Measured Performance of Caching in the Sprite Network File System

Brent B. Welch
Xerox PARC
3333 Coyote Hill Road
Palo Alto CA 94304

welch@parc.xerox.com

Abstract

This paper reports on the effectiveness of the caching strategy used in the Sprite network file system based on data taken over several weeks of day-to-day usage by a variety of users. Measurements include cache consistency activity, long term I/O traffic rates, long term cache hit rates, and the averages and variations in the size of the variable-sized caches. Network traffic is compared with traffic to the local cache, and the effects of paging traffic are considered. The overall conclusion is that the caching system is quite effective and poses a low overhead. Using a delayed write strategy, 40% to 50% of the data written to client caches is never written through to a server, and less than 1% of the open operations by clients resulted in cache consistency actions by a server. †

July 19, 1991

1. Introduction

This paper presents performance measurements of the caching subsystem of the Sprite distributed file system [Ousterhout88]. In a Sprite network, client workstations are usually diskless, and the file servers implement different parts of a uniformly shared file system that provides the semantics of a 4.3 BSD UNIX timesharing system. Both clients and servers cache file data in their main memories to optimize I/O operations[Nelson88b]. Cache consistency is provided so that a read returns the most recently written data regardless of the way files are shared. A delayed write policy is used on the clients and servers, and it provides two

† This paper will appear in *Computing Systems*, the Journal of the USENIX Association.

Part of this work was done while the author was at the University of California, Berkeley, and was supported in part by the Defense Advanced Research Projects Agency under contract N00039-85-C-0269, in part by the National Science Foundation under grant ECS-8351961, and in part by General Motors Corporation.

benefits. First, applications do not have to wait for the relatively slow network and disk operations because writes occur in the background. Second, the delay period means that data written by applications can be deleted or overwritten without being written through to the file servers.

The main results presented in this paper are summarized in Table 1 and Table 2. They were obtained by monitoring the Sprite network for a period of several months. Less than 1% of the files opened triggered server consistency actions, which indicates that the consistency scheme imposes low overhead. Concurrent read sharing is quite common, mainly due to executable files. The variable-sized caches adapt to the different needs of clients and servers, with servers using more memory for their file cache than clients. The read miss rate for client caches is good, but the low client write traffic ratio is even more significant. 40% to 50% of the data written by applications is never written through to the file servers. This data is deleted or overwritten within the 30-second delay period.

These results are based on statistics taken from our Sprite network from July through December, 1989, and a follow up study made in March and April 1991. At the time of the

Summary of Caching Measurements (Fall '89)	
Files requiring consistency callbacks	< 1% opens
Files uncachable due to sharing	7.5% opens
Files concurrently read shared	37% opens
Average client cache sizes	17%-35% memory
Average server cache sizes	25%-61% memory
Average client read miss ratios	35% bytes
Average client write traffic ratios	52% bytes

Table 1. Summary of results from the initial study made during the fall of 1989. The first part of the table contains cache consistency related figures based on open operations. The second part contains gives average cache sizes. The third part gives cache effectiveness figures based on I/O rates.

Summary of Caching Measurements (March-April '91)	
Files requiring consistency callbacks	< 1% opens
Files uncachable due to sharing	< 1% opens
Files concurrently read shared	51% opens
Average client cache sizes	7%-33% memory
Average server cache sizes	40%-73% memory
Average client read miss ratios	39% bytes
Average client write traffic ratios	62% bytes

Table 2. Summary of results from a follow-up study made in the Spring of 1991. The most dramatic change is the reduction in the number of uncachable files due to the replacement of a network-wide shared database file with a server-based implementation.

original study, the Sprite network was composed of four file servers (one Sun-4/280, 2 Sun-3s, and 1 DECstation 3100) and about 30 client workstations (11 Sun-3s, 4 Sun-4s, and 15 DECstation 3100s). One year later the network had around 36 clients (6 Sun-3s, 13 Sun-4s, and 17 DECstation 3100s), the two Sun-3 servers were retired and another Sun-4 server was added for experiments. Table 3 lists the characteristics of the file servers measured in the study.

The results were obtained from raw data in the form of about 450 different statistics that were maintained in the kernel and periodically sampled. File servers were sampled hourly, and clients were sampled 5 times each day (at 8am, 11am, 2pm, 5pm, and 8pm). Some changes were made to the system after the original study, and a second set of measurements was taken to determine the effects of the changes on the system's behavior. (A 6th sample taken at 11pm was also added during the second study period.)

Sprite is 4.3 BSD UNIX compatible, although the operating system kernel has been implemented from scratch. During the study, Sprite was used for all the day-to-day computing needs of twenty or more graduate students, a few professors, and a couple of staff members. There were about a dozen more occasional users.. The workload consisted of large compilation jobs, word processing, electronic mail, simulation studies, and other assorted programming tasks. Perhaps the heaviest load on the system stemmed from development of Sprite itself. Especially during the initial study, Sprite was undergoing considerable tuning, performance enhancements, and bug fixes. Process migration was used extensively to distribute large compilation jobs among idle workstations. With migration, recompiling a large module of the kernel (e.g., the VM system or the file system) was fast enough that it was done rather frequently.

The remainder of this paper is organized as follows. Section 2 reviews the Sprite caching system and the algorithm used to maintain consistency of client caches. Section 3 presents results on the cache consistency overhead. Section 4 presents measurements of the effectiveness of the caching system during normal system activity. Section 5 shows how variable-sized caches dynamically adapt to clients and servers of different memory sizes. Section 6 describes related work. Section 7 concludes the paper.

File Server Profiles

Name	CPU	Memory	Files
Mint	Sun-3	16 Meg	Root, commands, libraries
Oregano	Sun-3	16 Meg	/tmp, commands, sources, users
Allspice	Sun-4	128 Meg	Root, commands, swap, sources, users
Assault	DS3100	24 Meg	Users
Anise	Sun-4	32 Meg	Expermental

Table 3. A profile of the file servers measured in the two studies. The memory size is the total physical memory, not the cache size. Mint was the root server in the first study. By the time of the second study, Mint and Oregano had been retired and Allspice was the server for most files. Assault continued to service a few user directories, and Anise was used for experments and scratch space.

2. The Sprite Caching System

This section reviews the Sprite caching system originally described in [Nelson88b]. The important properties of Sprite's caching system are: 1) diskless clients of the file system use their main memories to cache data; 2) clients use a delayed-writing policy so that temporary data does not have to be written to the server; and 3) the servers guarantee that clients always get data that is consistent with activity by other clients, regardless of how files are being shared throughout the network. Servers also cache data in their main memory and use delayed writes, and the implementation of the client and server caches is basically the same.

