ANALYSIS OF LONG-TERM UNIX FILE ACCESS PATTERNS FOR APPLICATION TO AUTOMATIC FILE MIGRATION STRATEGIES

Stephen Strange

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720

Abstract

A study of file access patterns can be useful in designing an efficient hierarchical storage system that employs automatic file migration. This paper proposes a specific design for such a storage system, and describes a detailed study of long-term file access patterns on a number of UNIX filesystems in use at Berkeley. File access traces are collected and used to drive a file migration system simulator. The results of the study are used to support the argument that the proposed storage system design is reasonable, and to propose file migration algorithms that might best suit the storage system. Previous studies of this topic took similar approaches. However, they were based on data collected ten or more years ago from systems designed very differently than those of today. Also, because the storage capacity of memory and disks has increased 50 to 100 times in the past ten years, there is good reason to question whether the results of previous studies are valid for today's systems. Although the distributed UNIX-based system we trace in this study is vastly different than the large time-sharing systems used in previous studies, we find that some of the same types of migration algorithms still perform well. Specifically, file replacement algorithms based on multiplying the file size by the time since last access to obtain an ordering of migration preference between files work well.

Table of Contents

1 Introduction 1
2 Design Assumptions for a Hierarchical Storage System 3
2.1 Configuration Overview 3
2.2 Inode Structure Enhancements 4
2.3 Storage System Operation 5
2.4 Migration Algorithms and Strategies 7
2.4.1 Fetch Policies 7
2.4.2 Placement Policies 8
2.4.3 Replacement Policies 8
3 The Traced Systems and Trace Collection 10
3.1 Description of the System Configurations 10
3.2 Implementation of the Trace Collection Software 11
3.3 Problems Encountered During Trace Collection 13
3.4 Drawbacks of the Trace Collection Methodology 14
4 Overview of the Statistics Analyzer and Simulator 15
4.1 Statistical Analysis 15
4.1.1 File Size Distributions 15
4.1.2 Access Patterns Over Time 16
4.1.3 File Interreference Intervals 16
4.2 Simulation of a Two-Level Storage Hierarchy 17
4.2.1 Simulator Operation Overview 18
4.2.2 Migration Algorithms 19
5 Results 19
5.1 Statistical Analysis Results 20
5.1.1 File Size Distributions 20
5.1.2 Access Patterns Over Time 22
5.1.3 File Interreference Intervals 23
5.2 Simulation Results - Comparison of Migration Algorithms 25
5.3 Comparison with Previous Published Research Results 28
6 Conclusions 28
7 Further Study 29
7.1 File Chunking 30
7.2 Disk-To-Archive Traffic Analysis 30
8 References 31
9 Further Reading 31

1 Introduction

Almost all computer systems employ a storage hierarchy such as the one depicted in Figure 1. Lower levels of the hierarchy provide far more storage capacity, while higher levels provide high bandwidth and low latencies. In most systems, data is moved between all these levels automatically by the system software (virtual memory page and swap routines) and/or hardware (cache controllers), with the exception of the mass storage level. In the past, data to and from this level had to be moved manually, that is, with the assistance of an operator, as tapes had to be hand-loaded. With the recent appearance of jukebox-type storage devices on the market, this becomes unnecessary. Optical disks or tape cartridges can be loaded automatically by robot within tens of seconds, rather than the minutes or even hours it could take to manually load tapes.

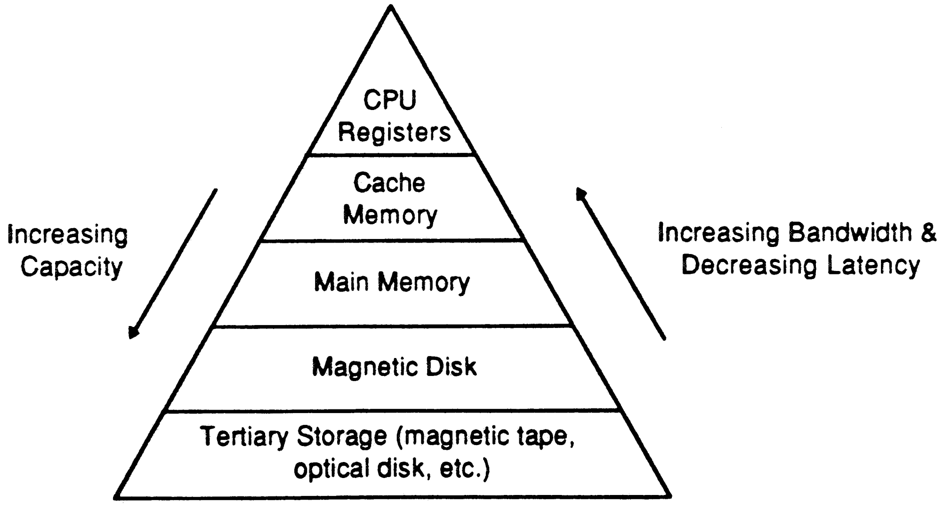

CPU
Registers
Cache
Memory
Main Memory
Magnetic Disk
Tertiary Storage (magnetic tape, optical disk, etc.)
Increasing Capacity
Increasing Bandwidth & Decreasing Latency

Figure 1: Basic Memory Hierarchy of a Computer System.

Instead of viewing tertiary storage as simply a backup, archive, or transfer medium, as it has been in the past, it can now be viewed as simply another level in the memory hierarchy.

Files moved between mass storage and disk can be compared to virtual memory pages moved between disk and physical memory. The principle of locality [Denn72] suggests that the vast majority of references to files in the storage system will not reach the bottom level of the hierarchy. Therefore, the significant latencies in jukebox devices (on the order of ten seconds to mount a new volume) may not pose a performance hazard to the system.

To take advantage of the principle of locality, algorithms for the movement of data between the levels of the storage hierarchy must be developed. Which algorithms perform best is dependent on file access patterns. Some studies of this problem have been made and published, including [Smit81b, Lawr82], but most involved data collected more than ten years ago. A primary motivation for this study was to determine whether typical file reference patterns have changed over the years. There is good reason to believe that the tremendous increase in storage capacity and processor speed over the past ten years may have changed the way in which users use the system, which in turn may have changed file access patterns. In addition, the system we trace is a distributed system of workstations and servers networked together, very much unlike the large, centralized, multi-user machines studied previously.

In this study, we analyze file reference traces collected from a number of UNIX file systems used by Computer Science Division students, faculty, and staff. The analysis is geared toward developing and comparing file migration algorithms (via trace-driven simulation) for a general hierarchical storage system model. We present statistics on file access counts and distribution of file sizes (weighted by read and write access and unweighted). This shows how the different file systems are used and how a storage system design can take advantage of particular patterns.

The following section describes the basic design assumptions for the proposed hierarchical storage system. Section 3 describes our trace collection methodology, the tools available for trace collection, and the peculiarities of the file systems traced. We also discuss the advantages and drawbacks of this methodology, and their expected effects on the results.

Section 4 describes the statistics analyzer and the simulator used to generate the results of the study. The results are discussed in Section 5. Graphs of the results are shown in Appendix A.

2 Design Assumptions for a Hierarchical Storage System

To have useful traces to apply to the analysis of a hierarchical storage system, a basic, somewhat detailed design of the system must be proposed. System design assumptions affect what information is needed in the traces, how the traces will be collected, and at what intervals the system should be traced. The system design also dictates how the traces should be analyzed, and, of course, how the storage system should be simulated in software. The following subsections describe the system design in detail, including the overall configuration, UNIX file system enhancements, day-to-day operations, and migration strategies.

2.1 Configuration Overview

The storage system consists of a collection of magnetic disks and one or more magneto-optical jukeboxes or tape robots. For brevity, the magnetic disks will henceforth be referred to as simply *disk*, and the optical disk or tape device(s) will be referred to as *archive*. The disk serves as a write-back cache for the archive – files are automatically migrated between disk and archive by the system. The primary goal is to provide users with the high storage capacity of the archive with the performance characteristics of the disk. UNIX directory files are never migrated – the entire inode structure of each disk is kept on the disk, only the "leaves" of the directory tree are migrated.

We assume that the archive has at least twice the storage capacity of the disks, and could have over ten times the capacity. Although supercomputer installations often have much higher archive-to-disk ratios[1], an initial analysis of our results suggests that it is unreasonable to expect overall performance to approximate pure magnetic disk with these ratios in our UNIX environment. The archive-to-disk ratio is based on a tradeoff between performance and hardware cost – a high ratio results in lower cost per megabyte of storage, while a smaller ratio generally results in better performance in terms of shorter latencies and greater bandwidth. One goal of this study is to investigate how this ratio affects performance.

To make further discussion clear, we'll define the following operations that can be performed on each file:

- MigIn – The copying of a file from archive to disk. The copy on archive remains intact.
- WriteOut – The copying of a file from disk to archive. The disk copy remains intact, but the *dirty* bit is cleared (the dirty bit is described in the following section).
- MigOut – The moving of a file from disk to archive. The disk copy of the file is removed, but the inode structure for the file remains on disk. The *resident* bit is cleared (see the following section).
- Delete – The removal of a file from disk. The inode structure remains intact, and the resident bit is cleared, as in migOut.
- Scratch – The removal of a file and its inode structure from disk, and the removal of the archive copy of the file, if one exists. (Note: We consider scratch to be a system-level definition that does not necessarily correspond to the action taken when a user uses the rm command. For example, a "trashcan" feature to protect the user from himself would be implemented between the implementation levels of rm and scratch.)

In addition to these operations, we define a *file hit* and a *file miss* to be the results of file requests on resident and non-resident files, respectively.

2.2 Inode Structure Enhancements

To maintain the additional information needed to keep track of the status of files in the storage system, a number of elements are added to the standard UNIX inode structure. These elements include the following:

[1] One Cray-based installation we know of consists of 45 GB of on-line disk storage, 1.2 TB near-line, and well over 10 TB off-line.

- A "resident" bit – This flag indicates whether the file is currently resident on disk, or has been removed from disk to make room for other files.

- A "dirty" bit – This bit is set if the copy of the file on disk is "dirty," and clear if the file is "clean." A file which has been updated (modified or overwritten) or created on disk more recently than the last migOut or writeOut of the file is considered dirty. (All files are dirty when first created, since no copy yet resides on the archive). Conversely, a file that has not been written since its last migOut is clean. A clean disk file has an identical copy on the archive, and therefore can be removed from disk without losing data.

- An archive address field – This field contains an address that points to the location of the file on the archive. For example, it could consist of a platter ID and an inode number for the file if the storage device were an optical jukebox with each platter formatted as a separate UNIX file system. The archive address field could also be an index into a database stored on disk, with each database entry containing the file's location on the archive. This latter arrangement would consume more storage space, but would be potentially more maintainable, as independent utilities could access and update the database to perform media replacement or archive defragmentation. This field is set to zero when a file is first created, and is later updated when a migOut or writeOut first occurs on the file.

- MigVal – This value represents the file's preference for migration. It is set and used by the nightly migration run described in Section 2.3.

2.3 Storage System Operation

The operation of the hierarchical storage system is based on a daily cycle. It is assumed that the bulk of interactive system activity occurs during normal working hours, and that there are few if any interactive users late at night. This gives the system a chance to "clean up" each night in preparation for the following day's activity. The primary goal of this nightly activity is to clear enough disk space that users will not run out of high-speed storage. "Enough" is defined by a per-file system parameter, a *migration watermark*, that is set based on previous experience.

If enough disk space is not provided, performance could be severely degraded as the system scrambles to migOut files during periods of high interactive activity. Performing migOuts during active periods should be avoided except as a last resort to avoid data loss. Clearing too much space on disk might cause performance problems, as reducing the number of resident files is likely to lead to a higher file miss rate. However, this latter problem can be avoided by performing writeOuts on files that are candidates for removal, leaving the "non-dirty" file on disk until the space is actually needed. At that time, the disk copy of the file can be deleted quickly. Thus the migration watermark specifies the maximum amount of space on disk that can be occupied by *dirty* files. Given this approach, it may appear that the best strategy would be to perform writeOuts on every file on disk each night. Problems with this approach include excessive fragmentation of write-in-place devices such as optical jukeboxes, and the amount of time required to copy every file to archive. As we shall see later, many files, particularly small ones, need never be written to archive to maintain good performance.

A major motivation for this research is to determine what a "reasonable" migration watermark is for a given hardware configuration and usage patterns. What is defined as "reasonable" will depend on the level of performance required, and the ratio of disk to archive space (which is directly related to cost). Also, the approach of performing writeOuts without deleting files from disk will be used in simulation to determine what hit rate can be achieved by using various migration algorithms.

The nightly migration run involves the following steps performed for each file system:

- Generate a list of files on disk in order of their preference for migration. This preference is expressed as an integer value in migVal, an element of the inode structure for each file. This value is determined by a file replacement algorithm. Higher values represent a higher preference for migration.

- If the amount of data stored on disk exceeds the migration watermark, perform writeOuts on files starting at the top of the migration preference list. Continue down the list until the migration watermark goal is reached. The migration preference list is maintained in a disk file for use during the active daytime period, so that files can be deleted from disk as needed in the same order they were written out to archive.

When a file is accessed (typically via a file open call), the storage system checks to see if the file is currently residing on disk. If so, the request is handled in the normal fashion. If not, the file is located on the archive, and a migIn is performed. If a migIn would cause the disk to become too full, files are deleted from disk. A *maxResident* watermark determines the maximum amount of space allowed to be used by the disk before files are deleted to make space. This watermark would typically be set close to the capacity of the disk, but low enough that the daemon handling deletions can keep up with other processes' writing of new data. Files are deleted in the migration preference order established during the previous night's migration run, except that any files with their dirty bits set are skipped.

2.4 Migration Algorithms and Strategies

In general, a *migration policy* consists of three separate but related policies: a *fetch policy*, which determines when a migIn should be performed, a *placement policy*, which determines where on disk or archive a file should be placed, and a *replacement policy*, which determines how and when disk files are chosen to be removed to keep sufficient free working space on disk. This project concentrates on the investigation of replacement algorithms, but because all three should be considered in a storage system design, they are all discussed briefly in the following sections.

2.4.1 Fetch Policies

The most straightforward fetch policy is simply fetching on demand. This policy dictates that when a non-resident file is requested, a migIn is immediately performed, blocking the requesting process until the file (or at least the first block of the file) has been copied to disk. Under certain circumstances, it might be best to allow the requesting process to access the file directly from archive. For example, performing a migIn on a very large file might force migOuts on many other files, causing long queueing delays on the archive. For purposes of this study, however, we will assume all files are demand-fetched.

In addition to demand fetching, it might be beneficial to *prefetch* files, before they are actually referenced. Prefetching might be based on user id or directory placement. For example, if one file in a directory is referenced, we might assume it is likely that other files in the same directory with the same user id will be referenced in the near future, and that prefetching them might reduce average latencies.

2.4.2 Placement Policies

To obtain reasonable performance from an archival storage device, data should be placed according to a policy based on the locality inherent in the file access patterns. Jukebox-type devices usually consist of tens or hundreds of physical volumes, each with a fixed and limited storage capacity. The goal in placement is to maximize repeated references to the same physical volume, or a small set of physical volumes, in a given time period. This not only minimizes access latencies, but also reduces device busy time during which subsequent requests must wait. Data placement policies can also be influenced by other concerns, such as grouping data of a particular type or owner together for the purposes of archiving or data exchange. We do not study the placement problem, but assume a reasonable approach is taken.

2.4.3 Replacement Policies

The replacement policy specifies which files are to be copied to archive and removed from disk. These removals make room for newly-created files or the migIn of non-resident files referenced in the near future. As in cache line or virtual memory page replacement policies, the goal here is to remove the data that is least likely to be accessed in the near future. One simple, effective strategy used in caches and VM is least-recently used (LRU). In file migration, this translates into using the time-since-last-reference of the file, which can be computed from the file access time stored in the UNIX inode structure.

Because the data units here are files of various sizes (unlike in caches or VM where lines and pages are fixed-size), the size of the file could have some effect on replacement decisions. To see this, consider a case in which one large file takes as much space as 100 small files. If we assume all 101 files are equally likely to be accessed in the near future, removing the large file instead of the many small files will result in a lower file miss ratio in future accesses. This in turn means better performance, as the bottleneck in jukebox-type devices is often access latency per file rather than bandwidth. So clearly file size should be considered, but it is not clear how it should fit in with an expected-time-to-next-reference strategy to maximize performance. The strategies simulated here will be discussed in a later section.

