# 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 months 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, about half 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. †

February 1, 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 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

† This work was done 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.

servers.

The main results presented in this paper are summarized in Table 1. Less than 1% of the files opened triggered server consistency actions, which indicates that the consistency scheme poses 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. About half 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 January 1991. At the time of the 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. Sprite is 4.3 BSD compatible, and was used for all the day-to-day computing needs of twenty or more grad students, a few professors, and a couple of staff members. There were about a dozen more other users that used the system occasionally. 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). The data was recorded for a 6 month period. However, some new statistics were introduced late in the study period to focus on particular metrics, so there are occasional references to shorter study periods in the paper, including the study made a year later to see how I/O traffic patterns had changed.

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, and Section 7 concludes the paper.

| Summary of Caching Measurements | |
|---|---|
| Files requiring consistency callbacks | < 1% opens |
| Files concurrently read shared | 36% 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. 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.

## 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 that 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. 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. This scheme is based on the assumption that concurrent write-sharing is rare, although there was one application in our network that made heavy use of a shared database file. The effects of this database will be described in Sections 3 and 4.

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 clientA if clientB opens a file and clientA 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 clientA if clientB opens a file for writing and clientA 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 use of delayed writes reflects a tradeoff between reliability and performance. Sprite has a recovery system that recovers from server and client failures [Welch89], although a power failure on a client can result in the loss of recently generated data. This is no worse than a timeshared UNIX system. Our editors and source code control programs use a system call to force files through to the server's disk. Even still, there is 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 a 36-day study are given in Table 2. The table is organized to show both hourly rates and per-server rates. The large number of opens done 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.

| File Sharing and Cache Consistency Actions | | | | | | | |
|---|---|---|---|---|---|---|---|
| Hour or Server | Num Opens | Non-File | Can't Cache | Read Sharing | Last Writer | Server Action Write-back | Server Action 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 2. 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. 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. Mint is a Sun3-180 with 16 Meg of memory. It serves the root directory, system log files, some shared database-like files, and most libraries and commands. Oregano is a Sun3-140 with 16 Meg of memory. It serves /tmp, some commands, and some source directories. Allspice is a Sun4-280 with 128 Meg of memory. It serves most user files, most source directories, and the swap directories used for VM paging. Assault is a DECstation 3100 with 24 Meg. It serves a few user directories.

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].[1]

Can't Cache

This value indicates the percentage of files opened that could have been cached on clients but were not cachable because of concurrent write sharing. The large amount of sharing measured on Mint, 15%, is explained below. 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.

[1] There are potential benefits of caching name translations on clients, but this would introduce two sources of complexity that we wished to avoid. First, data cache consistency depends on the servers seeing each open and close so they can track file usage, and this is more complex if clients cache name translations. Second, the name translation caches would also require a consistency mechanism.

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. (85.2% opens were read-only, 9.3% opens were write-only, and 5.5% 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 2. 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. The database was kept open on each client by a daemon that updated the client's entry once a minute. The database was kept open intentionally to make it uncached on clients and prevent the database from moving among client caches as different daemons made their updates. However, some other process apparently opens the database periodically (the daemon opens it once and keeps it open).

The second anomaly in Table 2 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.

Since the first study was made, the function of the load average database was reimplemented by an active server process, or pseudo-device [Welch88]. The server can make more intelligent choices for migration, plus its interface is more efficient [Douglis90]. Table 3 shows consistency related statistics after this change. There is a small rate of opens to uncachable files, but nothing like the 7.5% overall rate found in the earlier data.