The key characteristics of the Sprite consistency scheme are: 1) the server sees all open and close operations by clients; 2) a version number is associated with each file and incremented when a file is opened for writing; and 3) a file is not cachable on clients when it is concurrently open for writing on one client and for reading and/or writing on another client. The last point is a key simplification in the Sprite caching system: if a file is concurrently write-shared by different clients, I/O operations on that file bypass the client caches and are serialized in the server cache.

In order to provide a consistent view of file data, the file servers track open and close operations and keep state about how their files are being cached by clients. Servers issue cache control messages to clients at open time, if needed, so that clients always get the most up-to-date file system data. A file server issues a write-back command to client-A if client-B opens a file and client-A has the most recent version of the file still dirty in its cache. Because clients use a delayed write policy, this can occur if a file is generated on one client and used by another within the 30-second aging period. A file server issues a disable caching command to client-A if client-B opens a file for writing and client-A still has the file open. The result of an open operation indicates to the opening client whether or not it can cache the file. In addition, servers increment a per-file version number each time the file is opened for writing, and clients use the version number to detect stale data in their cache.

The caching scheme is based on the assumption that concurrent write-sharing is rare, although there was one heavily shared file in our network. A host load database was maintained using a shared file for the database. A daemon process on each host updated the database once a minute with the host's weighted load average and the time since last keyboard input. The database was consulted in order to choose idle hosts for the targets of process migration. Because the database was continuously open for writing by a daemon process on every client, it was in an uncachable state. This was done by design to prevent the database from moving among client caches in response to updates and queries. The effects of this database will be described in Sections 3 and 4.

The use of delayed writes reflects a tradeoff between reliability and performance. Sprite has a recovery system that recovers from server and client failures [Welch89]. A cooperative recovery protocol is used after a server reboots in which clients help the server rebuild its state about files cached on the clients. Under normal circumstances, processes on clients can continue to use open files after the recovery protocol completes. Network partitions can lead to diabolical conflict cases, but these are detected by the recovery protocol. With delayed write caching, however, a power failure on a client can result in the loss of recently generated data. This is no worse than a timeshared UNIX system because UNIX also uses a 30-second delay period before writing data to the local disk. To guard against this, our editors and source code control programs use a system call to force files through to the server's disk. There is still a large amount of temporary data that is deleted before being written back to the server, as shown in Section 4.

3. Sharing and Consistency Overhead Measurements

The amount of file sharing and the consistency-related traffic was measured on the file servers by instrumenting the procedure that checks cache consistency and issues callbacks to clients. The results from the initial 36-day study are given in Table 4. The table is organized to show both hourly rates and per-server rates. The tables are broken down into different time periods based on the time data was collected, where each row indicates the activity over the preceding interval (e.g. 8pm to 8am, 8am to 11am, and so on). The per-server and combined rows shows the results averaged over the whole study period. The large number of opens at night result from the nightly dumps. The various cases in the table are explained below. Note that the measurements in this section are in terms of files opened, not bytes transferred. Measurements presented in the next section indicate how much I/O traffic there is to uncachable files and what the cache hit ratios are.

Non-File

This value indicates the number of directories, symbolic links, and swap files that were opened. These files are not cached on the clients. Swap files are not cached so that VM pages really leave the machine upon page-out. Directories and links are not cached on clients because servers do all pathname evaluation [Welch86].

Can’t Cache

This value indicates the percentage of files opened that could have been cached on

File Sharing and Cache Consistency Actions (Nov-Dec ’89)

Hour or	Num	Non-	Can’t	Read	Last	Server Action	
Server	Opens	File	Cache	Sharing	Writer	Write-back	Invalidate
8	6,932,578	35%	7.18%	39%	12%	0.34%	0.14%
11	1,252,039	20%	8.71%	34%	9%	0.19%	0.13%
14	1,800,274	17%	7.86%	40%	10%	0.35%	0.22%
17	2,857,620	19%	7.45%	34%	10%	0.42%	0.31%
20	1,940,819	17%	7.85%	30%	13%	0.26%	0.20%
Mint	6,976,180	10%	14.69%	44%	12%	0.17%	0.07%
Allspice	4,784,280	46%	0.04%	27%	12%	0.58%	0.01%
Oregano	1,746,430	38%	0.98%	67%	9%	0.03%	0.89%
Assault	486,320	34%	0.33%	17%	3%	1.52%	0.08%
Combined	13,993,210	27%	7.47%	37%	11%	0.34%	0.15%

Table 4. Cache consistency statistics including the number of files opened, read sharing, reuse of dirty files, and cache consistency actions. The top-half of the table gives an hourly breakdown of all the servers combined. Each row summarizes the activity in the interval before the time listed in the first column. The bottom half gives the total breakdown for each server individually. The last row has the totals for all the servers combined, which represents the total file system traffic. The data was taken over a 36-day period in the fall of 1989.

clients but were not cachable because of concurrent write sharing. The large amount of sharing measured on Mint, 15%, is due to the shared host load database. The other servers see very few concurrently write shared files.

Read Sharing

This value counts the number of files that were open for reading by more than one process at a time, either on the same or different clients. This case is relatively frequent because of shared executable files; it happens in about 37% of the cases. It is more frequent on Mint and Oregano, the servers for the commands directories.

Last Writer

This value counts the files that were written to a client's cache and then re-read or re-written by the same client before the 30-second delayed write period expired. File servers do not issue write-back commands in this case. Each of the servers, except Assault, sees a significant amount of this case, about 11% overall. This percentage is quite close to the percentage of files open for writing and suggests that most data is re-read or re-written shortly after it is generated. (During this period, 85.2% of all opens were read-only, 9.3% of all opens were write-only, and 5.5% of all opens were for read-write access. See [Welch90] Appendix B, Table B-3.) Mint, for example, has log files that can be repeatedly updated by the same client. Oregano serves ‘‘/tmp’’, and compiler and editor temporaries account for the reuse of dirty files. Allspice has the system source directories, and compiler output usually gets re-read by the linker. Assault is too lightly loaded to experience much of this behavior.

Server Action

This value indicates how often the servers had to issue cache control messages. ‘‘Write-back’’ indicates how many times the last writer of a file was told to write its version back to the file server. ‘‘Invalidate’’ indicates how many clients had to stop caching a file they were actively using because it became concurrently write-shared after it was opened. Write-backs happen in less than 1% of the cases, which indicates that sequential write-sharing (within the delay period) between clients is rather rare. Invalidations are also rare, except on Oregano as described below. These measurements are consistent with trace data studied by Thompson [Thompson87] who found relatively little write sharing among different users.

