and Computer Science, MIT, 22 May 1990.

Author information

Chris Ruemmler is currently finishing his MS degree in Computer Science at the University of California, Berkeley. He received his BS degree from the University of California, Berkeley in May, 1991. He completed the work in this paper during an internship at Hewlett-Packard Laboratories. His technical interests include architectural design, operating systems, graphics, and watching disks spin around and around at 4002 RPM. His personal interests include deep sea fishing, swimming, and music.

John Wilkes graduated with degrees in Physics (BA 1978, MA 1980), and a Diploma (1979) and PhD (1984) in Computer Science from the University of Cambridge. He has worked since 1982 as a researcher and project manager at Hewlett-Packard Laboratories. His current primary research interests are in resource management in scalable systems in general, and fast, highly available parallel storage systems in particular. He particularly enjoys working with university students on projects such as this one.

The authors can be reached by electronic mail at ruemmler@cs.berkeley.edu and wilkes@hpl.hp.com.