| New File Sharing and Cache Consistency Actions | | | | | | | |
|---|---|---|---|---|---|---|---|
| Hour or Server | Num Opens | Non-File | Can't Cache | Read Sharing | Last Writer | Server Action Write-back | Server Action Invalidate |
| 8 | 934,566 | 47% | 0.21% | 36% | 1% | 0.75% | 0.10% |
| 11 | 165,476 | 21% | 0.61% | 47% | 5% | 0.98% | 0.28% |
| 14 | 280,043 | 22% | 0.45% | 46% | 8% | 0.81% | 0.32% |
| 17 | 346,710 | 40% | 0.30% | 35% | 7% | 0.90% | 0.17% |
| 20 | 211,317 | 26% | 0.30% | 48% | 6% | 0.96% | 0.16% |
| Allspice | 1,542,140 | 37% | 0.32% | 45% | 4% | 0.65% | 0.19% |
| Anise | 214,013 | 40% | 0.05% | 2% | 7% | 1.28% | 0.02% |
| Assault | 181,964 | 46% | 0.49% | 32% | 7% | 1.83% | 0.15% |
| Combined | 1,938,117 | 38% | 0.31% | 40% | 4% | 0.83% | 0.16% |

Table 3. Consistency data from Allspice, Assault, and Anise for one week in January 1991. Allspice is the primary server, while Assault and Anise store user files. The top-half of the table gives an hourly breakdown of all the servers combined. The bottom half gives the total breakdown for each server individually. The last row has the totals for all the servers combined.

## 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 data presented is based on a 36-day study in November and December, 1989. The tables present I/O rates in bytes per second, the miss ratio of the cache, and a breakdown of the network traffic in terms of cache misses, uncachable data, and paging data from the VM system. In particular, 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 uncachable data and VM data, as well as the total remote I/O rates.

Table 4 through Table 6 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 average I/O rates over the preceding interval (e.g. 8pm to 8am, 8am to 11am, and so on). The bottom row gives I/O rates and percentages averaged over 24 hours. The ''Cache'' column gives the read rate from the cache (not counting reads to uncachable data), and the ''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 remote read traffic. The other two primary sources of remote read traffic, uncachable files and page faults, are listed in the columns

| DS3100 Client Read Traffic (Bytes/Seconds and ratios) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 4. 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 remote read traffic. The ''Total'' column gives the total remote read traffic.

| Sun-3 (12 Meg) Client Read Traffic (Bytes/Seconds and ratios) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 5. Hourly read traffic for Sun-3 clients with 12 Meg main memory.

| Sun-3 (8 Meg) Client Read Traffic (Bytes/Seconds and ratios) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 6. Hourly read traffic for Sun-3 clients with 8 Meg main memory.

labeled ‘‘Uncached’’ and ‘‘PageIn’’. The last column gives the total remote I/O traffic. Some of the total remote 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 7. Their poor miss rate, almost 60%, stems from a single application that scanned a shared database periodically (5K to 10K every 15 seconds!). The database recorded host load averages and was used

| Abnormal Client Read Traffic (Bytes/Seconds and ratios) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 7. 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. Note that this collection of clients has the largest I/O rates to its cache and that the miss rate does not include traffic to uncachable data, only to data that could be cached.

to choose hosts for process migration. A daemon process on each host updated the database once a minute, and it kept the database open to force it to be uncachable. An X widget that displayed the number of hosts available for migration caused periodic scans of the database, 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 shared file. Table 8 shows the read traffic for a collection of DECstation and Sun-4 clients after this change was made. There is almost no read traffic to uncachable data, and VM paging traffic dominates the network.

### 4.2. Client Write Traffic

Table 9 through Table 11 give the write traffic for the DECstation and Sun3 clients. The format of the tables is similar to those with read traffic. Hourly breakdowns are given, and remote traffic is divided among cache misses, uncachable data, and writes due to page outs.