Two anomalies stand out in Table 4. The first is that almost 15% of the files opened on Mint were uncachable files. After some sleuthing, this value was traced to the host load database. While the daemons that periodically update the database keep the file open, some other process apparently opens the database periodically as part of a query.

The second anomaly in Table 4 is the relatively large number of invalidation commands issued by Oregano. These are due to a temporary file used by pmake, our parallel compilation tool that uses process migration. Pmake generates a temporary file containing the commands to be executed on the remote host. Initially, this file is cached on the host running pmake. During migration it is open by both the parent (pmake) and the child (a shell that will execute the commands on the remote host). These processes share a read-write I/O stream that the parent used to write the file and the child will use to read it. When the child migrates to the remote host the file server detects this as a case of concurrent write sharing and issues a write-back and invalidate command to the host running pmake. If the parent closed the file before the migration this would appear as sequential write sharing and contribute to the ‘‘Write-back’’ column instead.

Sprite Caching

Since the first study was made, the function of the load average database was reimplemented by an active server process, or pseudo-device. A pseudo-device is a special file that represents a server process; all file operations on the pseudo-device are forwarded to the server process by the kernel [Welch88]. The server can make more intelligent choices for migration, plus its interface is more efficient [Douglis90]. Previously a process had to fetch the whole database over the network in order to select a host for migration (recall that the database was uncachable). With a centralized server making host selections, host selection only requires a single query. Furthermore, the server selects the same idle hosts again and again, and this tends to increase the effectiveness of those clients' file caches. Table 5 shows consistency related statistics after this change. There is still a small rate of opens to uncachable files. These are from other shared databases such as the user login database, and from shared system log files that occasionally get appended to by multiple clients simultaneously.

4. Measured Effectiveness of Sprite File Caches

This section presents results on the I/O traffic of the clients and servers, and it shows how effective the caches are during normal system use. Traffic between applications and the cache is compared with network traffic, a breakdown of the network traffic is given, and the traffic to the servers' caches is compared with the servers' traffic to their disks.

4.1. Client Read Traffic

This section compares I/O traffic from applications to the cache with network traffic generated by the clients. The tables present I/O rates in bytes per second, the miss ratio of the

New File Sharing and Cache Consistency Actions (Mar-Apr '91)

Hour or	Num	Non-	Can't	Read	Last	Server Action	
Server	Opens	File	Cache	Sharing	Writer	Write-back	Invalidate
8	5,071,637	30%	0.13%	44%	1%	0.61%	0.07%
11	1,256,606	16%	0.26%	54%	3%	0.52%	0.08%
14	2,209,775	13%	0.43%	53%	6%	0.68%	0.23%
17	2,200,846	14%	0.63%	46%	5%	0.74%	0.28%
20	1,964,212	11%	0.44%	48%	5%	0.70%	0.25%
23	1,937,916	9%	0.33%	54%	3%	0.37%	0.18%
Allspice	11,175,500	13%	0.33%	55%	2%	0.42%	0.17%
Anise	896,419	37%	0.02%	4%	9%	1.78%	0.01%
Assault	1,004,500	49%	0.14%	35%	9%	1.43%	0.03%
Combined	13,076,419	18%	0.30%	51%	3%	0.59%	0.15%

Table 5. Consistency data from Allspice, Assault, and Anise during a 36-day study in March and April 1991. Allspice is the primary server, while Assault and Anise store user files. The top half of the table summarizes the activity for all servers during the interval before the time listed in the first column. The bottom half summarizes the activity for each server individually. The last row has the totals for all the servers combined.

cache, and a breakdown of the network traffic in terms of cache misses, uncachable data, and paging data from the VM system. Regular files on the file server are used for VM backing store, but paging traffic on the client bypasses the client's file system cache so that data is not cached by both the VM system and the file system. There are two percentages associated with cache misses. The first percentage is the miss ratio, which is computed as follows:

$$Miss\ \ Ratio\ =\ \frac{M\ +\ U}{C\ +\ U}$$

Where *M* is the rate that data is fetched into the cache because of misses, *U* is the rate that uncachable data is read, and *C* is the rate that data is read from the cache by applications. The fact that *C* does not include *U* is because uncachable traffic bypasses the cache and the rates were monitored separately. In the tables below, the I/O rate in the column labeled ‘‘Cache’’ is for *C*, not *(C+U)*.

The second percentage associated with cache misses is its contribution to the total network traffic:

$$Miss\ \ Traffic\ =\ \frac{M}{M\ +\ U\ +\ V}$$

where *V* represents paging traffic from the VM system. The tables also give the proportion of network traffic made up by uncachable data and VM data, as well as the total network I/O rates.

Tables 6 through 8 give the read traffic for the DECstation and Sun-3 clients. The tables are broken down into different time periods based on the time data was collected, where each row indicates the activity over the preceding interval (e.g. 8pm to 8am, 8am to 11am, and so on). The bottom row shows the results averaged over the whole study period. The ‘‘Cache’’ column gives the read rate from the cache (not counting reads to uncachable data), and the

DS3100 Client Read Traffic (Bytes/Seconds and ratios) (Nov-Dec ’89)

Hr	Cache	Misses			Uncached		PageIn		Total
8	286	21%	52	28%	10	5%	124	66%	188
11	469	62%	289	65%	10	2%	139	31%	439
14	490	37%	173	40%	16	3%	239	55%	430
17	897	27%	232	58%	26	6%	138	34%	398
20	988	47%	463	83%	12	2%	81	14%	557
TL	570	34%	188	62%	12	4%	100	33%	302

Table 6. Hourly read traffic for DECstation 3100 clients, which have 24 Meg main memories. The first column indicates the time of day data was taken, and the other columns have I/O rates and relative percentages. The last row averages the data over the whole trace period. I/O rates are given in bytes/second. The first percentage in the ‘‘Misses’’ column is the cache miss rate. The second percentage is the relative proportion of cache miss traffic to other sources of network traffic. The ‘‘Uncached’’ and ‘‘PageIn’’ columns gives rates for traffic to uncachable files and traffic to swap files, respectively. The percentages in these columns indicate their relative proportion of the total network read traffic. The ‘‘Total’’ column gives the total network read traffic.

Sun-3 (12 Meg) Client Read Traffic (Bytes/Seconds and ratios) (Nov-Dec ’89)									
Hr	Cache	Misses			Uncached		PageIn		Total
8	132	21%	23	24%	6	6%	64	68%	94
11	566	22%	104	24%	29	6%	290	68%	425
14	887	32%	270	46%	30	5%	280	47%	584
17	969	25%	229	56%	20	5%	153	37%	405
20	678	23%	147	51%	14	5%	126	43%	288
TL	434	35%	144	43%	15	4%	167	51%	328

Table 7. Hourly read traffic for Sun-3 clients with 12 Meg main memory.