It should be noted here that a decision was made to migrate files rather than fixed-size blocks between archive and disk. Some of the motivations for this decision are as follows:

- The inode structure in UNIX already has fields for file size and access time. If a block-oriented system were implemented, new data structures would have to be developed on a per-block basis, in addition to the per-file structures already there. This fact also makes trace-driven simulation much easier, as traces can be collected from a running UNIX system by simply reading inode fields.

- Studies of short-term UNIX file access patterns show that there is a very high degree of locality of access within the blocks of a file. Over 70% of file accesses (reads and writes combined) result in the entire file being read or written, and over 20% of the remaining accesses are block-sequential after an initial reposition operation [Oust85]. It seems intuitively wrong to throw this locality advantage away by treating all blocks independently.

- Migrating whole files means that the archive device can be set up as simply another UNIX file system which could be mounted on other systems. Additional text header information, such as file owner, dates, etc., could be written to each archived file, making error recovery easier. Block-based migration would lead to an archive on which some blocks of a given file will be dirty, some clean, and some non-existent (if they hadn't been written out from disk yet). Archive media would not easily be transferrable between systems, and severe error recovery would prove very difficult.

There are some advantages to a block-oriented approach, such as handling very large files like databases, where seeks and in-place writes are common. This problem could be solved by allowing large files to be broken up into chunks, and treating each chunk as a unit that can be migrated. This strategy maintains a high degree of transparency to applications, while improving the performance of the underlying storage system. Another solution is to change the way the application program stores data to reduce file size. For example, individual database records could be stored as independent files, rather than as a single, large database file. This strategy has a beneficial effect similar to file chunking, except that data is more finely split up according to actual access patterns, rather than fixed-sized chunks. This leads to a potentially higher hit rate, because less unnecessary data is migrated in along with requested data. We choose not to consider the large-file problem, particularly because database-like accesses are rarely seen on the systems traced in this study.

3 The Traced Systems and Trace Collection

The method used for trace collection and the format of the trace records are based on the configuration of the systems traced, the resources available, and a perception of what data would yield the most interesting results. The following sections describe the configuration of the systems, the design of the trace collection software, the problems encountered during trace collection, and the advantages and drawbacks of the approach taken.

3.1 Description of the System Configurations

The Sprite and Ginger file servers are actively used by about 50 faculty members, staff members, and graduate students in the CS division of UC Berkeley. Most users work on their own workstations, which are networked via ethernet to the file servers. Activity on the system includes mail, news, editing, small and large compilation jobs (including operating system kernel development), CAD, trace collection activities, nightly backups, and CPU-intensive batch-like jobs, such as simulations.

The Sprite configuration consists of two Sun 4 fileservers and one DECstation 5000 fileserver serving 35 diskless workstations (6 DECstation 3100s, 12 DECstation 5000s, 7 Sun 3s, and 10 Sun Sparcstations) networked with a 10 Mbit/s ethernet. All the workstations and servers form a distributed system running the *Sprite* experimental operating system. The three fileservers provide access to about 14 file systems, including "user" disks (containing mainly users' home directories and subdirectories), disks dedicated to operating system kernel sources and builds, disks containing system command executables and windowing system binaries, and experimental RAID file systems. We chose to analyze six of the Sprite file systems. Some of the file systems that were not chosen were eliminated because of extremely low activity levels, and others were eliminated simply to reduce the number of time-consuming simulations required.

The Ginger configuration is similar to Sprite's, however, it is running a commercial operating system – SunOS. It too consists of a Sun fileserver, along with about 25 Sun workstations (Sun 3s and Sparcstations) connected via NFS. The Ginger fileserver has considerably less total storage space than Sprite's servers. We chose to trace only one of Ginger's file systems, the primary "user" disk. We felt it was important to include at least one file system from a server outside the Sprite environment to serve as an experimental "control." If the results from this file system are similar, we can be more sure that any peculiarities of the Sprite environment did not interfere with the data gathering process (see Section 3.3).

3.2 Implementation of the Trace Collection Software

The primary considerations in designing a trace collection scheme for investigating long-term file reference patterns were:

- Simple Implementation – Trace collection needed to be implemented as a set of utility routines independent from the file system and operating system code. Although Sprite file system source code is available here at Berkeley, the logistics of implementing and testing trace collection code in the file system itself, while file system development on Sprite went on in parallel, were prohibitive. On Ginger, the option did not even exist, as it runs a proprietary commercial file system (SunOS). In addition, time constraints made simple implementation essential.

- Compact Trace Files – The trace files generated needed to be small enough that 3 months worth of traces on over ten file systems[2] would not exceed the disk space available (about 200 Mbytes).

- Minimal Resource Consumption – The trace collector could not be allowed to interfere with normal system operation, or degrade the performance of the system in a perceptible way.

- Accurate Tracing – Whatever scheme was chosen, it was essential that the data collected in the traces truly represented the activity on the system.

[2] Although only six file systems were eventually chosen for analysis, nearly all the file systems available were traced.

The trace collection scheme chosen involves running a modified version of the UNIX find command, called sfind, once on each file system every night at midnight, when system usage is very light. Sfind is directed to find all files (except directory files) that have been accessed during the previous day. Each line generated by sfind corresponds to a single file, and consists of the following ascii, space-separated fields:

- `<inode_num>` – Disk inode number for the file.
- `<Kbytes>` – Size of the file in kilobytes.
- `<numlinks>` – Number of links to this file.
- `<userid>` – User ID of file owner.
- `<mtime>` – Time this file was last modified, in integer format (seconds since midnight, 1/1/70).
- `<atime>` – Time this file was last accessed, in integer format (seconds since midnight, 1/1/70).
- `<pathname>` – Full UNIX pathname for the file.

The output of sfind is directed to a file named with the current date, and placed in a directory named for the file system from which the trace was taken. Each file is also given a header, containing the file system name, the trace collection start time, and the trace time window size[3], and a footer, containing the trace collection completion time. Sfind is run on each file system sequentially via a cshell script, which automatically names the trace files, places them in the proper directory, and appends status messages to a log file. The script is activated via the crontab utility built into Sprite and SunOS (on Ginger) each night at midnight. Sfind is executed on the fileserver that serves the file systems being traced to minimize network traffic.

In addition to the traces taken every night throughout the trace period, a full trace (all files, regardless of last access time) of each file system is taken at the beginning of the trace period. Having this data eliminates the cold-start problem in simulating the system, thereby

[3] All files accessed within a specified time window are included in a file trace. This window ends at the time the file trace is taken, and begins a specified number of hours before that time. This number is the trace time window size stored in each file trace header. The window is typically 24 hours, but is shortened to 16 hours because of a problem with the Sprite file systems discussed in section 3.3.

simplifying simulation and improving the accuracy of the results. The full trace also provides a means for obtaining file count per file system and the distribution of file sizes on disk.

This scheme satisfied the four considerations listed above. Sfind was a straightforward enhancement of the original UNIX find, and its execution via a simple shell script and crontab took less than a week to set up and debug. Trace file sizes vary depending on file system activity and on the number of files in the file system, but average about 20 blocks, posing no storage problem. Because the tracing run is executed late at night when there are few if any users, the impact on performance is negligible. A potential for accidental inclusion or exclusion of files in a trace does exist because a given file system is not always traced at exactly the same time each night. However, these time differences are never more than a few minutes, and because of the light usage of the system at night, the effects of this problem are negligible.

3.3 Problems Encountered During Trace Collection

A number of problems occurred during the testing of the trace collection software and after the tracing period had begun. First, on Sprite, the UNIX tar utility is used for both incremental and full backups. Tar updates the access time stored in the inode of each file, which causes trace collection to include files that shouldn't be included. If a full backup is done in the 24 hours preceding trace collection, every file ends up in the trace! The problem is further complicated by the fact that although full backups are made weekly, they are not always started at the same time every week. The solution is to simply exclude traces taken on the day of the full backup in analysis and simulation. Incremental backups are predictable – they are started every morning at 2:00 am, and seldom finish later than 8:00 am. The solution to the incremental backup problem is to direct sfind to collect files accessed in the last 16 hours, rather than a full 24 hours. Since tracing is started at midnight, this solution works as long as nightly backups finish by 8:00 am. Because there are very few file accesses made between midnight and 8:00 am, this does not pose a major threat to the accuracy of the traces.

The second problem encountered involves unexplained errors in the access and modification times of a significant number of files on the /home/ginger/users ("Ginger Users") file system on Ginger. A few of the files in this file system had access and modification times that represented dates that are many years in the future. This problem caused these files to appear in every trace taken, when in fact the files had not been accessed in a quite a long time. Further investigation revealed that all these files resided in directory trees owned by long-inactive users. The problem was solved by simply eliminating the faulty files from the traces. This solution is not expected to affect the results, because the faulty files were inactive during the trace period. The results clearly would have been tainted if the error had not been found.

3.4 Drawbacks of the Trace Collection Methodology

There are clearly some drawbacks to the approach taken to trace collection described above. First, there is no way to know that a file has been deleted from a file system. Deleted files will simply never show up in traces taken any time after the deletion. The effects this has on the simulation results will be discussed in Section 5.

Second, the traces only indicate the most recent time a given file was accessed. They do not indicate how many times a file was accessed on that day, or exactly when the accesses occurred, with the exception of the last access. Therefore, devising a simulation method that accounts for queueing delays in the storage devices and the CPU would be difficult. Queueing delays in the archive storage device could have a significant affect on archive access latencies, because of the amount of time it takes to automatically load and unload media, e.g. tape or optical disk.

4 Overview of the Statistics Analyzer and Simulator

Two programs were developed to analyze the traces collected in the study, a file statistics analyzer and a file migration simulator. In the following subsections, the development and use of these programs will be discussed in some detail.

4.1 Statistical Analysis

The statistical analysis program accepts a group of option parameters that specify the name of an input trace file and the names of various graph data output files. If no input file is specified, stdin is assumed, allowing the user to pipe trace files in via UNIX cat. The input to the program is a sequence of trace files, usually from a single file system. Each file corresponds to a single day during the trace period. The headers and footers in each file serve as start and end markers for each day. A single statistics collection run reads in all the traces for a given file system in the sequence they were collected, then generates data for various types of graphs.

Three classes of graphs can be generated – file size distributions, access patterns over time, and file interreference patterns. Each graph class, and the methods used to generate the data for that class, are discussed in the following sections.

4.1.1 File Size Distributions

Two different types of file size distributions can be generated: *static* – the distribution of file sizes on a file system at a given time, and *access-weighted* – the distribution of the sizes of file accesses[4] over a period of time. A static distribution is generated from the full trace of the file system taken at the beginning of the trace period, while the access-weighted distribution is generated from the daily traces. Note that due to the trace collection method, a file is considered "accessed" only once per day, regardless of the number of accesses that took place that day.

[4] The size of a file access is simply the size of the file being accessed.

The access-weighted distribution actually consists of two distributions, because reads are distinguished from writes. A file access on a given day is considered a "write" if the file was modified at any time during that day, and is considered a "read" if it was not. The analyzer distinguishes between the two by subtracting the trace time window size from the trace collection start time, then comparing the result with the last-modified time of the file in question.

File size distribution statistics are collected by continuously updating a set of file counters as trace records are read in. Each pair of counters (one for reads, one for writes) represents a range of file sizes. For example, the first counter counts all files of size one through ten kilobytes, the second counts files of size eleven through twenty, etc. Data for a cumulative distribution graph is then generated from the counters.

4.1.2 Access Patterns Over Time

A simple indication of file access patterns is the volume of file activity over a number of weeks. To show these patterns, the statistics analyzer produces two graphs – the number of different files accessed vs. days[5], and the amount of disk space associated with those files vs. days. As with the file size distributions, reads and writes (as defined in the previous section) are considered separately. The analyzer collects this data in the same way as with file distributions – file counters and Kbyte counters for both reads and writes are kept for each day, and graph data is generated from these counters.

4.1.3 File Interreference Intervals

An interreference interval is defined as the number of days between two consecutive appearances of a given file in a sequence of traces. The distribution of interreference intervals is a useful measure of how often active files are referenced. For example, if the

[5] Note that this is *not* the total number of file accesses per day. The number of *different* files accessed is more useful in migration studies, because once a file is accessed once, it will be migrated to disk, and will most likely remain on disk for the remainder of the day. Therefore, further accesses on the same day will not cause migrations.

distribution is heavily skewed toward small interreference intervals, there is a high degree of temporal locality present in the access patterns. Conversely, a distribution in which the number of long intervals is significant indicates a lesser degree of temporal locality. Clearly, a high degree of locality is desirable for purposes of file migration.

The analyzer generates separate interreference interval distributions for different file size classes. File size classes are based on the base-two logarithm of the file size in kilobytes – a file of size x Kbytes is in class $\lceil \log_2(x) \rceil$. The input to the analyzer is the full file system trace followed by the daily traces. The full trace serves to eliminate the cold-start problem, because the last-access times stored in the full trace define the starting point for each file's reference history. As traces are read in, a large hash table of file entries is built. When a file is first encountered in the trace stream, a file entry containing the sfind fields is created in the table. On subsequent appearances of that file, a small "reference" record, containing the last-access and last-modification fields, is added to a linked list. The head of this list is a field in the original file entry. Once all traces have been processed, the hash table is scanned to count the number of intervals for each interval length and for each file size class. These counters are then processed to generate interval distribution curves for each file size class. These curves can then be displayed as a single contour graph (see Appendix A page 13 for an example).

4.2 Simulation of a Two-Level Storage Hierarchy

The simulator is designed to simulate the storage system defined in Section 2. The basic strategy of the simulator is to treat the file traces as streams of accesses to the storage system. The output of this simulation is a plot of file miss ratios versus the simulated size of the magnetic disk. The goal is to determine which migration algorithms work better than others, and to visualize how the miss rate is affected by changes in magnetic disk size.

4.2.1 Simulator Operation Overview

A simulation run begins by reading in the full file system trace. From each trace record, a file entry is created and placed into a hash table. The variables *Kresident*, to keep track of total occupied disk space, and *Kdirty*, to keep track of total disk space occupied by dirty files, are updated as entries are created. All files are initially marked *dirty* and *resident*. Next, a nightly migration run is simulated. A migration run consist of two steps, as described in Section 2.3. First, a file migration preference list is generated by scanning the entire hash table, applying the specified migration algorithm to each file entry, and placing the entry appropriately in the preference list. Second, if the sum of the size fields in all the file entries exceeds the size of the magnetic disk being simulated (henceforth *maxResident*), simulated migOuts are performed on files starting at the top of the preference list until the sum falls below *maxResident*. Now, to clear enough disk space for the next day's activity, writeOuts are performed on files starting in the preference list where the migOuts left off. A simulated writeOut simply clears the file's dirty flag. The amount of space "cleaned" by this operation is defaulted to 50% of *maxResident* – a value decided upon after some experimentation. Note that this value has no affect on miss rate as long as it is large enough to avoid migOuts during active periods.

The state of the simulation system following these steps closely represents a storage system in a quiescent state. This is considered an acceptable start condition for file migration simulation, which is essentially a cache simulation with variable-sized cache blocks. There is no need to be concerned with the "cold-start" problem encountered in cache simulation, thanks to the fact that the initial full file system trace contains sizes and last-access times for all the files.

Simulation continues by reading in the first daily trace and performing the appropriate operations on the file entries in the hash table for each trace record encountered. For example, if a write-file record is encountered, and the corresponding file entry has its resident bit set and dirty bit cleared, the dirty bit is set for that file, and *Kdirty* is updated. If a non-resident file is encountered (either a file that was previously migrated out or a new file), we must determine whether performing a migIn will cause Kresident to exceed maxResident. If so, one or more "clean" files must be deleted (set non-resident). Clean files must be deleted in the order they appear in the most recently generated migration preference list. If the system should run out of clean files, dirty files must be made clean before deletion (eg. a migOut). Each encountered record increments a reference counter, and each record that refers to a non-resident file increments a miss counter. These counters are used to determine file miss ratio. After the first daily trace is processed, another migration run is simulated, and the process continues with the second day's trace.

4.2.2 Migration Algorithms

The migration algorithms used in the simulator are based partially on previous published research on the subject [Smit81b, Lawr82]. Only relatively simple algorithms were attempted, as previous research seems to indicate that very complex algorithms resulted in only marginal benefits, if any at all. All the algorithms considered here involve only file size (*s*) and time-since-last-access (*t*). These algorithms include LRU (or simply *time*), as discussed in Section 2.4.3, weighted size-plus-time[6] (*spt*x, where x is a constant weighting factor to be multiplied by *t* in calculating migration preference), weighted size-times-time (*sxt*, where x is a constant weighting factor power to which s is raised, e.g., $s^x t$), and simply *size*.

5 Results

The results generated using the analysis and simulation methods described in the previous section are discussed in the following subsections. Discussions refer to the graphs and tables provided in Appendix A.