| New Client Read Traffic (Bytes/Seconds and ratios) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Hr | Cache | Misses | | | Uncached | | PageIn | | Total |
| 8 | 68 | 23% | 13 | 14% | 2 | 3.0% | 80 | 82% | 96 |
| 11 | 375 | 53% | 196 | 32% | 7 | 1.3% | 399 | 66% | 604 |
| 14 | 590 | 37% | 211 | 28% | 11 | 1.6% | 514 | 69% | 738 |
| 17 | 736 | 37% | 266 | 32% | 10 | 1.3% | 547 | 66% | 824 |
| 20 | 275 | 30% | 80 | 18% | 6 | 1.5% | 341 | 79% | 428 |
| 23 | 276 | 35% | 91 | 29% | 11 | 3.8% | 210 | 67% | 313 |
| TL | 174 | 35% | 59 | 28% | 3 | 1.9% | 149 | 70% | 213 |

Table 8. 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) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 9. Hourly write traffic for all the DECstation 3100 clients. The format of the table is the same as Table 4. 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) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 10. 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) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 11. 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.

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.

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 12 shows more recent data for the Sprite network, after the uncachable migration 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 70%, well above the 50% found in the initial study. This follow-on study, however, was only one week long and occurred during a period of relative inactivity as shown by the overall I/O rates. It was probably not long enough to match the accuracy of the earlier study with regards to long term cache effectiveness.

| New Client Write Traffic (Bytes/Seconds and ratios) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Hr | Cache | WriteBack | | | Uncached | | | PageOut | | | Total |
| 8 | 28 | 72% | 20 | 71% | 1 | 5.0% | | 5 | 19% | | 28 |
| 11 | 111 | 84% | 94 | 77% | 1 | 1.4% | | 21 | 17% | | 121 |
| 14 | 176 | 72% | 127 | 60% | 1 | 0.8% | | 72 | 34% | | 210 |
| 17 | 259 | 66% | 173 | 57% | 1 | 0.5% | | 120 | 39% | | 301 |
| 20 | 85 | 66% | 56 | 32% | 1 | 0.9% | | 109 | 63% | | 171 |
| 23 | 68 | 69% | 47 | 48% | 1 | 1.7% | | 32 | 33% | | 97 |
| TL | 56 | 70% | 39 | 56% | 1 | 1.3% | | 26 | 38% | | 69 |

Table 12. Hourly write traffic after the shared migration 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.

### 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 compare cache traffic to disk traffic:

$$File\ \ Traffic\ \ Ratio = \frac{F}{C}$$

$$Metadata\ \ Traffic\ \ Ratio = \frac{M}{C}$$

$$Total\ \ Traffic\ \ Ratio = \frac{M + F}{C}$$

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 Table 13 and Table 14. Figure 1 shows the server I/O traffic for a combination of all servers averaged over a 6-month study period. 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

| Server I/O Traffic (Bytes/Second) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 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 13. I/O traffic on the file servers over a 20-day period. 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 14. This gives the total megabytes transferred for the results given in Table 13, 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.

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 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 meta-data, is logged [Ousterhout89].