Sun-3 (8 Meg) Client Read Traffic (Bytes/Seconds and ratios) (Nov-Dec ’89)									
Hr	Cache	Misses			Uncached		PageIn		Total
8	145	45%	57	40%	15	10%	68	47%	142
11	205	43%	77	21%	23	6%	249	69%	360
14	485	40%	179	37%	25	5%	262	55%	475
17	725	41%	283	36%	25	3%	410	52%	778
20	522	42%	215	46%	14	3%	228	49%	466
TL	322	40%	122	37%	14	4%	185	55%	330

Table 8. Hourly read traffic for Sun-3 clients with 8 Meg main memory.

‘‘Misses’’ column gives the rate at which the cache requested data from a server. The first percentage under ‘‘Misses’’ is the miss ratio defined above, while the second percentage is the ratio of cache misses to all network read traffic. The other two primary sources of network read traffic, uncachable files and page faults, are listed in the columns labeled ‘‘Uncached’’ and ‘‘PageIn’’. The last column gives the total network I/O traffic. Some of the network I/O traffic (up to 1% or 2%) is due to remote device and remote window access, which is not shown in the table.

The overall read miss rates are around 35%, with the 8 Meg Sun-3s slightly worse at a 40% miss rate. The hourly average miss rates range from about 20% to 60%, with lower percentages indicating more effective caches. Note that VM paging traffic accounts for slightly more network read traffic than cache misses. The VM traffic includes page faults on program image files, as well as faults on swap files. The DECstations, which all have 24 Meg of main memory, have the lowest paging traffic. Note that the workstations with larger memories have larger read rates to their cache, but all the workstations have about the same overall network read traffic. The larger memories reduce paging and allow for larger, more effective file caches.

Initial measurements of the read traffic highlighted a number of clients with abnormally high traffic to uncachable data. The traffic for these clients is given in Table 9. Their poor miss rate, almost 60%, was due to an X widget application that displayed the number of hosts currently available for migration. Every 15 seconds the entire database was scanned to count up the available hosts, and the effect on network traffic was significant. The process migration system has been changed to use a server process to manage host selection instead of using a

Abnormal Client Read Traffic (Bytes/Seconds and ratios) (Nov-Dec '89)									
Hr	Cache	Misses			Uncached		PageIn		Total
8	265	62%	55	13%	290	70%	66	15%	414
11	653	48%	164	20%	293	36%	336	42%	797
14	609	52%	170	23%	306	42%	244	33%	725
17	1625	34%	345	24%	336	24%	705	50%	1391
20	980	49%	332	42%	294	37%	158	20%	788
TL	516	57%	143	22%	358	56%	128	20%	633

Table 9. Hourly read traffic for a collection of DS3100 and Sun-3 clients that have abnormally high read traffic to uncachable data. The traffic stems from an application that periodically scans a heavily shared (and therefore uncached) database.

shared file. The server can return the number of available hosts with a single, small query. Table 10 shows the read traffic for a collection of DECstation and Sun-4 clients during the second study, after this change was made. There is almost no read traffic to uncachable data, and VM paging traffic dominates the network. The increase in page-in traffic was traced to a change in the way initialized data pages of executable programs are handled. These pages have to be reloaded each time a program is executed. Previously they were copied into the file system cache at the time they were first faulted into memory so that future executions of the same program would find the initialized data in the local cache. This feature was simplified away, but because it causes a significant increase in page-in traffic it has since been reintroduced.

4.2. Client Write Traffic

Tables 11 through 13 give the write traffic for the DECstation and Sun3 clients. The format of the tables is similar to those for read traffic. Hourly breakdowns are given, and remote

New Client Read Traffic (Bytes/Seconds and ratios) (Mar-Apr '91)									
Hr	Cache	Misses			Uncached		PageIn		Total
8	115	40%	45	15%	1	0.7%	242	83%	290
11	257	38%	93	6%	9	0.7%	1310	92%	1413
14	593	40%	235	15%	12	0.8%	1241	83%	1489
17	1093	45%	490	24%	15	0.8%	1456	74%	1962
20	695	41%	282	19%	7	0.5%	1165	80%	1455
23	475	42%	198	11%	8	0.5%	1519	88%	1725
TL	301	39%	117	14%	4	0.6%	673	84%	794

Table 10. Read traffic for a collection of DECstation and Sun-4 clients after the shared database was replace by a server process. There is almost no read traffic to uncachable data, and VM paging traffic dominates the network traffic.

DS3100 Client Write Traffic (Bytes/Seconds and ratios) (Nov-Dec '89)									
Hr	Cache	WriteBack			Uncached		PageOut		Total
8	146	48%	63	49%	14	11%	48	38%	127
11	174	41%	69	46%	5	3%	74	49%	150
14	270	55%	146	65%	5	2%	69	31%	222
17	530	44%	230	73%	6	2%	74	23%	314
20	333	47%	154	80%	5	3%	29	15%	190
TL	244	51%	120	63%	10	5%	57	30%	189

Table 11. Hourly write traffic for all the DECstation 3100 clients. The format of the table is the same as Table 6. An uncached database is updated once a minute by each host, and this creates a small amount of uncachable I/O traffic. Note that the overall write miss rate is 51%, meaning that half the data written to the cache is not written out.

Sun-3 (12 Meg) Client Write Traffic (Bytes/Seconds and ratios) (Nov-Dec '89)									
Hr	Cache	WriteBack			Uncached		PageOut		Total
8	69	52%	30	53%	11	20%	13	23%	57
11	148	50%	71	49%	7	5%	61	43%	143
14	267	57%	150	54%	8	3%	79	28%	273
17	327	53%	170	67%	7	3%	70	28%	251
20	239	37%	84	55%	7	4%	56	37%	151
TL	148	52%	74	53%	8	6%	45	32%	140

Table 12. Hourly write traffic for all the 12 Meg Sun-3 clients. The write traffic is similar to that of the DECstation clients.