[6] The "size plus time" algorithm was motivated by the plan to use it in a new hierarchical storage system/server product from a major computer manufacturer.

5.1 Statistical Analysis Results

Table 1 provides overall statistics for all the file systems traced in this study. The first three columns show "snapshot" statistics of the file systems at the beginning of the trace period. The remaining columns show statistics related to the daily file traces collected. The explanations and discussions of the statistical analysis results (shown in Appendix A) in the following subsections will refer to this table.

File system Name	Number of Files on 8/19/91	Total Kbytes of Files on 8/19/91	Avg. File Size (Kbytes)	Num. of Days Traced[7]	Num. of File Access-Days[8]	Total KB in File Access-Days	Average File Access Size (KB)	% "Write" File Access-Days[9]	% "Write" weighted by size
user1	37,988	427,721	11.26	78	34,603	721,720	20.86	37	67
user4	17,520	334,491	19.09	76	12,625	735,265	58.24	34	62
user6	47,122	827,745	17.57	80	96,211	2,118,861	22.02	22	34
ginger users	21,901	270,862	12.37	79	22,274	578,815	25.99	28	46
pcs	59,923	756,653	12.63	77	31,388	1,044,782	33.29	20	49
x11	13,853	526,139	37.98	84	31,956	4,305,959	134.75	13	10
local	4993	218,823	43.83	81	14,700	1,077,469	73.30	27	8

Table 1: Overall Trace Statistics

5.1.1 File Size Distributions

Pages 3 and 4 of Appendix A show the static file size distributions for the file systems traced[10]. The distributions of the user disks (top graph) show that a large portion of the files are small. On `user1`, for example, over 53,000 of the files (about 90% of the total) are smaller than 5 kilobytes. `User6` has significantly more files larger than 5K, but nearly all are smaller than about 35K. The few very large files on the user disks cause the average file size, shown in Table 1, to be larger than most files on the disk.

[7] The number of days traced differ across filesystems because some traces were corrupted or missed during the trace period. The "ginger users" filesystem had very few missing traces (ginger's backup method doesn't corrupt access times), but tracing started a few days later, which tended to compensate for this.

[8] A "file access-day" is is one or more accesses to a file on a specific day. Thus, a given file can contribute at most one access-day to the sum in this column for each day in the trace period.

[9] A "write" file access-day is one in which at least one of the accesses on that day was a write (eg., modified the file, according to the <u>mtime</u> in the inode of the file). File creations count as writes.

[10] Notes: The small upturn in the distributions at the extreme right of the graphs on page 3 represents all files over 200 kilobytes. The top graph on page 4 shows the same data as the graphs on page 3, except that the file size scale is logarithmic. Thus page 3 provides a more intuitive picture of the distributions, while page 4 may be more useful in comparing the distributions of different filesystems.

The distributions of the "non-user" disks are significantly more skewed toward larger files. For example, compare the curve for X11 with the curve for user4. Both disks have about the same total number of files, but X11 has a much steeper distribution curve. This fact is due to the types of files stored on these file systems. For example, local contains many large binary command files, emacs lisp files, man pages, and command source files. Similarly, X11 contains binary executables for the X windowing system. The exception is the pcs file system, which has a distribution similar to the user disks. This result prompted a closer look at the files on the pcs file system, revealing that it was actually a "user" disk, containing the home directories of about ten users! So clearly, there is a significant difference in file size distributions on file systems that are used for user home directories and those used for common-access files like command binaries. In addition, distributions across different user disks are somewhat similar.

File size distributions weighted by access appear on pages 4 (bottom) and 5 of Appendix A[11]. Read accesses and write accesses are considered separately. For each file system, the solid symbol designates *reads*, and the corresponding hollow symbol designates *writes*. The graphs indicate that on the "user" disks, write accesses tend to be skewed slightly more toward larger files than small ones. The non-user disks on page 6 show a high read-to-write ratio of accesses, and a tendency for many accesses to be to rather large files. In particular, the X11 plot shows a significant portion (about 25%) of read accesses were to files in the 7-to-9 file size range, which translates to files of 128 to 512 kilobytes. The user disks, on the other hand, have a very small number of accesses above log-size 6. These two characteristics of the X11 disk are expected because of the type of files present on this disk – large, binary executables for the windowing system which are rarely changed.

[11] File sizes are expressed as logarithms, and should therefore be compared to the static file size distributions at the top of page 4. File counts are also plotted on a log scale, to make comparisons between filesystems with different total file counts easier.

5.1.2 Access Patterns Over Time

Pages 6 through 12 of Appendix A contain area graphs of file activity over the trace period. The black areas describe the number of files or kilobytes written, while the gray areas describe reads. The top border of the gray area indicates the sum of reads and writes – the total number of file access-days or the total kilobytes associated with the file accesses.

There is clearly a cyclical pattern in the file access graphs of the "user" disks and the pcs disk, with lulls in activity occurring every six to seven days (Table 3 in Appendix A details the mapping of days of the week to the day numbers shown on the graphs). By comparing the day-of-the-week column in the table with the location of the "valleys" in the graph, we see that periods of inactivity tend to occur on weekends, which is expected. The local and X11 disks, however, do not appear to have a strong cyclic pattern – file use is more evenly distributed across different days. Files on these disks are mostly executables for commands and the X windowing system. We would expect that these files are accessed more often during the week than on weekends, but recall that the trace collection method recognizes only one access of a file per day. So as long as most "active" files are accessed at least once on weekend days, they will appear in the graph. Thus the graphs indicate that there is certainly some user activity during weekends – enough that many system-related binaries are accessed at least once per day.

It is interesting to compare the height of the peaks of these graphs with the total number of files or Kbytes on the file system (table 1) to see what percentage of files or disk space are accessed each day. For all the disks, the highest peaks represent roughly between 5% and 20% of the totals for files and kilobytes of disk space. There doesn't seem to be a tendency for non-user disks, as compared with user disks, to have a higher percentage of their files accessed each day. This may indicate that a large portion of the files on these disks are seldom accessed, if we assume that the same set of files are accessed every day. This is a reasonable assumption considering the small variation in access volumes from day to day, including weekends, and considering the access patterns we would expect on the files stored on these disks. However, this assumption can't be confirmed with this data set – file interreference data and simulation data are needed to determine if the set of "active" files changes slowly or quickly over time.

There are some particularly high peaks of activity on some days of certain file systems that should be explained. For example, a large spike of activity on day 40 on X11 was the result of updating some X-windows utility sources. On the local file system, a single spike of activity at day 38 and a number of active days near the end of the trace period were the result of various builds, including a recompilation of emacs. Large "write" spikes on local and X11 are usually due to a system administrator updating or adding new utility or application software. We found that the large spikes of activity on user disks are often the result of a user copying many files to or from a different disk or to tape. A large "write" spike at day 32 on user6 was apparently caused by copying a 129 Mbyte tar file, probably from an 8mm tape. Large spikes seem to occur not because a tremendous amount of computation is being performed on files, but rather because users are moving many files from one disk or tape to another. If a migration system were in place, many of these operations probably should be performed by accessing the archive device directly. For example, if a user desires to copy many large data files (which are currently not resident on disk) to an 8mm tape to send to a colleague, it would not make sense to copy all the files to disk before copying to tape. By providing direct user access to the archive device, many of these large activity spikes might be completely hidden from the migration process.

5.1.3 File Interreference Intervals

File interreference distributions appear on pages 13–19 of Appendix A. There are two graphs for each file system to distinguish between interreference patterns for reads and writes[12]. Each graph represents ten different interreference distributions, one for each file

[12] There were so few write accesses to the X11 disk that the corresponding interreference distributions were deemed non-useful. Therefore, no interreference graph is presented for X11 writes.

size group[13]. The data is displayed as a contour graph, with the "z" axis representing the normalized cumulative distribution of interreference intervals. The "z" values for each grayscale level are shown in the legend. For example, in the graph for user1 reads, the grayscale level boundary at file size group two (x-axis) and interreference time 10 days (y-axis) represents a cumulative distribution of about 0.6. This indicates that 60% of intervals between "read" file accesses to files in size group two (4–7 kilobytes) are 10 or fewer days long.

All of the graphs, with the exception of those for the pcs file system, indicate that more than half of the interreference intervals for all file sizes are no longer than a few days. This indicates a high degree of temporal locality of access on these file systems. One possible explanation for the pcs file system's notably different overall distributions is that pcs had far fewer file accesses than the other file systems, as a percentage of the total number of files resident at the start of the trace period (see table 1). In fact, on a number of the days during the trace period, there were no accesses to files on pcs. In general, accesses to pcs appear to be rather bursty (page 10 of Appendix A). This burstiness could explain the steep transitions around 32 and 47 days on the "reads" graph, and at 22 and 47 days on the "writes" graph.

There does not seem to be a consistent, strong correlation between the shapes of interreference interval distributions and file size. The widest and narrowest distributions for each file system occur within different file size groups on different file systems. Furthermore, there does not seem to be a strong trend of distributions narrowing or widening as file size increases. Therefore, file size does not seem to be a good predictor of temporal locality of access. This result is in contrast to Smith's findings in [Smit81a]. He found that for his data, large files tend to have shorter interreference intervals than small files.

[13] Note that the accuracy of distributions is dependent on the total number of accesses to files in that file size group. Because there are generally a small number of files in the largest file size groups, these distributions are statistically less accurate than those of smaller file size groups. For this reason, it is useful to refer to the cumulative file size distributions weighted by access that appear on pages 4 and 5 of appendix A when interpreting these graphs.

For the "user" file systems and `pcs`, interreference intervals are generally shorter for writes than for reads, as indicated by the heavier clustering of the darker grayscales toward the top of the graphs of the distributions for writes. For the non-user filesystems `X11` and `local`, however, the opposite is true. As emphasized before, these disks tend to have a higher read-to-write ratio, because they contain primarily binary executable files for X-windows and various commands available on Sprite, which are seldom modified. This fact would explain longer interreference intervals for writes. Also, because the "write" distributions for these two filesystems are based on a small number of accesses, there is a larger margin of error. One or two long interreference intervals in a given file size group could stretch the distribution out toward the bottom of the graph. This is also the reason for the steep transitions in the distributions for these filesystems.

5.2 Simulation Results - Comparison of Migration Algorithms

The results of the migration simulation runs are shown on pages 20–26 of Appendix A. All the graphs show file miss ratio versus simulated magnetic disk size. The top graph of each page shows a subset of the algorithms considered, including the best overall algorithm[14] (always one of the size-times-time algorithms), the best size-plus-weighted-time algorithm, and the pure size and pure time algorithms. File miss ratio is plotted on a linear scale for these graphs, to emphasize the location of the "knee" of the curves. The bottom graphs include most or all of the algorithms considered. These graphs show file miss ratio plotted on a logarithmic scale, to make comparisons between different algorithms easier.

The abbreviations in the legend for size-times-time algorithms are s, for size[15], followed by a size weighting factor, and ending in t, for time[16]. The size weighting factor is the power to which s is raised, eg. "s12t" represents $s^{1.2}t$, "s6t" represents $s^{0.6}t$, and "s10t" is simply st. Similarly, size-plus-time algorithms are designated "spt," followed by the weighting factor

[14] The "best" algorithm is the one corresponding to the curve with the lowest file miss ratio.

[15] Size is measured in kilobytes.

[16] Time is measured in days.

for *t*, eg. "spt5" represents s+5t, and "spt" represents simply s+t. We chose the best overall range of weighting factors for these two algorithms by trying a number of weights on each of the filesystems. The migration algorithms based purely on time-since-last-access and purely on file size are designated "time" and "size," respectively.

The best algorithm for all the filesystems traced is consistently a size-times-time algorithm. However, the *s* weighting factors of the best algorithms differ across filesystems. The distance between the various size-times-time algorithms is relatively small, indicating that choosing any weighting factor (close to 1.0) would probably result in reasonable hit rates. The only filesystem for which a size weighting factor larger than 1.0 is best is X11. A weighting factor less than 1.0 is best for all the "user" disks on Sprite, including pcs, while 1.0 is best for the ginger users disk. As we saw earlier, the file size distribution weighted by access for X11 (page 5, Appendix A) was notably different than the same distributions for other filesystems. Because the distribution of file sizes on the X11 disk stretches out toward larger files much more than on other filesystems, basing file removal on size more heavily than on time tends to clear more space. Also, recall that the "active" files on X11 are accessed very frequently (see interreference intervals on page 13, Appendix A), so *time* may not be as useful a value. In fact, we can see that the pure size algorithm performed nearly as well as the best algorithm for X11, while the pure time algorithm (LRU) performed rather poorly.

The size-plus-time algorithms performed significantly more poorly than the size-times-time algorithms for all the filesystems. The time weighting factor for the best size-plus-time algorithm for each filesystem varied significantly across filesystems. For example, the best factor for /home/ginger/users was 1, while the best for user4 was 10, a factor of 10 difference. This suggests that the size-plus-time algorithm may be more dependent on an appropriate weighting factor than size-times-time, which varied only by a factor of 2 between the two extreme cases. In any case, because size-times-time can be computed as easily as size-plus-time, and because it consistently performs better, there seems to be no reason to consider size-plus-time further.

The pure size and pure time algorithms performed considerably more poorly than the other algorithms, and should therefore be avoided. However, it is interesting to note that without exception, the pure size algorithm performed much better than pure time. Thus file size is clearly an important factor in determining migration, and should not be ignored. Note that there are performance issues besides file miss rate that should be considered in choosing a migration algorithm. For example, if a size-based algorithm is used, the files that are migrated will tend to be the largest files in the system, and will therefore incur a larger average transfer time. However, because latency, not bandwidth, is expected to be the performance bottleneck on archive devices, migrating large files is to our advantage. Migrating many small files is far more detrimental to overall performance than migrating a few large files, because long latencies must be endured more often.

Filesystem Name	Disk-Archive Ratio for 5% Miss Ratio	Disk-Archive Ratio for 1% Miss Ratio
user1	0.12	0.40
user4	0.19	0.60
user6	0.21	0.46
ginger users	0.20	0.48
pcs	0.22	0.50
x11	0.53	0.76
local	0.27	0.46

Table 2: Disk-to-archive ratios necessary to obtain specific file miss ratios.

It is useful to compare the simulated disk size at which file miss ratio reaches a reasonable value with the total size of the filesystem traced for that simulation. This gives an indication of the ratio between disk and archive that is needed to obtain reasonable performance. The knee of the best curve for each filesystem seems to occur consistently at a miss ratio of about 0.05, so we will use this value to compare the ratio on different filesystems. We will also use a miss ratio of 0.01, or 1%, to represent a tradeoff of cost for performance. These values are shown in Table 2.

The data in Table 2 indicates that the disk-archive ratios needed to obtain a given miss ratio are very consistent across filesystems, with the exception of X11. The high values for X11 indicate that a large portion of this filesystem is active, and that the cost benefits of setting up a migration system for this filesystem are relatively small. For the other filesystems, if a miss ratio of about 5% is acceptable, only about 20% of the total filesystem space need be on magnetic disk. Even if 1% is the highest tolerable miss ratio, half of the disk space occupied by files in these filesystems could be migrated to archive media.

5.3 Comparison with Previous Published Research Results

In [Smit81b], Smith found the space-time product algorithm to work well, although not quite as well as his "stochastically optimal" algorithm, which is significantly more complex to compute. For his data, taken from the Stanford Linear Accelerator Center (SLAC) around 1977, exponentially weighting the time-since-last-access by values between 1.3 and 1.6 worked best. This is equivalent to weighting the size with the inverse of these values – between 0.63 and 0.77. This is the same range of values we observed for all but one of the "user" disks. In [Lawr82], Lawrie et. al. also found the space-time product to work well. For their data, a weighting factor of 1.0 worked best, which is what we observed for the remaining "user" disk.

6 Conclusions

In this study, we have developed a straightforward strategy for collecting file access trace data from a live system. We can then analyze the data to investigate file usage patterns and evaluate migration algorithms. We found that most of the files on the systems studied were small, with the exception of one file system that contained many large binary executable files rather than user files. In comparing static file size distributions with file size distributions weighted by access, we found

File access activity per day over the trace period tended to be rather "bursty" on all file systems, with the lowest periods of activity occurring on the weekends. This burstiness indicates that nightly migration runs must often overestimate the amount of disk space that will be used the following day to avoid running out of "clean" files to delete during active periods. We also observed that while there were more file read accesses than file writes, most of the blocks accessed were associated with file writes. We therefore concluded that write accesses tended to be associated with larger files than read accesses.

We found that most file interreference intervals were rather short – usually less than five days. We did not find a strong correlation between file size and file interreference interval length, though the distribution of intervals did differ with file size for each file system. Almost invariably, file interreference intervals for writes were shorter than for reads.