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

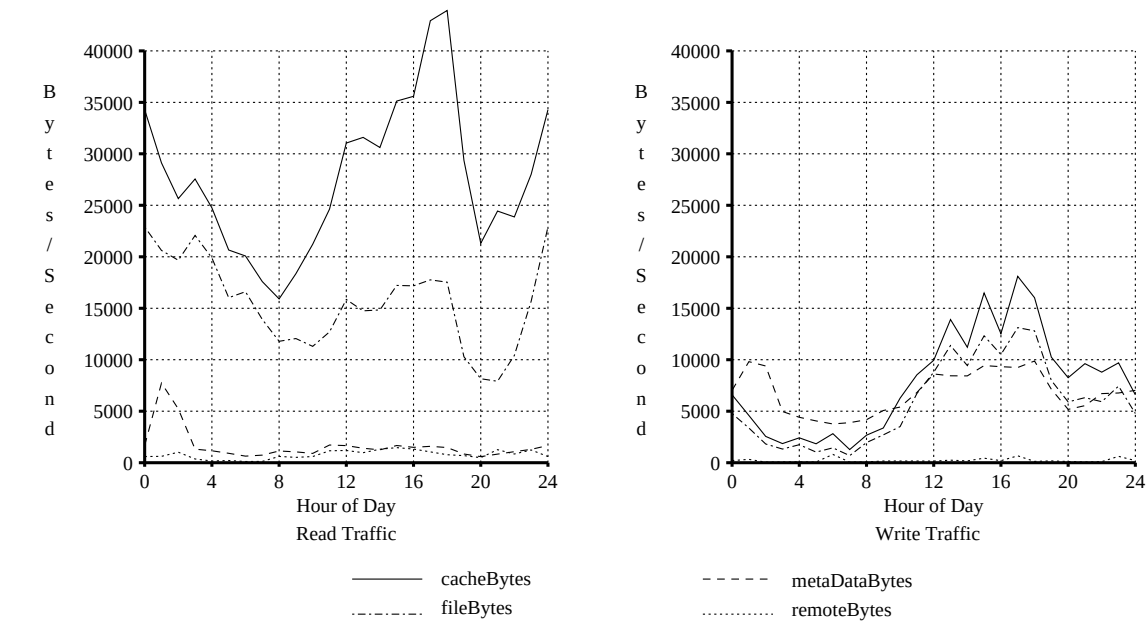


Figure 1. Server I/O traffic averaged from July 8 to December 22, 1989. The left-hand graph has read traffic and the write-hand graph has write traffic. ‘‘cacheBytes’’ are file data bytes transferred between the cache and remote clients or server-resident applications. ‘‘fileBytes’’ are file data bytes 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 ‘‘meta-DataBytes’’.

descriptors, which explains Mint’s high metadata traffic.

The 20-day study reported in the tables was made after a bug was fixed that prevented continuously updated files from being written through to disk. With this bug present, Mint’s traffic ratio was about 50% for file data, 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 this 20-day study was made after it was fixed. That Mint’s current traffic ratio changed from 50% to 94% indicates almost half the data written to it is to continuously updated files, i.e. the migration 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 migration 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 towards writes at the disk level should continue as the 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 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. 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 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. Average Cache Sizes

Table 15 gives the average and maximum cache sizes as measured over the full 6-month study period. The file servers are listed individually. The clients are grouped according to the amount of physical memory and processor type of the host, and the results are averaged. 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. This trend is most evident on the Sun3 clients, of which there is a good population size, and in which the Sprite implementation is the most mature. 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.

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 because their cache was limited to about 8.7 meg during most of the study period. (This limitation was a software limit that was eventually removed.)

The results from the servers show that Mint and Oregano can only devote on average a little over half their memory to their file cache, yet their maximum cache size is as much as 3/4 of their memory. Their caches ramp up to a maximum shortly after booting, and then

| Cache Size (Megabytes) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 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 15. Cache sizes as a function of main memory size and processor type, averaged over the 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.

gradually decline in size as the kernel builds up state information about how its files are being used. The servers also grow the number of RPC server threads they keep, and each thread has a significant amount of pre-allocated buffer space as well as a kernel stack.

In the case of Allspice, however, the limitation on its cache size is due to an artifact of the memory mapping hardware on the Sun4. The file cache uses hardware page map entries, and if the cache gets too large it can cause extreme contention for the few remaining map entries. Its cache is fixed arbitrarily at about 80 meg because of this. Even with this limitation, the cache on Allspice is 10 times the size of the cache on Mint and Oregano, and measurements presented in Section 4.3 indicate that a server needs a large cache like this for the cache to be really effective.

## 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 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 a 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 35% miss rate 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.

## 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. There are a number of significant results. Client write traffic ratios average about 50%, meaning that half the data generated by applications is never written through to the server because it is deleted or overwritten before the 30-second aging period expires. Client read miss rates are about 35%, indicating reasonable effectiveness. There is 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 (after fixing one application!) there is very little data traffic to uncachable data.

## 8. References


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

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.

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.

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.