Sun-3 (8 Meg) Client Write Traffic (Bytes/Seconds and ratios) (Nov-Dec '89)									
Hr	Cache	WriteBack			Uncached		PageOut		Total
8	86	53%	38	46%	15	19%	27	33%	83
11	134	61%	77	46%	11	6%	77	46%	167
14	186	53%	94	49%	11	6%	82	43%	190
17	354	69%	244	67%	9	2%	104	29%	360
20	172	59%	98	41%	9	3%	125	53%	236
TL	154	59%	85	49%	12	7%	73	42%	173

Table 13. Hourly write traffic for all the 8 Meg Sun-3 clients. The write miss rate is slightly worse (higher) in comparison with the DECstation and 12 Meg Sun-3 clients.

traffic is divided among cache misses, uncachable data, and writes due to page outs. The effectiveness of the cache in trapping short lived data is given by the traffic ratio:

$$Traffic\ Ratio = \frac{W + U}{C + U}$$

Where W is the amount of data written out of the cache to a server, C is the amount of data written into the cache by applications, and U is the amount of data written to uncachable files.

Sprite Caching

The most notable result in the tables is that the writeback traffic ratio ranges from about 40% to 60%, averaging 52% overall. This means that about half the data written by applications was removed or overwritten in the 30 second aging period. Traffic to uncachable data accounts for only a few percent of the network traffic, although it does increase the miss ratio by a couple percent. Page out traffic accounts for less than half the network traffic, and it is as low as 30% of the traffic from the DECstation 3100s. Table 14 shows more recent data for the Sprite network, after the uncachable host load database was replaced with a server. There is still evidence of some shared files that are updated steadily, but the I/O rate is much smaller. Shared files are still used to record user logins and to log system events. Note that the write traffic ratio is 62%, which is worse than the 50% found in the initial study. Note also that the I/O rates to the cache are lower during this study, suggesting that the network was being used less intensively. The initial study was made during an intense period of system development, and large compilation jobs generated much of the I/O traffic. Compiler temporary files are created and deleted before being written out, so they contribute favorably to the write traffic ratio. Traffic ratios as low as 30% have been observed during large compilations.

4.3. Server I/O Traffic

This section presents the I/O traffic from the standpoint of the file servers. In the case of a file server it is interesting to compare the traffic to its cache to the traffic to its disks. Two metrics are given, the ‘‘File Traffic’’ and the ‘‘MetaData Traffic.’’ *Metadata* is data on the disk that describes a file and where it lives on disk. This includes the descriptor that stores the file’s attributes, and the index blocks used for the file map. The ‘‘File Traffic’’ represents I/O to file data blocks as opposed to the metadata information. The combination of file traffic and metadata traffic gives the total disk traffic for the file server. Three ratios are given that

New Client Write Traffic (Bytes/Seconds and ratios) (Mar-Apr ’91)									
Hr	Cache	WriteBack			Uncached		PageOut		Total
8	36	96%	34	62%	1	2.6%	15	28%	55
11	160	50%	79	42%	1	0.8%	65	34%	188
14	174	74%	129	46%	1	0.5%	107	38%	276
17	350	74%	260	52%	10	2.0%	178	35%	497
20	233	79%	185	58%	1	0.4%	104	32%	318
23	170	85%	145	54%	1	0.5%	95	36%	264
TL	122	62%	75	55%	1	1.4%	45	33%	134

Table 14. Hourly write traffic after the shared host load database was replaced with a server process. The clients are a collection of Sun-4s and DECstations. The rate of I/O to uncachable files has dropped significantly, but there is still evidence of uncachable files, most likely system log files and the user login database. There are two anomalies in this table. First, there was one spike of uncachable traffic on three clients which shows up under hour 17 that may been an artificial test. Second, the percentages only total approximately 90% of the network traffic. The missing data stems from remote access to devices and user-level server processes. In particular, one user-level server acts as a gateway to remote NFS servers, and a number of long-running simulations directed their output to NFS files.

compare cache traffic to disk traffic:

$$File\ \ Traf\!f\!ic\ \ Ratio\ =\ \frac{F}{C}$$

$$Metadata\ \ Traf\!f\!ic\ \ Ratio\ =\ \frac{M}{C}$$

$$Total\ \ Traf\!f\!ic\ \ Ratio\ =\ \frac{M\ +F}{C}$$

Where F is the disk traffic from the cache to file data, M is the disk traffic to metadata, and C is the traffic between applications and the cache. There are no uncachable files on a file server. The file traffic ratio ignores the effects of metadata, while the total traffic ratio includes it.

The server I/O traffic from a 20-day study period, October 29 through November 19, 1989, is given in Tabled 15 and 16. Figure 1 shows the server I/O traffic for a combination of all servers averaged over a 6-month study period, from July to December 1989. Servers were sampled every hour, 24 hours a day. The graphs indicate that the server caches are effective for reads during peak usage hours. The server caches are less effective for writes because client caches trap out most of the short-lived data. The write graph highlights the large amount of write traffic to metadata. File descriptors have to be updated with access and modify times, so just reading a file ultimately causes its descriptor to be written to disk. Furthermore, the 128-byte descriptors are written 32 at time in 4K blocks so there is extra traffic from unmodified descriptors. The large amount of metadata traffic at 2am is because the UNIX tar program is used for our nightly dumps, and it changes the access time of every directory and any files that were dumped. The effect of metadata changes has been noted by

Server I/O Traffic (Bytes/Second) (Fall '89)								
		Cache Traffic		File Traffic		MetaData Traffic		Total Disk
Host		bytes/s	(dev)	bytes/s	(dev)	bytes/s	(dev)	bytes/s
Mint	r	8601	(9321)	3427	(7876)	283	(982)	3710
	w	921	(640)	863	(507)	4133	(1913)	4996
Oregano	r	4421	(9129)	3201	(8242)	453	(1101)	3654
	w	932	(3163)	804	(2994)	1295	(1295)	2099
Allspice	r	11478	(18313)	5970	(16398)	520	(2062)	6490
	w	5174	(9495)	3838	(6142)	1692	(2013)	5530
Assault	r	1808	(7342)	1481	(7128)	180	(604)	1661
	w	529	(3081)	291	(1385)	350	(676)	641
combined	r	25946	-	13932	-	1433	-	15515
	w	7305	-	5620	-	7481	-	13266

Table 15. I/O traffic on the file servers over a 20-day period from October 29 through November 19, 1989. The upper row for each server gives read I/O rates, the lower row gives write rates. The average and standard deviation are given for the bytes/sec transferred to and from the cache (‘‘Cache’’), file data blocks on disk (‘‘File’’), from file descriptors and index blocks on disk (‘‘MetaData’’), and the total traffic to the disk (‘‘Total’’).