Our simulation results indicate that algorithms based on the product of file size and time-since-last-access work well – significantly better than the other algorithms we considered. The best weighting factor on size in the size-time product algorithm varied across file systems, though all weightings performed relatively well. We found that for file systems with many large files that were accessed frequently, weighting size more heavily in the product worked best.

7 Further Study

There are a number of aspects of hierarchical storage systems that remain to be studied in depth. Some studies could make use of the traces collected in this study, while others would require more detailed and precise file access traces. The following sections describe a few enhancements that we considered, but did not have the time or resources to complete.

7.1 File Chunking

As we discussed in Section 2.4.3, some files may be large enough that migrating the entire file to or from archive could be detrimental to overall storage system performance. There are two reasons for this. First, transferring large files could keep the archive device busy for a long time, possibly delaying the servicing of other requests. Second, migrating a large file from archive to disk could force many smaller, "active" files off the disk, thus reducing overall file hit rates. One solution to this problem is to "chunk" large files into smaller pieces, each of which is migrated separately. The simulator developed for this study could be enhanced to handle file chunking. These enhancements would include determining which files should be chunked, and how large each chunk should be. Chunk size should be a simulator parameter that can be varied to compare strategies. The simulator would then handle each chunk as a separate file.

7.2 Disk-To-Archive Traffic Analysis

If a size-times-time migration algorithm is employed, the larger files in the file system tend to be the ones moved between disk and archive. It may be useful to investigate the distribution of sizes of files moved to determine how much bandwidth should be provided between the disk and archive devices. This could be particularly important if many file systems migrate large files across a single network. Using the traces collected in this study, file size distribution graphs, such as those on pages 3–5 of Appendix A, could be generated by enhancing the simulator program.

To fully understand the bandwidth and latency needs of a hierarchical storage server based on file reference traces, the simulator would need to be enhanced with queueing models. This enhancement would require a different trace collection scheme – one in which the exact time of each file access request was recorded. This would involve instrumenting the file system code to write a trace record each time a file is accessed.

8 References

[Denn72] P. J. Denning, "On Modelling Program Behavior," *Proceedings of the Spring Joint Computer Conference*, AFIPS Press, Reston, VA, 1972, pp. 937-944.

[Lawr82] D. H. Lawrie, J. M. Randal, and R. R. Barton, "Experiments with Automatic File Migration," IEEE Computer, July 1982, pp. 45-55 (1982).

[Smit81a] Alan Jay Smith, "Analysis of long term file reference patterns for application to file migration algorithms." *IEEE Transactions on Software Engineering* SE-7(4): 403-417, 1981.

[Smit81b] Alan Jay Smith, "Long term file migration: development and evaluation of algorithms." *Communications of the ACM* 24(8): 521-532, 1981.

9 Further Reading

[Boyd78] Donald L. Boyd, "Implementing Mass Storage Facilities in Operating Systems," IEEE Computer, February 1978, pp. 40-45 (1978).

[Coll88a] Bill Collins and Marjorie Devaney, "Profiles in Mass Storage: A Tale of Two Systems," *Digest of Papers*, Proc. Ninth IEEE Symposium on Mass Storage Systems, November 1988, pp. 61-67 (1988).

[Coll88b] Bill Collins and Catherine Mexal, "The Los Alamos Common File System," *Tutorial Notes*, Proc. Ninth IEEE Symposium on Mass Storage Systems, November 1988, pp. 171-180 (1988).

[Cope83] Lee Copeland, "Monitoring the Performance and Capacity of the IBM 3850 mass Storage System," Proceedings of the Computer Measurement Group, 1983, pp. 45-50 (1983).

[Hend90] Robert L. Henderson and Alan Poston, "MSS-II and RASH: a mainframe UNIX based mass storage system with a rapid access storage hierarchy file management system." *USENIX Winter 1989 Conference*, San Diego, California, January, 1990, pp 65-84.

[Hevn85] Alan R. Hevner, "Evaluation of Optical Disk Systems for Very Large Database Applications," Proceedings of Sigmetrics, May 1985, pp. 166-172 (1985).

[Katz89] Randy H. Katz, Garth A. Gibson, and David A. Patterson, "Disk System Architectures for High Performance Computing," *Proceedings of the IEEE* 77(12), December 1989, pp. 1842-1858 (1989).

[Kenl90] Gregory G. Kenley, "An Architecture for a Transparent Networked Mass Storage System," *Digest of Papers*, Proc. Tenth IEEE Symposium on Mass Storage Systems, May 1990, pp. 160-167 (1990).

[Mill90] Steve Miller and Sam Coleman, "Mass Storage System Reference Model: Version 4 (May, 1990)," IEEE Technical Committee on Mass Storage Systems and Technology, May 1990.

[Muus88] Michale Muuss, et al, "BUMP, The BRL/USNA Migration Project," *Proceedings, Workshop on Unix and Supercomputers*, pp. 183-214, USENIX, September 26, 1988.

[Nels87] Marc Nelson, David L. Kitts, John H. Merrill and Gene Harano, "The NCAR Mass Storage System," *Digest of Papers*, Proc. Eighth IEEE Symposium on Mass Storage Systems, May 1987, pp. 12-20 (1987).

[Oust85] John K. Ousterhout et al., "A trace-driven analysis of the UNIX 4.2 BSD file system." *Operating Systems Review* **19**(5): 15-24, 1985.

[Patt87] D. A. Patterson, G. Gibson, and R. H. Katz, *A case for redundant arrays of inexpensive disks*, Report No. UCB/CSD 87/391 Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, 1987.

[Rich88] J. Richards, "A Unix-MVS Based Mass Storage System for Supercomputers," *Digest of Papers*, Proc. Ninth IEEE Symposium on Mass Storage Systems, November 1988, pp. 108-113 (1988).

[Smit85] Alan Jay Smith, "Disk cache - miss ratio analysis and design consideration," *ACM Transactions on Computer Systems* **3**(3): 161-203, August 1985.

[Than88] Erich Thanhardt and Gene Harano, "File Migration in the NCAR Mass Storage System," *Digest of Papers*, Proc. Ninth IEEE Symposium on Mass Storage Systems, November 1988, pp. 114-121 (1988).

[Twet90] David Tweten, "Hiding Mass Storage Under Unix: NASA's MSS-II Architecture," *Digest of Papers*, Proc. Tenth IEEE Symposium on Mass Storage Systems, May 1990, pp. 140-145 (1990).

Appendix A - Analysis Results (Tables and Graphs)

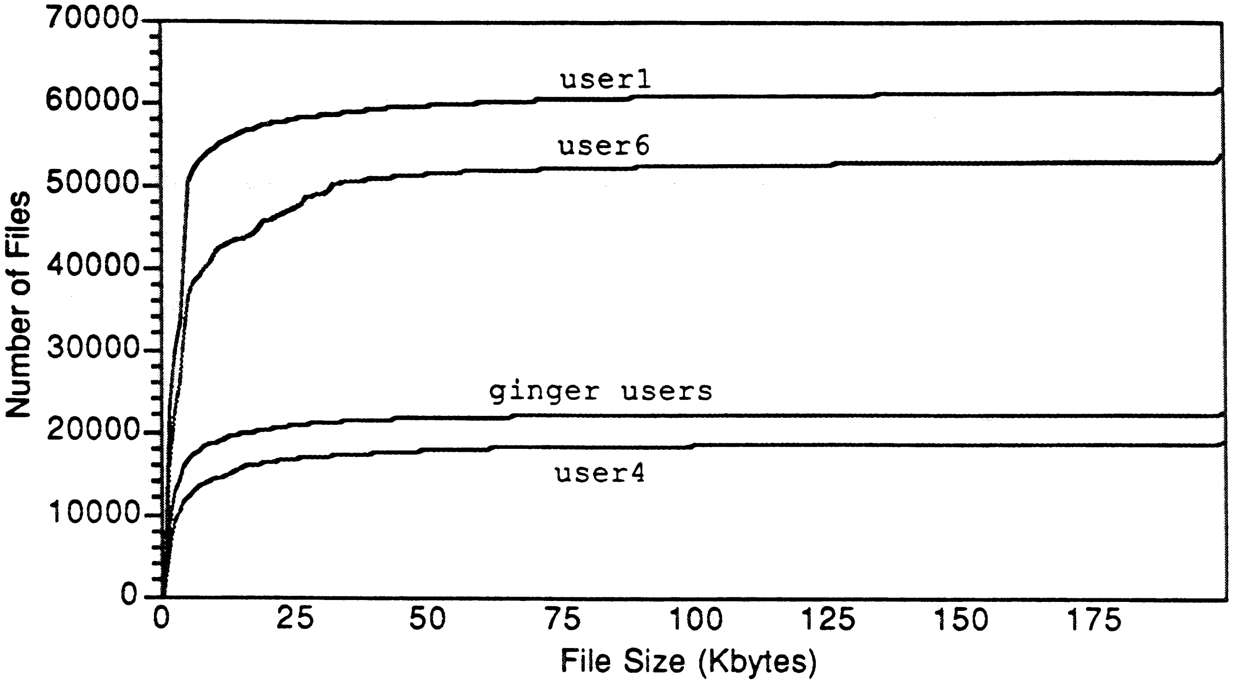
Day Num.	User1		User4		User6		Ginger Users		pcs		X11		local	
	day	date	day	date	day	date	day	date	day	date	day	date	day	date
1	Mon	8/19	Mon	8/19	Mon	8/19	Wed	9/4	Mon	8/19	Tue	8/20	Mon	8/19
2	Tue	8/20	Tue	8/20	Tue	8/20	Thu	9/5	Tue	8/20	Wed	8/21	Tue	8/20
3	Wed	8/21	Wed	8/21	Wed	8/21	Fri	9/6	Wed	8/21	Thu	8/22	Wed	8/21
4	Thu	8/22	Thu	8/22	Thu	8/22	Sat	9/7	Thu	8/22	Fri	8/23	Thu	8/22
5	Sat	8/24	Sat	8/24	Fri	8/23	Sun	9/8	Sat	8/24	Sat	8/24	Fri	8/23
6	Sun	8/25	Sun	8/25	Sat	8/24	Mon	9/9	Sun	8/25	Sun	8/25	Sat	8/24
7	Mon	8/26	Mon	8/26	Sun	8/25	Tue	9/10	Mon	8/26	Mon	8/26	Sun	8/25
8	Tue	8/27	Tue	8/27	Mon	8/26	Wed	9/11	Tue	8/27	Tue	8/27	Mon	8/26
9	Wed	8/28	Wed	8/28	Tue	8/27	Thu	9/12	Wed	8/28	Thu	8/29	Tue	8/27
10	Fri	8/30	Fri	8/30	Wed	8/28	Fri	9/13	Thu	8/29	Fri	8/30	Wed	8/28
11	Sat	8/31	Sat	8/31	Fri	8/30	Sat	9/14	Fri	8/30	Sat	8/31	Thu	8/29
12	Sun	9/1	Sun	9/1	Sat	8/31	Sun	9/15	Sat	8/31	Sun	9/1	Fri	8/30
13	Mon	9/2	Mon	9/2	Sun	9/1	Mon	9/16	Sun	9/1	Mon	9/2	Sat	8/31
14	Tue	9/3	Tue	9/3	Mon	9/2	Tue	9/17	Mon	9/2	Tue	9/3	Mon	9/2
15	Wed	9/4	Thu	9/5	Tue	9/3	Wed	9/18	Tue	9/3	Wed	9/4	Tue	9/3
16	Thu	9/5	Fri	9/6	Wed	9/4	Thu	9/19	Wed	9/4	Thu	9/5	Wed	9/4
17	Fri	9/6	Sat	9/7	Fri	9/6	Fri	9/20	Sat	9/7	Fri	9/6	Thu	9/5
18	Sun	9/8	Mon	9/9	Sat	9/7	Sat	9/21	Mon	9/9	Sat	9/7	Fri	9/6
19	Mon	9/9	Thu	9/12	Sun	9/8	Sun	9/22	Tue	9/10	Sun	9/8	Sat	9/7
20	Tue	9/10	Sat	9/14	Mon	9/9	Mon	9/23	Thu	9/12	Mon	9/9	Sun	9/8
21	Thu	9/12	Sun	9/15	Tue	9/10	Tue	9/24	Sat	9/14	Tue	9/10	Mon	9/9
22	Sat	9/14	Mon	9/16	Thu	9/12	Wed	9/25	Mon	9/16	Thu	9/12	Thu	9/12
23	Sun	9/15	Tue	9/17	Sat	9/14	Thu	9/26	Tue	9/17	Fri	9/13	Fri	9/13
24	Mon	9/16	Wed	9/18	Sun	9/15	Fri	9/27	Wed	9/18	Sun	9/15	Sun	9/15
25	Tue	9/17	Thu	9/19	Mon	9/16	Sat	9/28	Thu	9/19	Mon	9/16	Mon	9/16
26	Thu	9/19	Sun	9/22	Tue	9/17	Sun	9/29	Fri	9/20	Tue	9/17	Tue	9/17
27	Sun	9/22	Mon	9/23	Wed	9/18	Mon	9/30	Sun	9/22	Wed	9/18	Wed	9/18
28	Mon	9/23	Tue	9/24	Thu	9/19	Tue	10/1	Mon	9/23	Thu	9/19	Thu	9/19
29	Tue	9/24	Wed	9/25	Sun	9/22	Wed	10/2	Tue	9/24	Fri	9/20	Fri	9/20
30	Wed	9/25	Thu	9/26	Mon	9/23	Thu	10/3	Wed	9/25	Sat	9/21	Sun	9/22
31	Thu	9/26	Sat	9/28	Tue	9/24	Fri	10/4	Sat	9/28	Sun	9/22	Mon	9/23
32	Sat	9/28	Sun	9/29	Wed	9/25	Sat	10/5	Sun	9/29	Mon	9/23	Tue	9/24
33	Sun	9/29	Mon	9/30	Thu	9/26	Mon	10/7	Tue	10/1	Tue	9/24	Wed	9/25
34	Mon	9/30	Tue	10/1	Sat	9/28	Tue	10/8	Wed	10/2	Wed	9/25	Thu	9/26
35	Tue	10/1	Wed	10/2	Sun	9/29	Wed	10/9	Thu	10/3	Thu	9/26	Fri	9/27
36	Wed	10/2	Thu	10/3	Mon	9/30	Thu	10/10	Fri	10/4	Fri	9/27	Sun	9/29
37	Thu	10/3	Sat	10/5	Tue	10/1	Fri	10/11	Sat	10/5	Sun	9/29	Mon	9/30
38	Sat	10/5	Sun	10/6	Wed	10/2	Sat	10/12	Sun	10/6	Mon	9/30	Tue	10/1
39	Sun	10/6	Mon	10/7	Thu	10/3	Sun	10/13	Wed	10/9	Tue	10/1	Wed	10/2
40	Mon	10/7	Tue	10/8	Sat	10/5	Mon	10/14	Fri	10/11	Thu	10/3	Thu	10/3
41	Tue	10/8	Wed	10/9	Sun	10/6	Tue	10/15	Sat	10/12	Fri	10/4	Fri	10/4
42	Wed	10/9	Thu	10/10	Mon	10/7	Wed	10/16	Sun	10/13	Sun	10/6	Sun	10/6

Table 3: Dates of File Activity Captured in Traces (continued on next page)