Server I/O Traffic (Megabytes and Ratios)								
Host		Cache	File		MetaData		Total Disk	
Mint	r	15172	6046	40%	500	3%	6546	43%
	w	1624	1523	94%	7291	449%	8814	542%
Allspice	r	18861	9811	52%	854	5%	10665	57%
	w	8503	6307	74%	2781	33%	9088	107%
Oregano	r	8199	5937	72%	839	10%	6776	83%
	w	1729	1491	86%	2402	139%	3893	225%
Assault	r	3131	2565	82%	311	10%	2876	92%
	w	917	505	55%	606	66%	1111	121%
combined	r	45364	24358	54%	2505	6%	26863	59%
	w	12772	9826	77%	13080	102%	22906	179%

Table 16. This gives the total megabytes transferred for the results given in Table 14, and the percentage that this is of the megabytes transferred to or from the cache. The total disk traffic can be greater than 100% of the cache traffic because of metadata traffic.

Hagmann[Hagmann87], who converted the Cedar file system to log metadata changes, which reduced metadata traffic considerably. Current research in the Sprite group involves a log-structured file system where all data, file and metadata, is logged [Ousterhout89] [Rosenblum91].

The tables show that the cache on Mint, the root server, is effective in eliminating reads (40% file traffic in the 20-day study), but not as good at eliminating writes (94% traffic ratio in the 20-day study). Its read hits occur on frequently-used program images and the load average database. The steady updates to this file and other logs trigger a steady update of disk descriptors, which explains Mint's high metadata traffic.

Preliminary server traffic data showed that Mint had a traffic ratio of only 50%, while the other servers had write traffic ratios of 70% to 80% or more. Mint's low traffic ratio prompted a search for a bug in the cache write-back code, and indeed it turned out that continuously updated files were never being written out to disk! The 20-day study reported in the tables was made after this bug was fixed. Mint's traffic ratio changed from 50% to 94%, which indicates that almost half the data written to it is to continuously updated files, i.e. the host load database. The 6 month study graphed in Figure 1, however, includes both versions of the system so there actually should have been slightly more file write traffic to update the host load database.

Allspice and Oregano are directly comparable because they store the same type of files (many files were shifted from Oregano to Allspice during the 20-day study period). Allspice's cache is about 10 times the size of Oregano's and it is clearly more effective. This is to be expected because the server's cache is a second-level cache, with the clients' caches being the first level. The server's caches have to be much larger than the client's caches because the locality of references to their cache is not as good.

It is also interesting to see how the caches skew the disk traffic towards writes. During the 20-day study period, the traffic to the server caches was about 22% writes. The traffic to the server disks was 26% metadata writes and 20% data writes. If the metadata traffic is discounted as an artifact, then the data writes accounted for 40% of the disk traffic. The skew

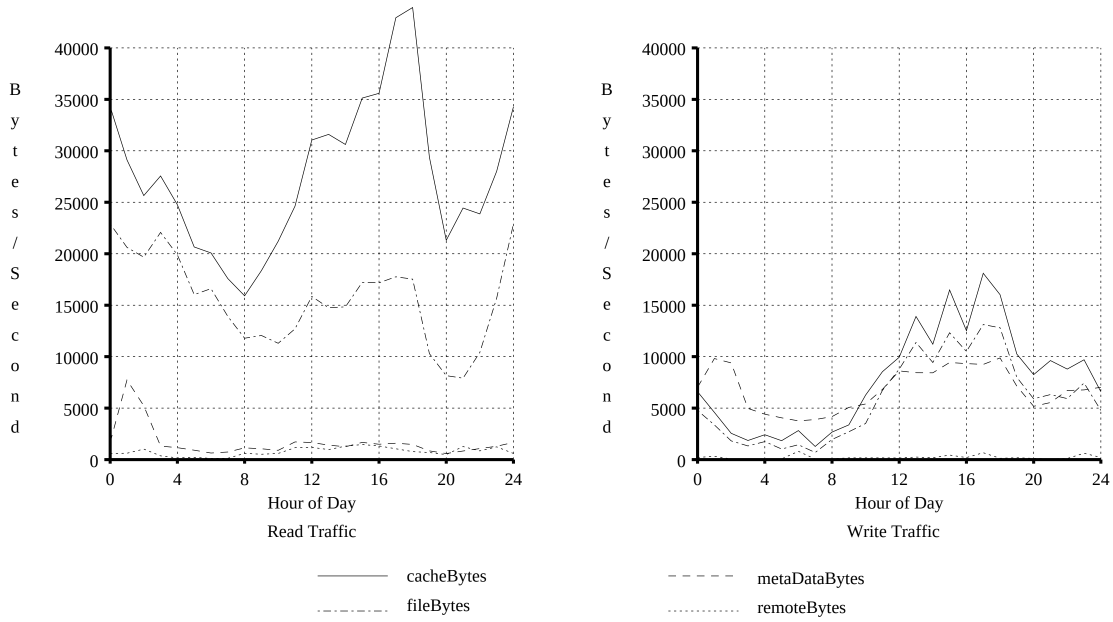

Figure 1. Server I/O traffic averaged from July 8 to December 22, 1989. The left-hand graph has read traffic and the write-hand graph has write traffic. ‘‘cacheBytes’’ are file data bytes transferred between the cache and remote clients or server-resident applications. ‘‘fileBytes’’ are file data bytes transferred between the disk and the cache. ‘‘remoteBytes’’ are file data bytes accessed remotely by server-resident applications. ‘‘metaDataBytes’’ are metadata bytes transferred to and from the disk. The total disk traffic is the sum of ‘‘fileBytes’’ and ‘‘metaDataBytes’’.

towards writes at the disk level should continue as server caches get larger and more effective at trapping reads.

5. Variable-Sized Caches

An important feature of Sprite caches is that they vary in size in order to make use of all available memory. Nelson[Nelson88a] explored ways of trading memory between the Sprite file system and the virtual memory system, which needs memory to run user programs. The basic approach Nelson developed was to compare LRU times (estimated ages) between the oldest page in the FS cache and the oldest VM page and pick the oldest one for replacement. Thus the file system cache size will grow or shrink depending on file system and virtual memory activity.

Nelson found that it was better to bias in favor of the VM system in order to reduce the page fault rate and provide a good interactive environment. The bias is achieved by adding a bias to the LRU time of the VM system so that its pages appear to be referenced more recently than they really were. We have arbitrarily chosen a bias against the file system of 20 minutes. Any VM page referenced within the last 20 minutes will never be replaced by a FS cache page. This policy is applied uniformly on all hosts, and it adapts naturally to both clients and servers. Servers use most of their memory for a file cache, while active clients use most of their memory to run user programs. Idle clients become hosts for process migration, and large idle programs (i.e. the window system) tend to get paged out and replaced by more file cache as well as the migrating applications.

5.1. Measured Cache Sizes

Table 17 gives the average and maximum cache sizes as measured over the 6-month study made in 1989. Table 18 gives the same measurements taken during the second study. The file servers are listed individually, and the clients are grouped according to the amount of physical memory they have. The adaptive nature of the cache sizes is evident when comparing clients and servers with the same memory size; the file servers devote more of their memory to the file cache. This difference is not achieved via any special cases in the implementation, but merely by the uniform application of the 20-minute bias against the file system described above.

The cache occupies a larger percentage of main memory as the memory size increases, indicating that the extra memory is being utilized more by the file cache than by the VM system. Consider the Sun3 clients in the first study. Doubling the physical memory on a Sun3 client quadruples the average cache size on the client; it increases from 17% to 34% of the physical memory. In the second study the average cache size as a percentage of memory size increases with memory size.

Cache Size (Megabytes) (Fall ’89)							
Host	Mem	Average		Std Dev		Maximum	
Allspice*	128	67.8	52%	22.14	17%	78.13	61%
Assault**	24	7.5	31%	4.55	19%	16.50	69%
Mint	16	9.0	56%	1.23	8%	11.80	74%
Oregano	16	8.6	54%	1.77	11%	12.06	75%
Sun3	8	1.4	17%	0.96	12%	4.42	55%
Sun3	12	3.2	27%	1.76	15%	7.88	66%
Sun3	16	5.5	34%	2.86	18%	12.22	76%
Sun4	12	2.1	17%	1.73	14%	6.87	57%
Sun4	24	6.0	25%	3.70	15%	13.43	56%
DS3100	24	6.3	26%	2.27	9%	10.36	43%

Table 17. Cache sizes as a function of main memory size and processor type, averaged over the 6-month 1989 study period. The average, standard deviation, and maximum values of the observed cache sizes are given. The sizes are megabytes and percentage of main memory size. The file servers are listed individually. The rest of the clients are averaged together based on CPU type and memory size.
* Allspice’s cache was limited to at most 78.13 Mbytes.
** Assault’s cache was limited to 8.7 Meg during most of the study.

Cache Size (Megabytes) (March-April ’91)							
Host	Mem	Average		Std Dev		Maximum	
Allspice	128	93.8	73%	15.1	12%	106.4	83%
Anise	32	17.9	56%	2.5	8%	22.2	70%
Assault	24	9.5	40%	2.3	10%	14.0	58%
Clients	8	0.5	7%	0.4	5%	1.6	20%
	16	1.7	11%	1.5	10%	7.7	48%
	24	4.8	20%	3.0	12%	12.7	53%
	28	6.8	24%	3.6	13%	16.3	58%
	32	10.5	33%	3.7	12%	15.2	48%

Table 18. Cache sizes as a function of main memory size and processor type, averaged over the March-April ’91 study period. The average, standard deviation, and maximum values of the observed cache sizes are given. The sizes are megabytes and percentage of main memory size. The file servers are listed individually. The rest of the clients are averaged together based on memory size. There were no limits on cache sizes during this study.

The variability of the client caches is indicated by the standard deviation and the maximum observed values. The variability tends to increase as the memory gets larger, indicating that the cache is trading more memory with the VM system. The DECstations have lower variability in the initial study because their cache was limited to about 8.7 Meg. This limitation was a software limit that was removed before the second study was made, and the variability was slightly higher during the second study period.

The results from the servers show that Mint and Oregano could only devote on average a little over half their memory to their file cache, yet their maximum cache size was as much as 3/4 of their memory. Similarly, during the second study, Allspice’s cache averaged 73% of its memory, while its maximum was 83%.† The server caches ramp up to a maximum shortly after booting, and then gradually decline in size as the kernel builds up state information about how its files are being used. There are no fixed sized tables in the implementation, and the file system data structures have not been tuned to reduce their space. The servers also grow the number of RPC server threads they keep, and each thread has a significant amount of preallocated buffer space as well as a kernel stack. Thus the file servers need large memories, both to increase their cache sizes and to accommodate the data structures they maintain in order to manage their cache and those of their clients.

6. Related Work

The Sprite caching system was initially studied by Nelson for his thesis work [Nelson88a]. He compared 9 different client writing policies in combination with 4 different

† During the initial study Allspice’s cache size was limited due to a poor interaction with the memory mapping hardware on the Sun4. The file cache used up hardware page map entries, and if the cache got too large it would cause extreme contention for the few remaining hardware map entries. The problem was fixed by the time the second study was made by allowing the MMU manager to steal hardware map entries associated with cache blocks.

server writing policies on a set of benchmarks. He found that using a delay policy on both clients and servers minimized I/O traffic and provided the best client response time. In contrast, a "write through on close" policy, which is used in NFS, increases network and disk traffic and causes clients to wait at close time.

AFS uses a hybrid strategy of caching temporary files (those under "/tmp") with a delay policy, but writing through other files at close. Nelson found this to be almost as good as the Sprite policy in terms of network bandwidth reduction, although significant delays can still occur when non-temporary data is written through to the server. AFS caches remote files on the local disk, in contrast to Sprite's use of main memory. Howard found a read miss rate of about 20% on the AFS client caches [Howard88], which is better than the miss rates found in this study. The better miss rate in AFS is because the disk-based caches of AFS are larger than the main memory caches in Sprite (AFS clients usually have 20 or 40 Meg caches). The response time and server CPU utilization of NFS, AFS, and Sprite were compared in [Nelson88b]. Sprite provided the best response time (25% faster than AFS and 35% faster than NFS), while AFS had better (i.e. lower) server utilization (16% server CPU utilization under load for AFS vs. 38% for Sprite and 80% for NFS). Another comparison between Sprite and NFS can be found in [Srinivasan89], in which the addition of the Sprite delayed-write policy and consistency mechanism to an NFS system improved performance significantly.

Earlier studies of I/O traffic include Ousterhout's measurements of timesharing VAXes running 4.2 BSD UNIX [Ousterhout85], which reported per-user I/O rates of 300-600 bytes/sec when averaged over 10 minute intervals, and rates of 1400 to 1800 bytes/sec when averaged over 10 second intervals. These rates do not include paging traffic, and they are for active users only. Rates are higher over shorter intervals because there are fewer active users in a shorter interval. The rates obtained for Sprite clients, about 1500 bytes/sec for combined read and write traffic in the mid-afternoon, are averaged over 180 minutes and include periods of inactivity. Ousterhout and his students have recently taken trace data from the Sprite network in order to make similar measurements as those of his original study, and these results should be published this year [Baker91].

Kent also took trace data from a timesharing UNIX system and used the data to drive a simulation of a network file cache [Kent87]. Kent found per-user I/O rates of about 2 kbytes/sec, which is slightly higher than the rates found by Ousterhout *et al* because Kent's measurements included paging requests.