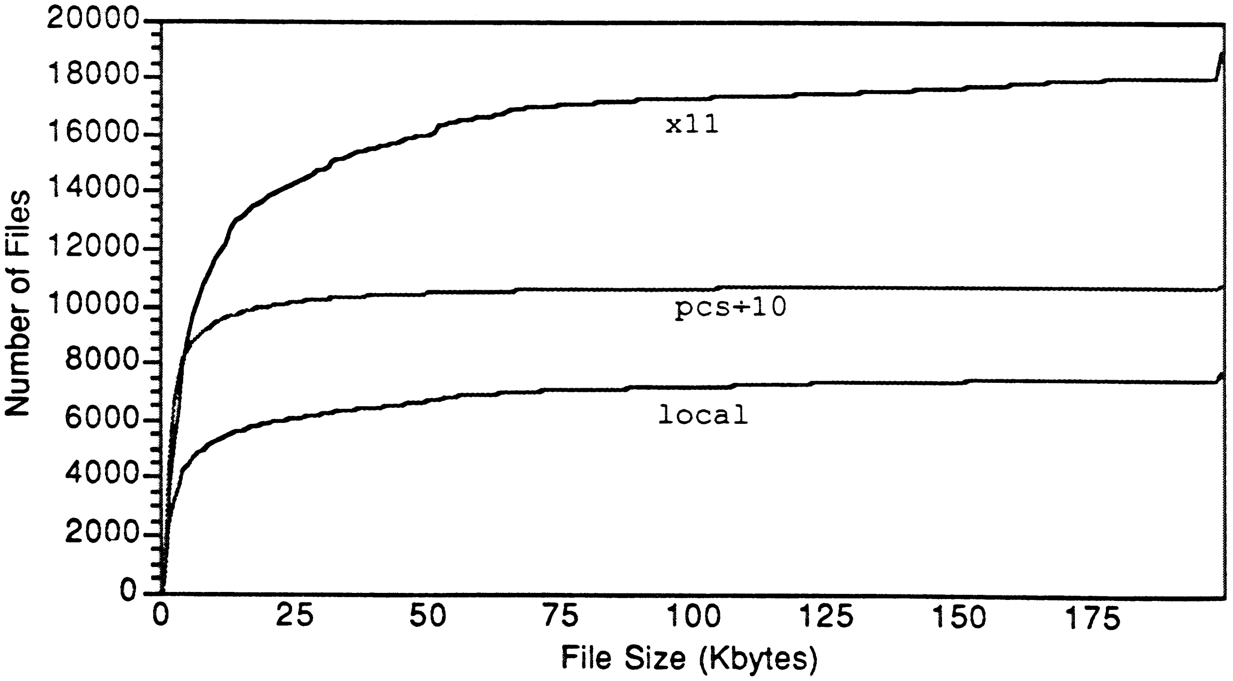
Day Num.	User1		User4		User6		Ginger Users		pcs		X11		local	
	day	date	day	date	day	date	day	date	day	date	day	date	day	date
43	Thu	10/10	Sat	10/12	Tue	10/8	Thu	10/17	Mon	10/14	Tue	10/8	Mon	10/7
44	Sat	10/12	Sun	10/13	Wed	10/9	Fri	10/18	Wed	10/16	Wed	10/9	Tue	10/8
45	Sun	10/13	Mon	10/14	Sat	10/12	Sat	10/19	Thu	10/17	Thu	10/10	Wed	10/9
46	Mon	10/14	Wed	10/16	Sun	10/13	Sun	10/21	Fri	10/18	Fri	10/11	Fri	10/11
47	Wed	10/16	Thu	10/17	Mon	10/14	Tue	10/22	Sat	10/19	Sat	10/12	Sun	10/13
48	Thu	10/17	Fri	10/18	Wed	10/16	Wed	10/23	Mon	10/21	Sun	10/13	Mon	10/14
49	Fri	10/18	Sat	10/19	Thu	10/17	Thu	10/24	Tue	10/22	Mon	10/14	Wed	10/16
50	Sat	10/19	Tue	10/22	Fri	10/18	Fri	10/25	Wed	10/23	Wed	10/16	Thu	10/17
51	Mon	10/21	Wed	10/23	Sat	10/19	Sat	10/26	Thu	10/24	Thu	10/17	Fri	10/18
52	Tue	10/22	Thu	10/24	Tue	10/22	Sun	10/27	Fri	10/25	Fri	10/18	Sat	10/19
53	Wed	10/23	Fri	10/25	Wed	10/23	Mon	10/28	Sat	10/26	Sat	10/19	Sun	10/21
54	Thu	10/24	Sun	10/27	Thu	10/24	Tue	10/29	Sun	10/27	Sun	10/21	Wed	10/23
55	Fri	10/25	Mon	10/28	Fri	10/25	Wed	10/30	Mon	10/28	Wed	10/23	Thu	10/24
56	Sun	10/27	Tue	10/29	Sun	10/27	Thu	10/31	Tue	10/29	Thu	10/24	Fri	10/25
57	Mon	10/28	Wed	10/30	Mon	10/28	Fri	11/1	Wed	10/30	Fri	10/25	Sat	10/26
58	Tue	10/29	Thu	10/31	Tue	10/29	Sat	11/2	Thu	10/31	Sat	10/26	Mon	10/28
59	Wed	10/30	Fri	11/1	Wed	10/30	Sun	11/3	Fri	11/1	Sun	10/27	Tue	10/29
60	Thu	10/31	Sat	11/2	Thu	10/31	Mon	11/4	Sat	11/2	Mon	10/28	Wed	10/30
61	Fri	11/1	Mon	11/4	Fri	11/1	Tue	11/5	Tue	11/5	Tue	10/29	Thu	10/31
62	Sat	11/2	Tue	11/5	Sat	11/2	Wed	11/6	Wed	11/6	Wed	10/30	Fri	11/1
63	Mon	11/4	Wed	11/6	Mon	11/4	Thu	11/7	Thu	11/7	Thu	10/31	Sat	11/2
64	Tue	11/5	Fri	11/8	Tue	11/5	Fri	11/8	Fri	11/8	Fri	11/1	Sun	11/3
65	Wed	11/6	Sat	11/9	Wed	11/6	Sat	11/9	Sat	11/9	Sat	11/2	Tue	11/5
66	Thu	11/7	Mon	11/11	Thu	11/7	Sun	11/10	Mon	11/11	Tue	11/5	Wed	11/6
67	Fri	11/8	Tue	11/12	Fri	11/8	Mon	11/11	Tue	11/12	Wed	11/6	Thu	11/7
68	Sat	11/9	Wed	11/13	Sat	11/9	Tue	11/12	Wed	11/13	Thu	11/7	Fri	11/8
69	Mon	11/11	Thu	11/14	Mon	11/11	Wed	11/13	Thu	11/14	Fri	11/8	Sat	11/9
70	Tue	11/12	Fri	11/15	Tue	11/12	Thu	11/14	Fri	11/15	Sat	11/9	Sun	11/10
71	Wed	11/13	Sat	11/16	Wed	11/13	Fri	11/15	Sat	11/16	Sun	11/10	Mon	11/11
72	Thu	11/14	Sun	11/17	Thu	11/14	Sat	11/16	Sun	11/17	Mon	11/11	Tue	11/12
73	Fri	11/15	Mon	11/18	Fri	11/15	Sun	11/17	Tue	11/19	Tue	11/12	Wed	11/13
74	Sat	11/16	Tue	11/19	Sat	11/16	Mon	11/18	Wed	11/20	Wed	11/13	Thu	11/14
75	Mon	11/18	Wed	11/20	Sun	11/17	Tue	11/19	Thu	11/21	Thu	11/14	Fri	11/15
76	Tue	11/19	Thu	11/21	Mon	11/18	Wed	11/20	Sat	11/23	Fri	11/15	Sat	11/16
77	Wed	11/20			Wed	11/20	Thu	11/21	Sun	11/24	Sat	11/16	Sun	11/17
78	Thu	11/21			Thu	11/21	Fri	11/22			Sun	11/17	Wed	11/20
79					Sun	11/24	Sat	11/23			Tue	11/19	Thu	11/21
80					Mon	11/25					Wed	11/20	Sat	11/23
81											Thu	11/21	Sun	11/24
82											Fri	11/22		
83											Sat	11/23		
84											Sun	11/24		

Table 3: Dates of File Activity Captured in Traces (continued from page 1)

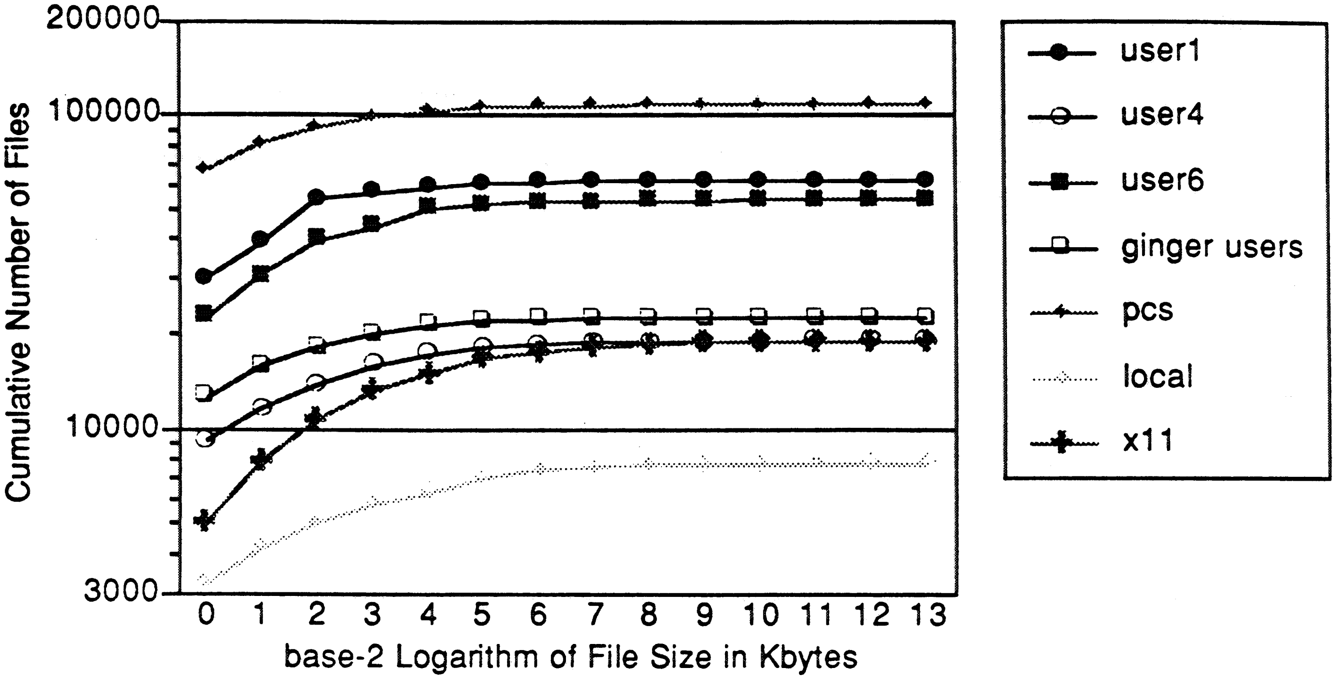
Cumulative File Size Distributions – User Disks



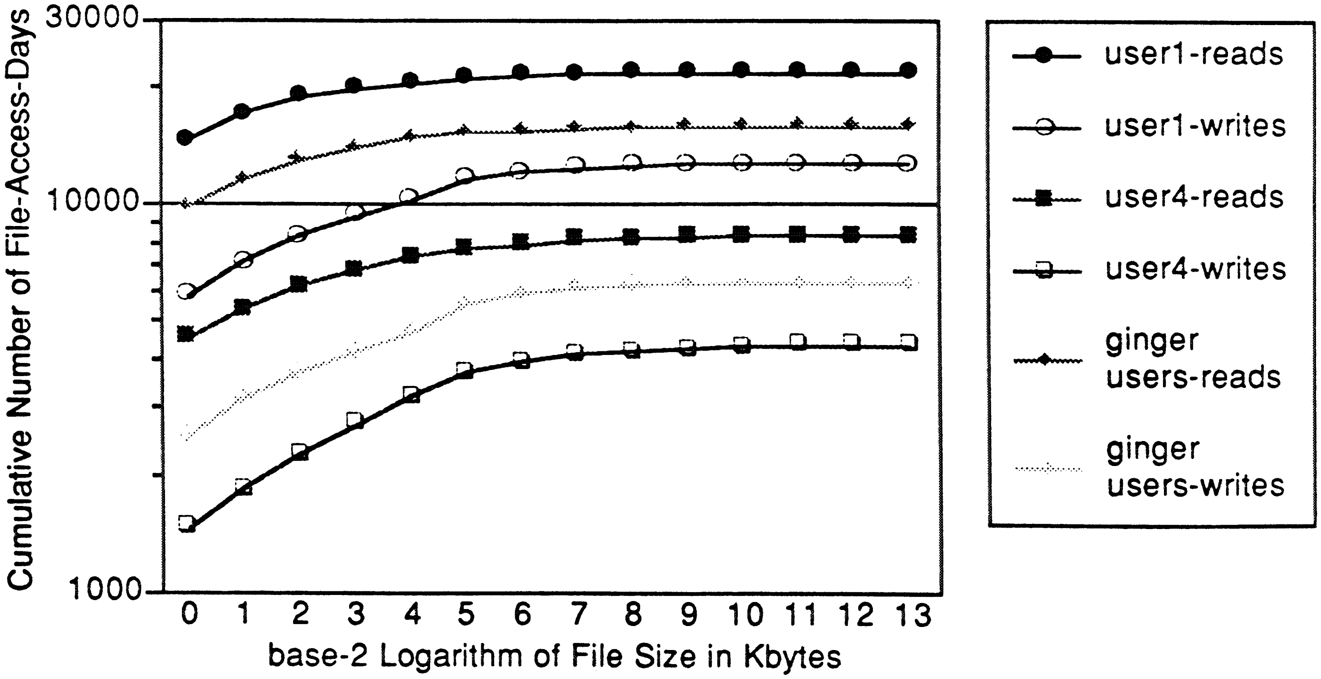
Cumulative File Size Distributions – "non-user" Disks

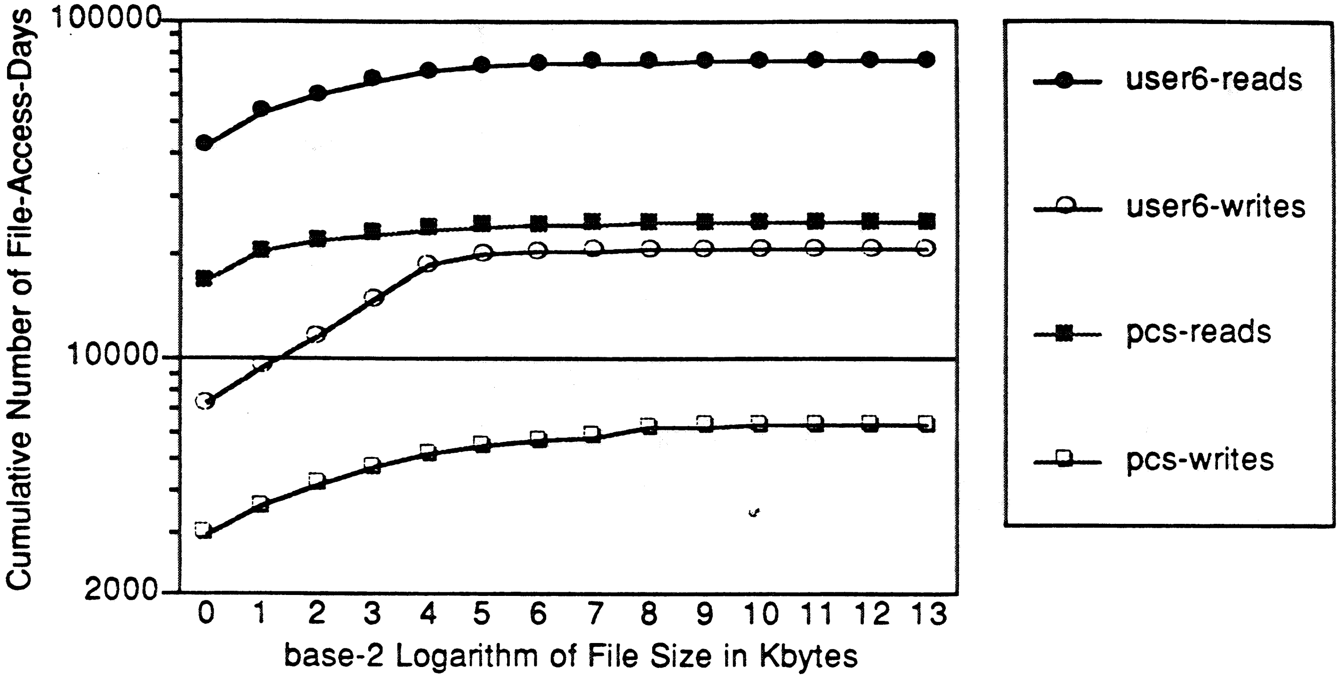


Cumulative File Size Distributions – All Disks (log-log scale)

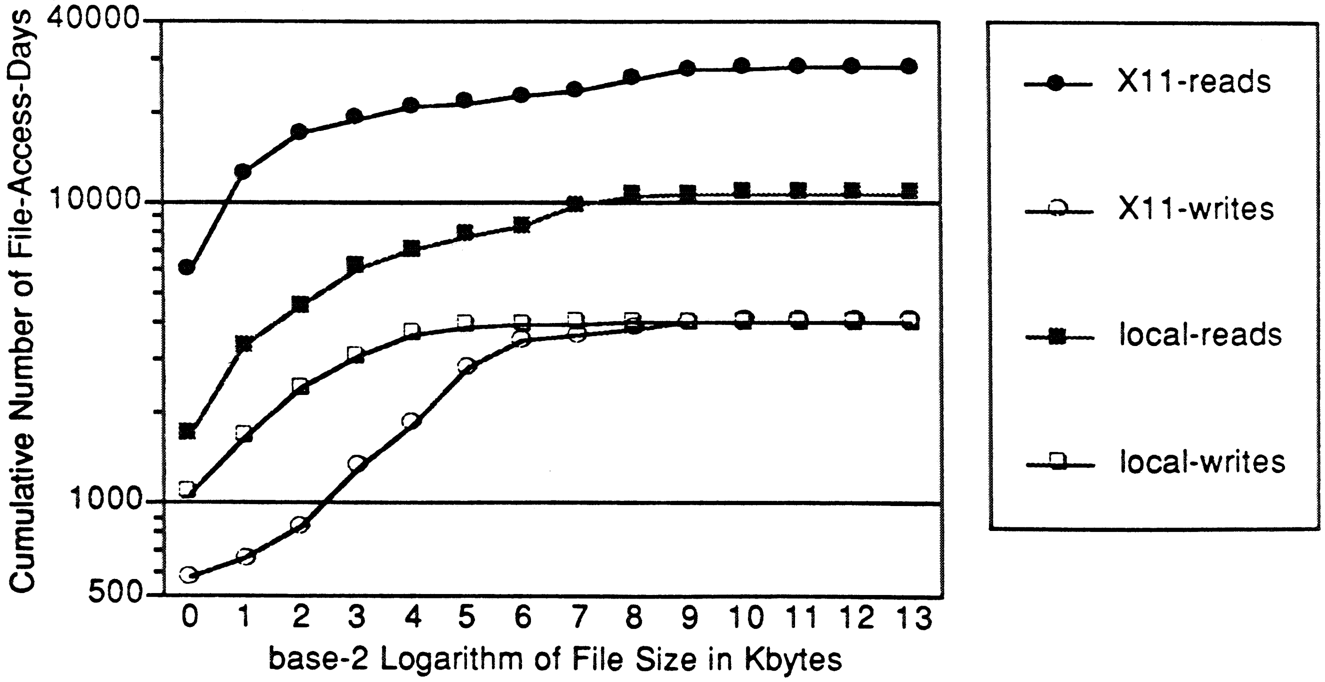


Cumulative File Size Distributions Weighted by Access (log-log)



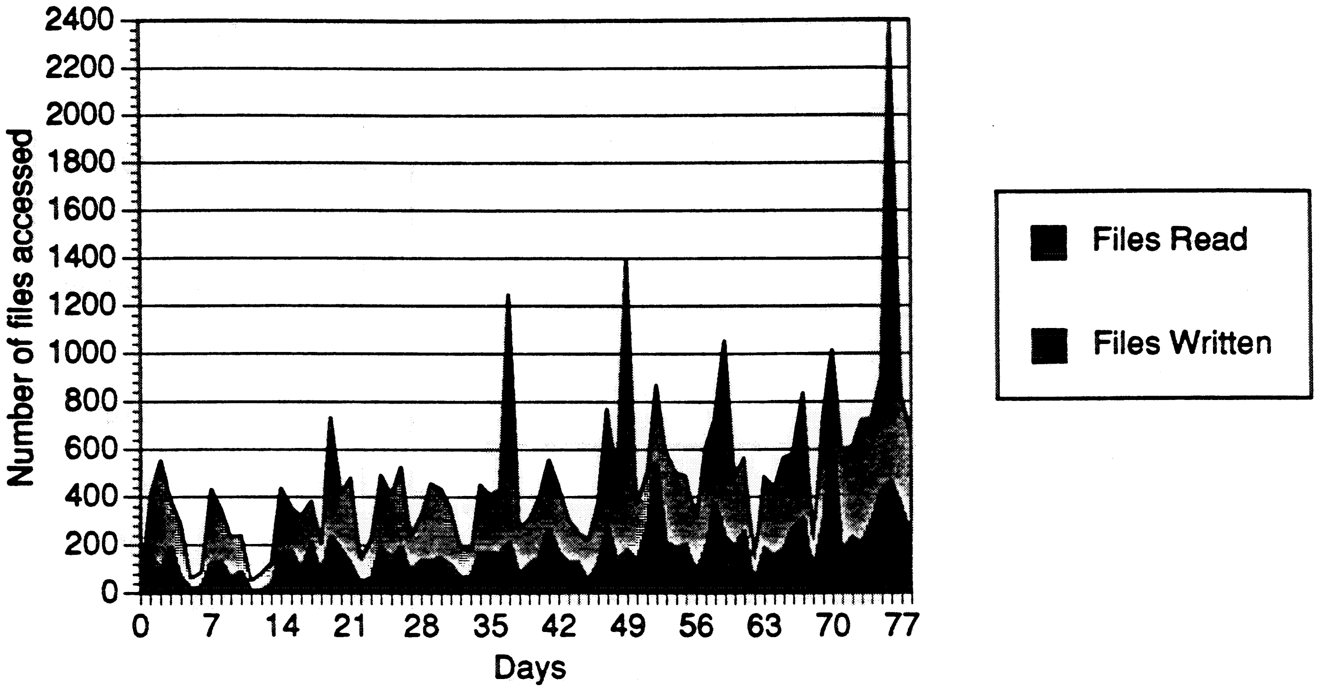


Cumulative File Size Distributions Weighted by Access (log-log)

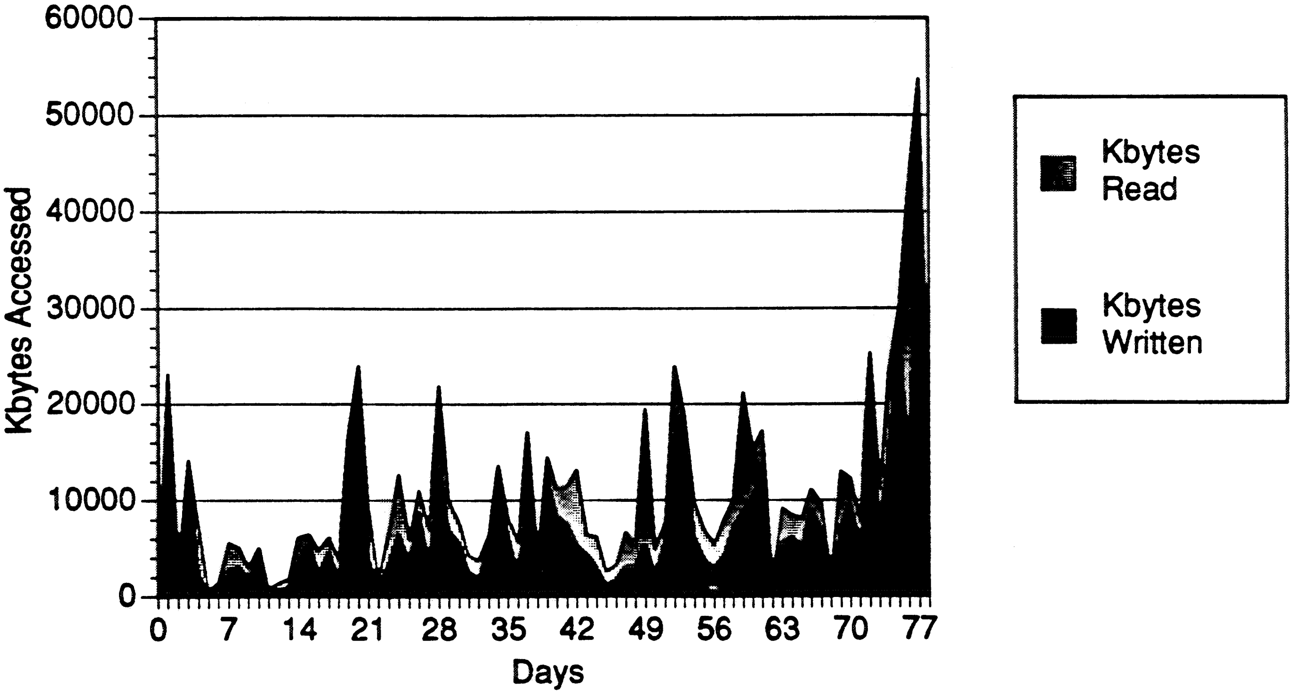


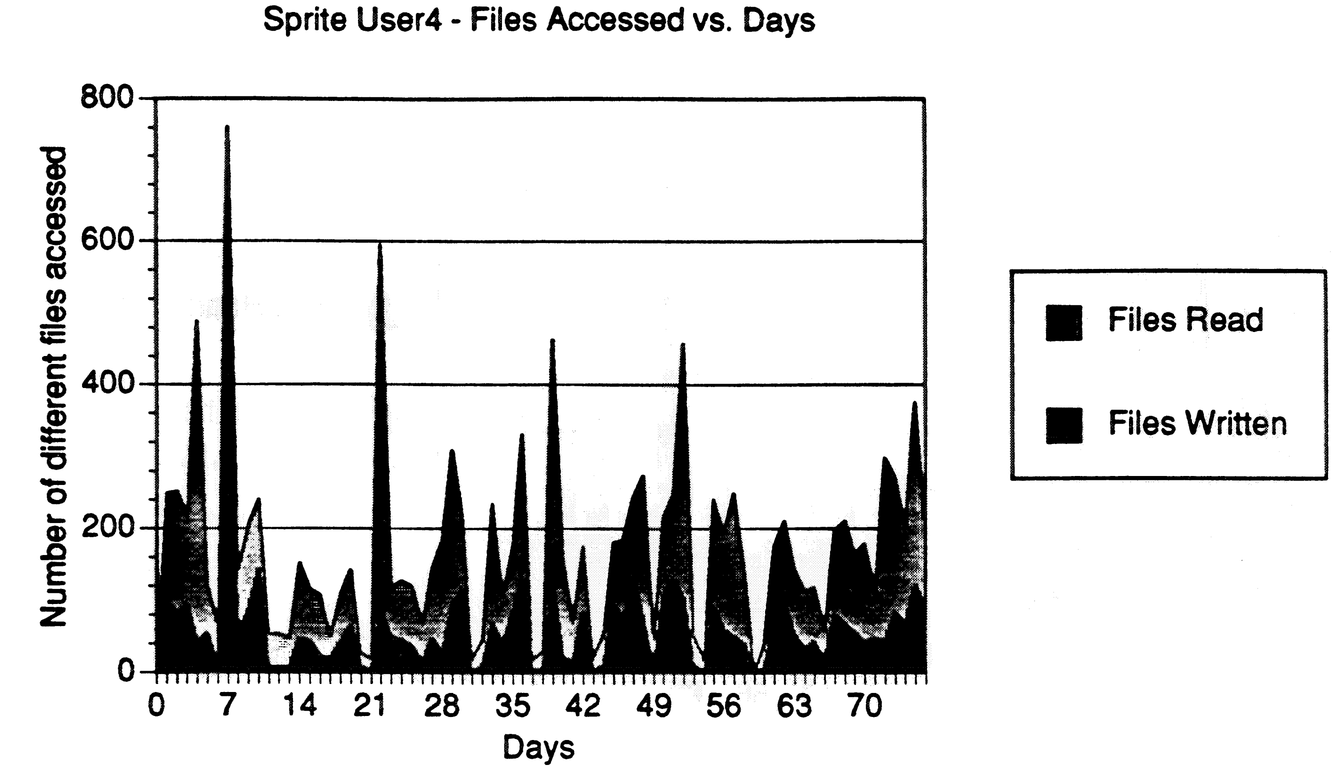
Cumulative File Size Distributions Weighted by Access (log-log)