Floyd studied file and directory access patterns in the UNIX environment [Floyd86]. His focus was not on I/O rates, but on the rate that files were opened, file lifetimes, and the difference in use of different classes of files. Floyd found, for example, that most temporary files live less than one minute. The faster workstations used today should consume temporary files at a faster rate, suggesting that the 30 second delay period used in Sprite is appropriate.

One common difference between the trace driven studies described above and the measurements presented here is that the trace driven studies predicted lower cache miss ratios. Kent simulated combined read-write miss ratios that varied from 32% down to 11% as the cache size increased from 256 kbytes to 8 mbytes (see Table 5-1 in [Kent87]). Ousterhout's simulations showed a combined read-write miss ratio that varied from 49% to 25% as the cache size increased from 390 kbytes to 16 mbytes. The combined miss ratios measured here are about 40%. Part of the difference is probably due to the variable sized caches, and part is probably due to changing workloads. The bias against the file system in the variable sized cache mechanism means that cache miss rates may increase while page fault rates are reduced. The file system cache may not perform as well, but the system feels faster to to user. There is also a difference in workload between a timeshared VAX 11-780 and a network of personal workstations that are each 10 times as fast as a VAX. Window systems invite users to do more things at once, and the guaranteed CPU power of a personal workstation encourages large jobs such a system recompilation. As a result, the workload applied to the file system has scaled up considerably.

7. Conclusion

This paper has reviewed the Sprite caching system and reported on its performance when supporting day-to-day work in our user community. Measurements from two study periods have been presented. The first study was made in the fall of 1989, shortly after Sprite was made available to users outside the development team, while the second study was made 16 months later. There are a number of significant results. Client write traffic ratios averaged 52% and 62% in the two studies, meaning that almost half the data generated by applications was never written through to the server because it was deleted or overwritten before the 30-second aging period expired. Client read miss rates were 35% and 39% in the two studies, indicating reasonable effectiveness. There was low overhead from consistency-related actions. In less than 1% of open operations did the server have to issue cache control messages. The most interesting negative result is that the shared database used to record host load averages accounted for approximately 10% of the network write traffic and up to 60% of the network read traffic for some clients. This problem has been cured by replacing this heavily shared file with a network server process and tuning the interface to it. The result is that concurrent write sharing, which causes files to be uncacheable on clients, occurs in less than 1% of the files opened. There is very little consistency-related traffic between the servers and clients, and there is very little data traffic to uncachable data.

Acknowledgements

I would like to thank Prof. Ousterhout and the other members of the Sprite team, Mary Baker, Fred Douglis, John Hartmann, Mendel Rosenblum, and Ken Shirriff, for their help and cooperation as I continued this study after my graduation. I would also like to thank Gene Spafford for this opportunity to publish this paper outside the initial workshop it was originally targeted for, and to thank all the outside reviewers for their feedback on initial versions of the paper.

References

Baker91. M. Baker, J. Hartman, M. Kupfer, K. Shirriff and J. Ousterhout, ‘‘Measurements of a Distributed File System’’, *Submitted for publication*, Feb. 1991.

Douglis90. F. Douglis, ‘‘Transparent Process Migration for Personal Workstations’’, PhD Thesis, Sep. 1990. University of California, Berkeley.

Floyd86. R. Floyd, ‘‘Short-Term File Reference Patterns in a UNIX Environment’’, Technical Report Tech. Rep. 177, University of Rochester, Mar. 1986.

Hagmann87. R. Hagmann, ‘‘Reimplementing the Cedar File System Using Logging and Group Commit’’, *Proc. of the 11th Symp. on Operating System Prin.*, Nov. 1987, 155-162.

Howard88. J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham and M. J. West, ‘‘Scale and Performance in a Distributed File System’’, *Trans. Computer Systems 6*, 1 (Feb. 1988), 51-81.

Kent87. C. Kent, ‘‘Cache Coherence in Distributed Systems’’, PhD Thesis, Dec. 1987. Purdue University.

Nelson88a. M. N. Nelson, ‘‘Physical Memory Management in a Network Operating System’’, PhD Thesis, Nov. 1988. University of California, Berkeley.

Nelson88b. M. Nelson, B. Welch and J. Ousterhout, ‘‘Caching in the Sprite Network File System’’, *Trans. Computer Systems 6*, 1 (Feb. 1988), 134-154.

Ousterhout85. J. Ousterhout, H. D. Costa, D. Harrison, J. Kunze, M. Kupfer and J. Thompson, ‘‘A Trace-Driven Analysis of the UNIX 4.2 BSD File System’’, *Proc. 10th Symp. on Operating System Prin., Operating Systems Review 19*, 5 (December 1985), 15-24.

Ousterhout88. J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson and B. Welch, ‘‘The Sprite Network Operating System’’, *IEEE Computer 21*, 2 (Feb. 1988), 23-36.

Ousterhout89. J. Ousterhout and F. Douglis, ‘‘Beating the I/O bottleneck: A Case for Log-Structured File Systems’’, *Operating Systems Review 23*, 1 (Jan. 1989), 11-28.

Rosenblum91. M. Rosenblum and J. Ousterhout, ‘‘The Design and Implementation of a Log-Structured File System’’, *Submitted for publication*, Feb. 1991.

Srinivasan89. V. Srinivasan and J. Mogul, ‘‘Spritely NFS: Experiments with Cache-Consistency Protocols’’, *Proc. 12th Symp. on Operating System Prin., Operating Systems Review 23*, 5 (December 1989), 45-57.

Thompson87. J. Thompson, ‘‘Efficient Analysis of Caching Systems’’, PhD Thesis, 1987. University of California, Berkeley.

Welch86. B. B. Welch and J. K. Ousterhout, ‘‘Prefix Tables: A Simple Mechanism for Locating Files in a Distributed Filesystem’’, *Proc. of the 6th ICDCS*, May 1986, 184-189.

Welch88. B. B. Welch and J. K. Ousterhout, ‘‘Pseudo-Devices: User-Level Extensions to the Sprite File System’’, *Proc. of the 1988 Summer USENIX Conf.*, June 1988, 184-189.

Welch89. B. Welch, M. Baker, F. Douglis, J. Hartmann, M. Rosenblum and J. Ousterhout, ‘‘Sprite Position Statement: Use Distributed State for Failure Recovery’’, *Proc. of the Second Workshop on Workstation Operating Systems (WWOS-II)*, Sep. 1989, 130-133.

Welch90. B. B. Welch, ‘‘Naming, State Management, and User-Level Extensions in the Sprite Distributed File System’’, PhD Thesis, 1990. University of California, Berkeley.