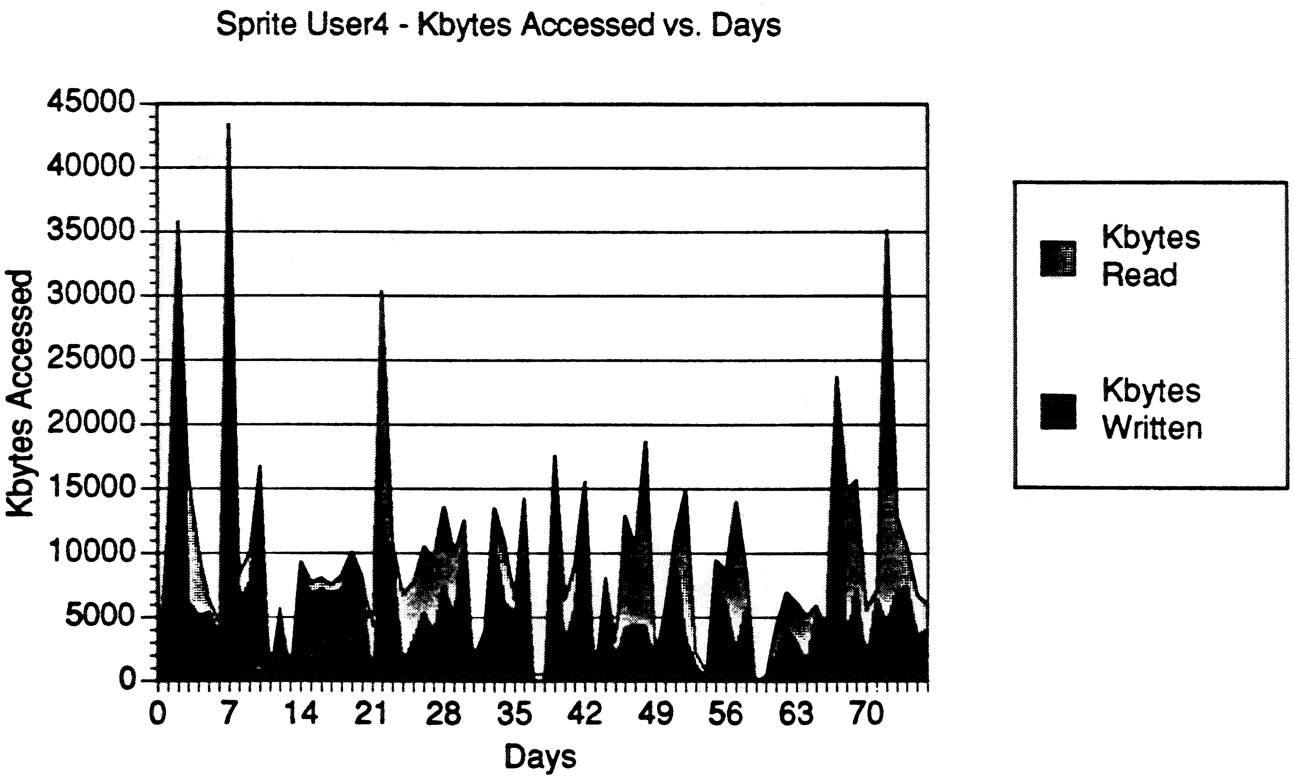
Sprite User1 - Files Accessed vs. Days



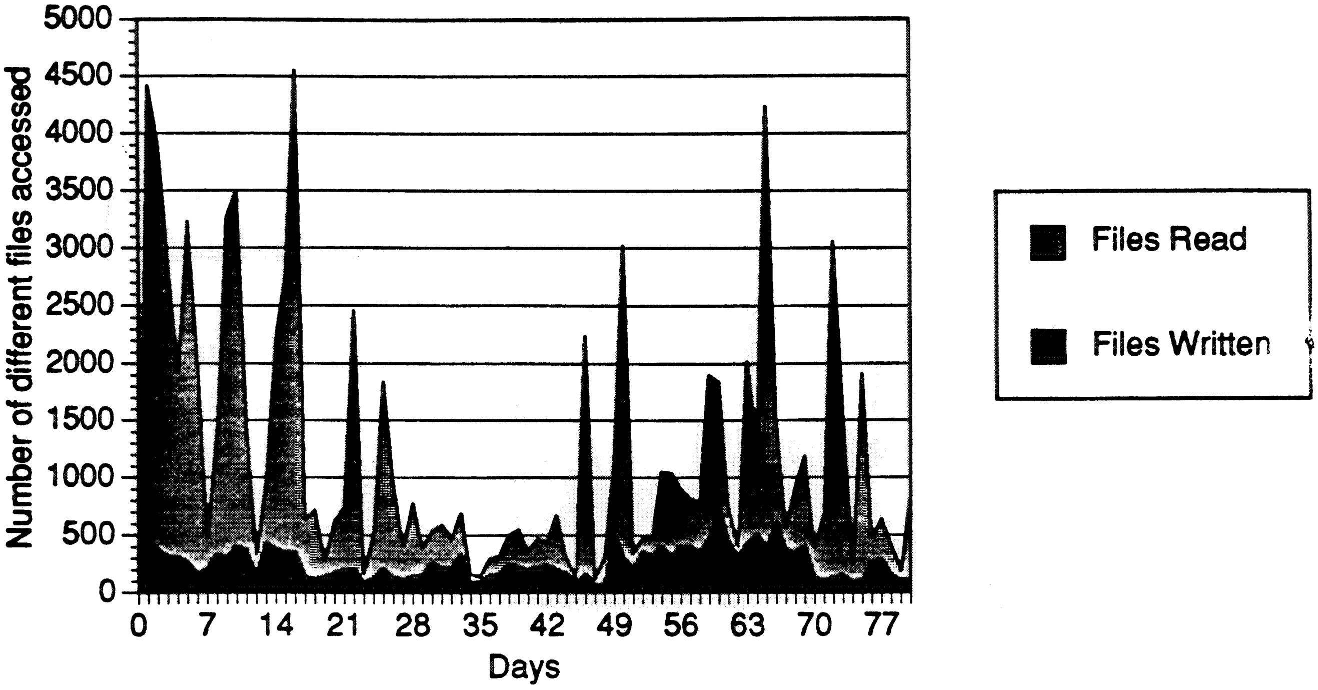
Sprite User1 - Kbytes Accessed vs. Days



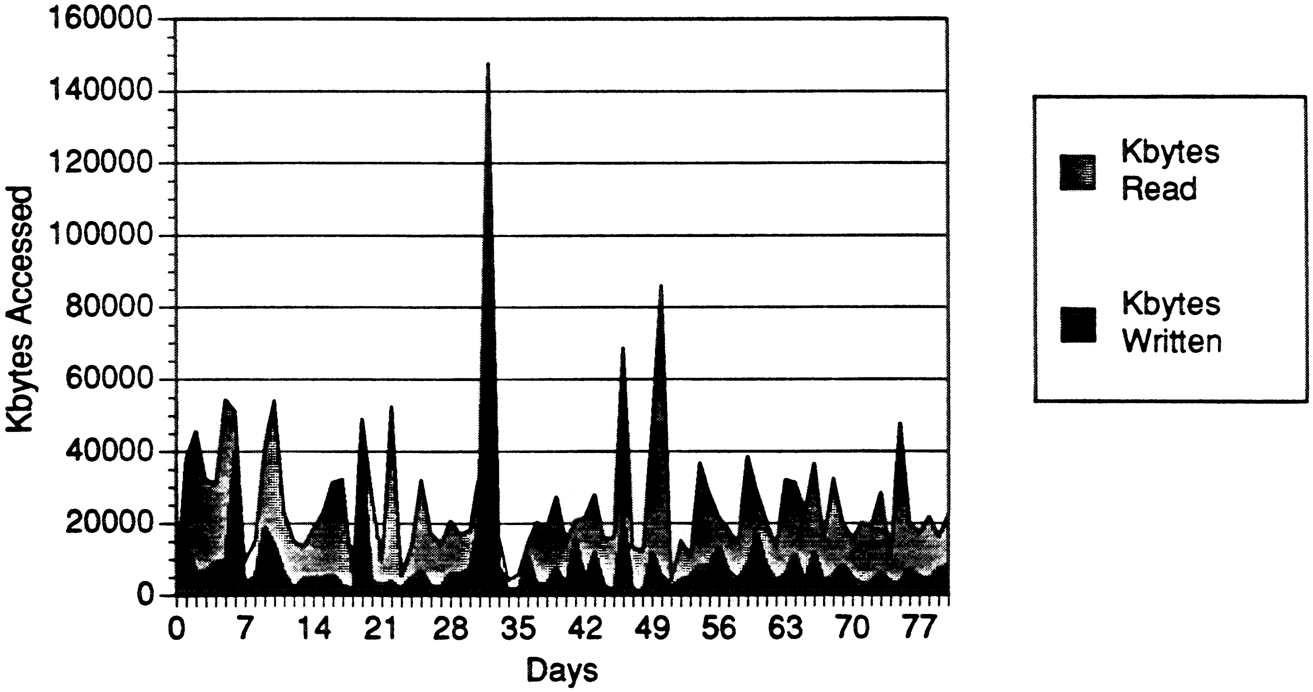

Sprite User4 - Files Accessed vs. Days
Number of different files accessed
800
600
400
200
0
0
7
14
21
28
35
42
49
56
63
70
Days
Files Read
Files Written

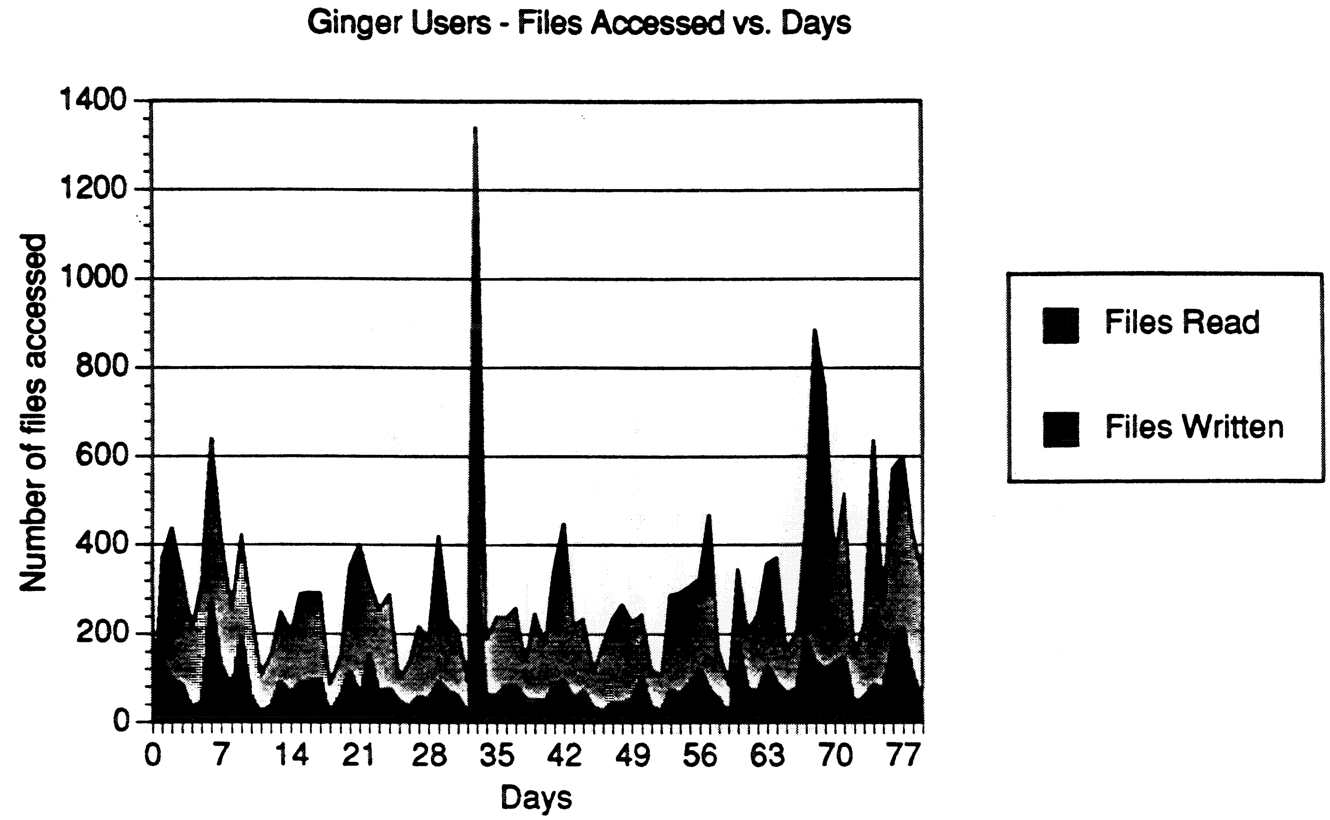

Sprite User4 - Kbytes Accessed vs. Days
Kbytes Accessed
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
0
7
14
21
28
35
42
49
56
63
70
Days
Kbytes Read
Kbytes Written

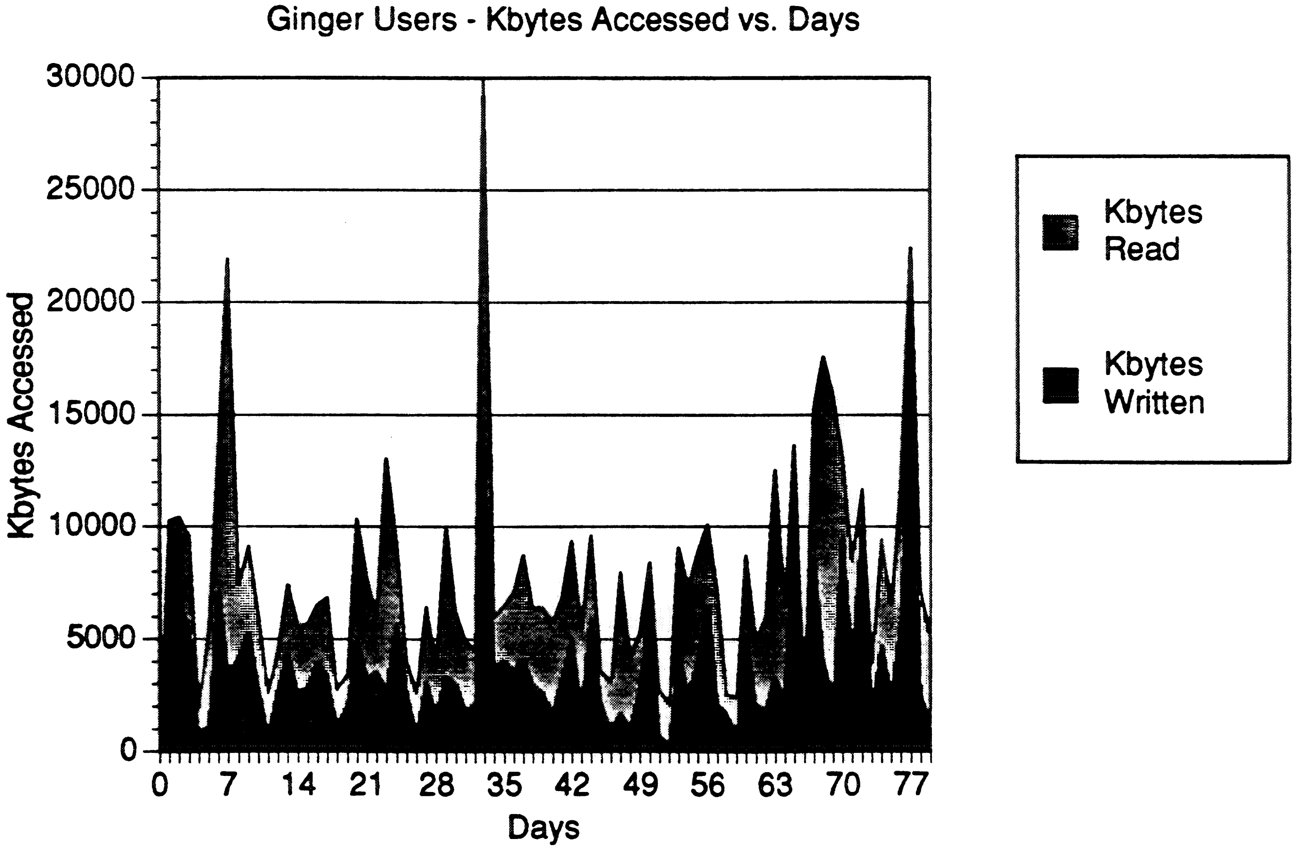
Sprite User6 - Files Accessed vs. Days

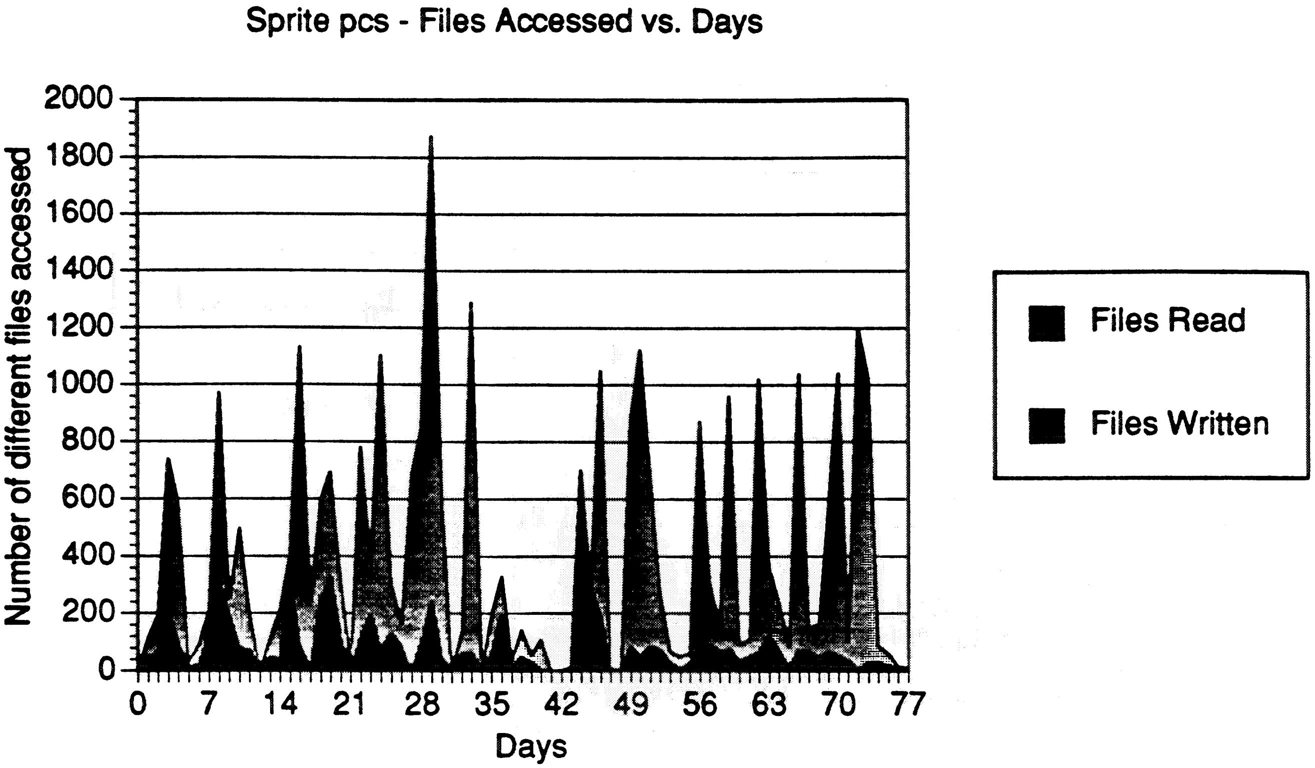


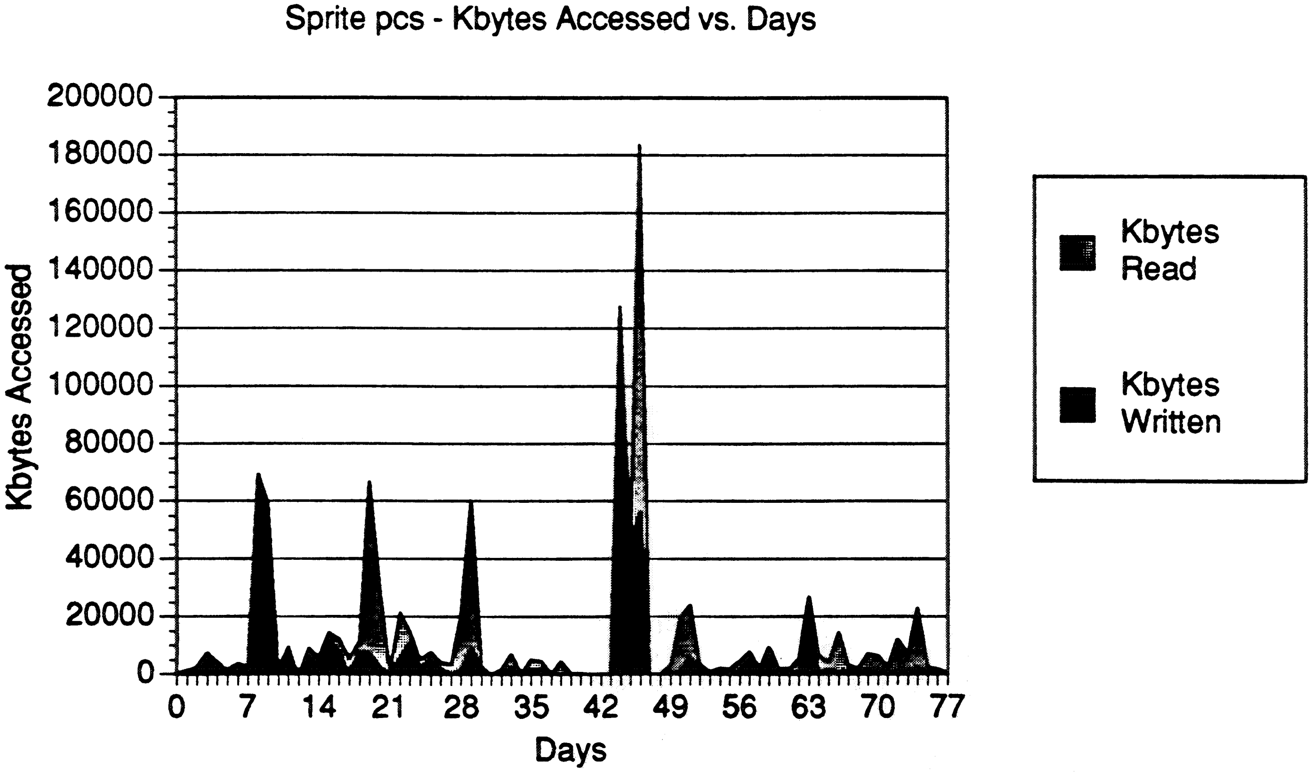
Sprite User6 - Kbytes Accessed vs. Days



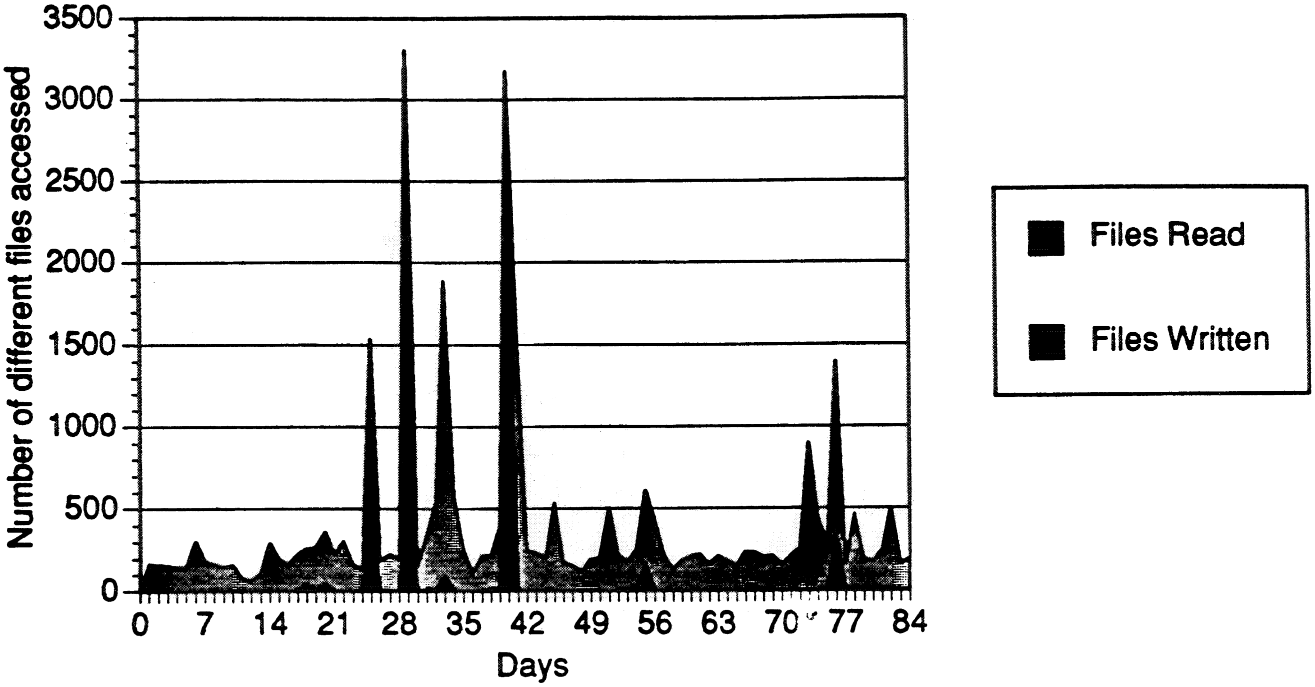

Ginger Users - Files Accessed vs. Days
Number of files accessed
0
200
400
600
800
1000
1200
1400
Days
0
7
14
21
28
35
42
49
56
63
70
77
Files Read
Files Written


Ginger Users - Kbytes Accessed vs. Days
Kbytes Accessed
0
5000
10000
15000
20000
25000
30000
Days
0
7
14
21
28
35
42
49
56
63
70
77
Kbytes Read
Kbytes Written

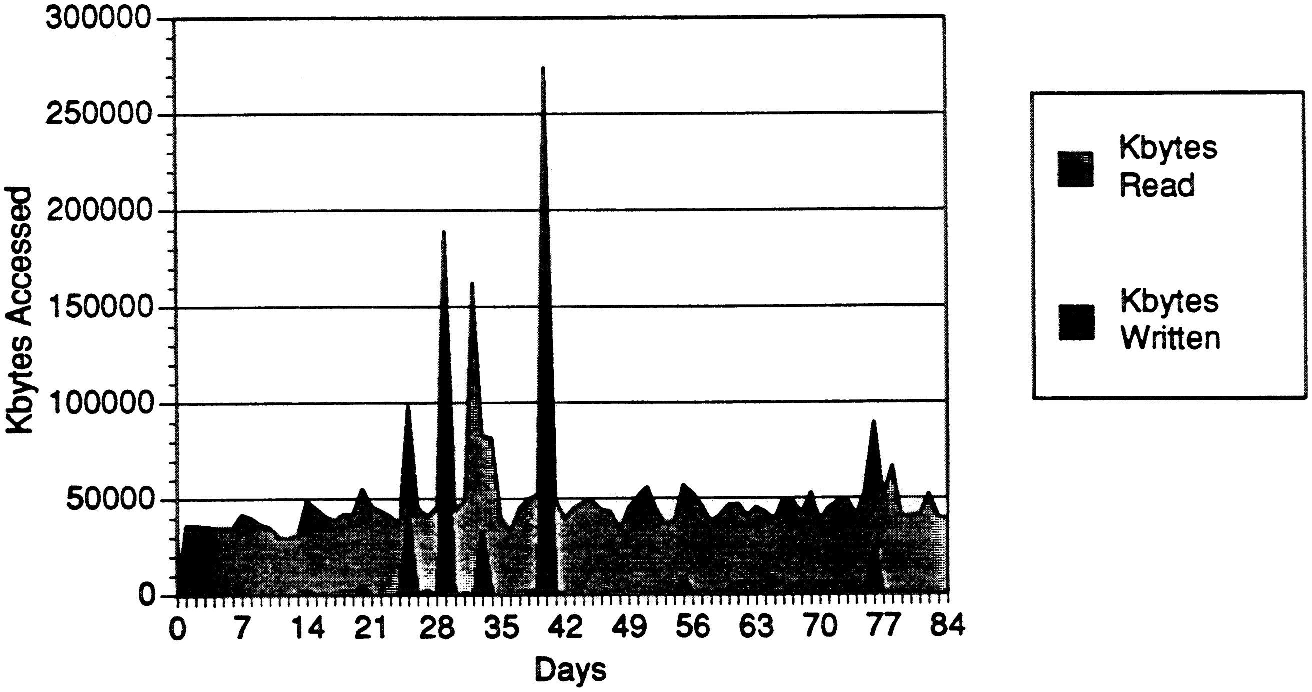

Sprite pcs - Files Accessed vs. Days
Number of different files accessed
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
7
14
21
28
35
42
49
56
63
70
77
Days
Files Read
Files Written


Sprite pcs - Kbytes Accessed vs. Days
Kbytes Accessed
200000
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
0
7
14
21
28
35
42
49
56
63
70
77
Days
Kbytes Read
Kbytes Written

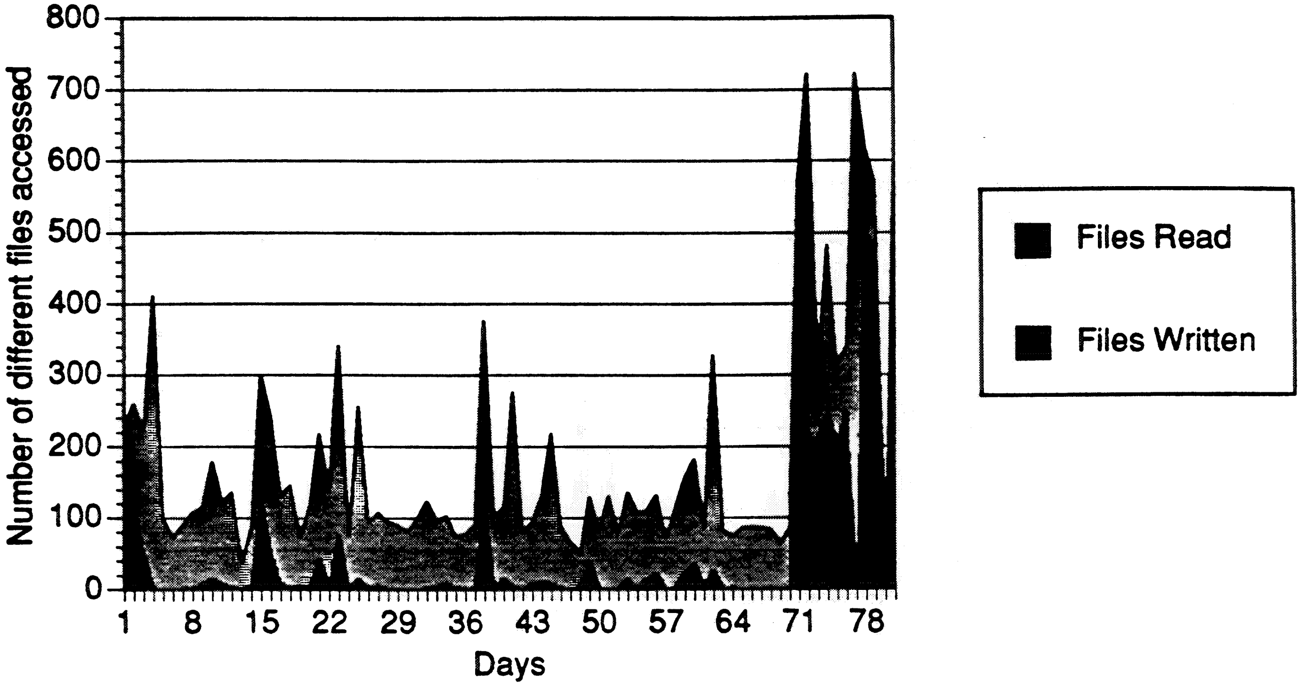
Sprite X11 - Files Accessed vs. Days



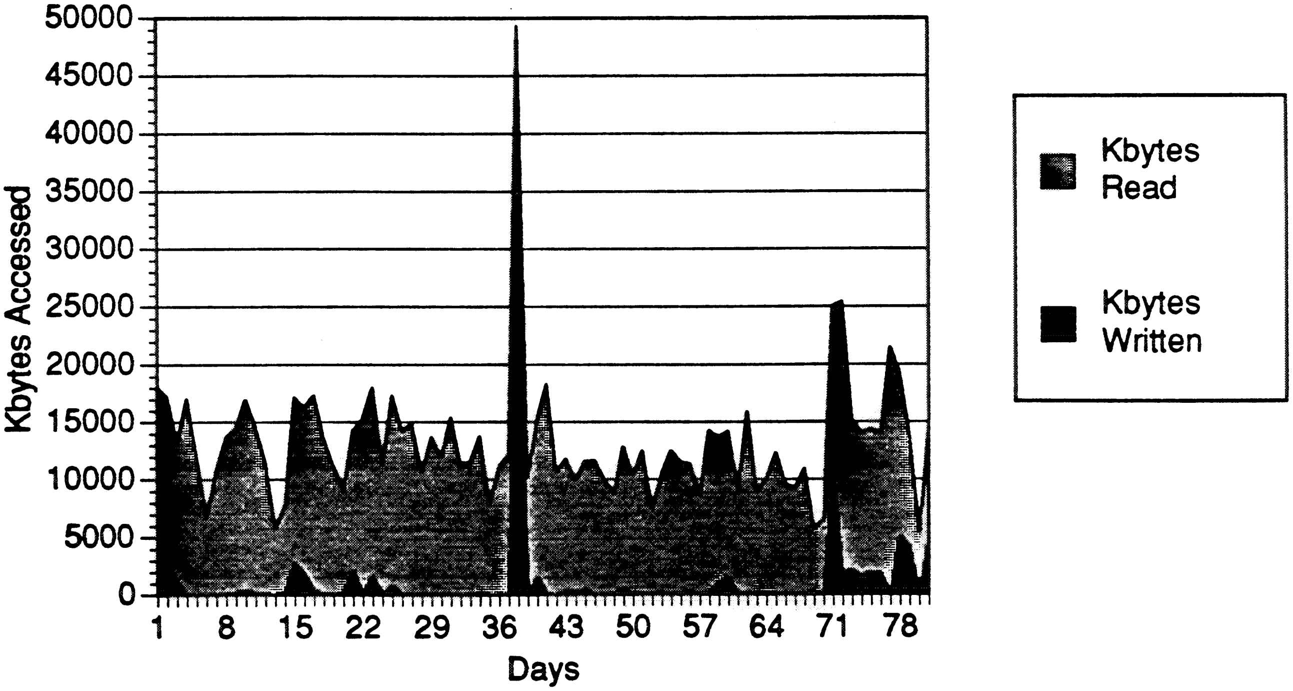
Sprite X11 - Kbytes Accessed vs. Days

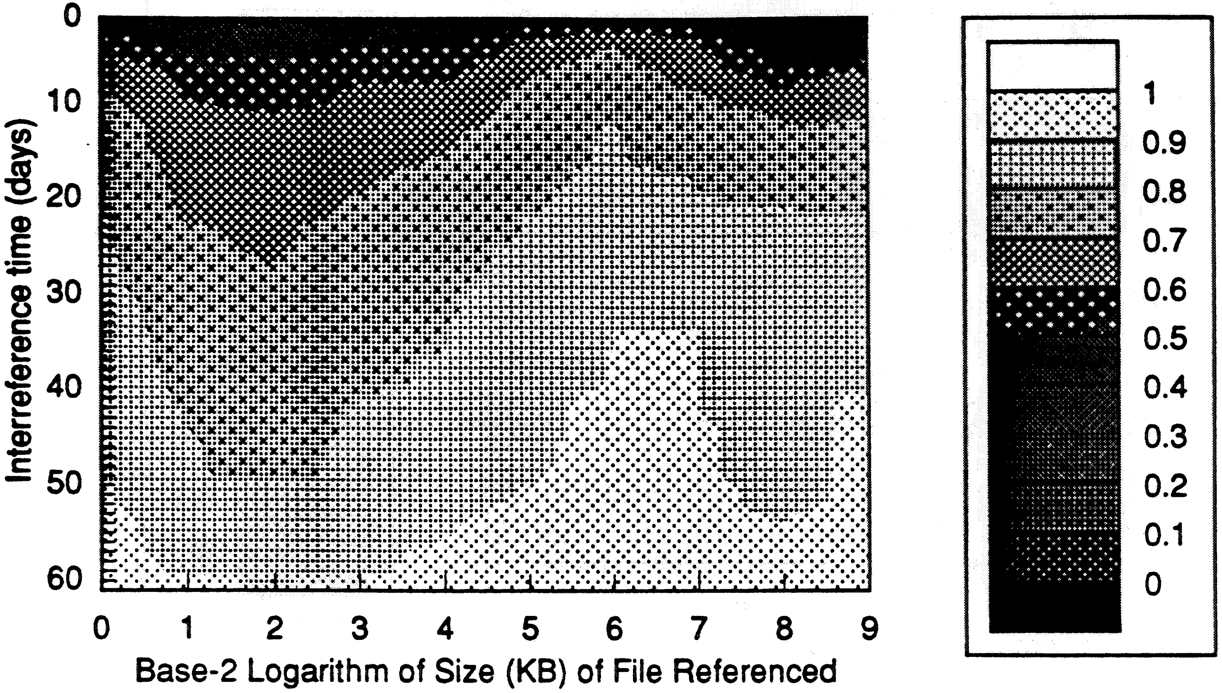


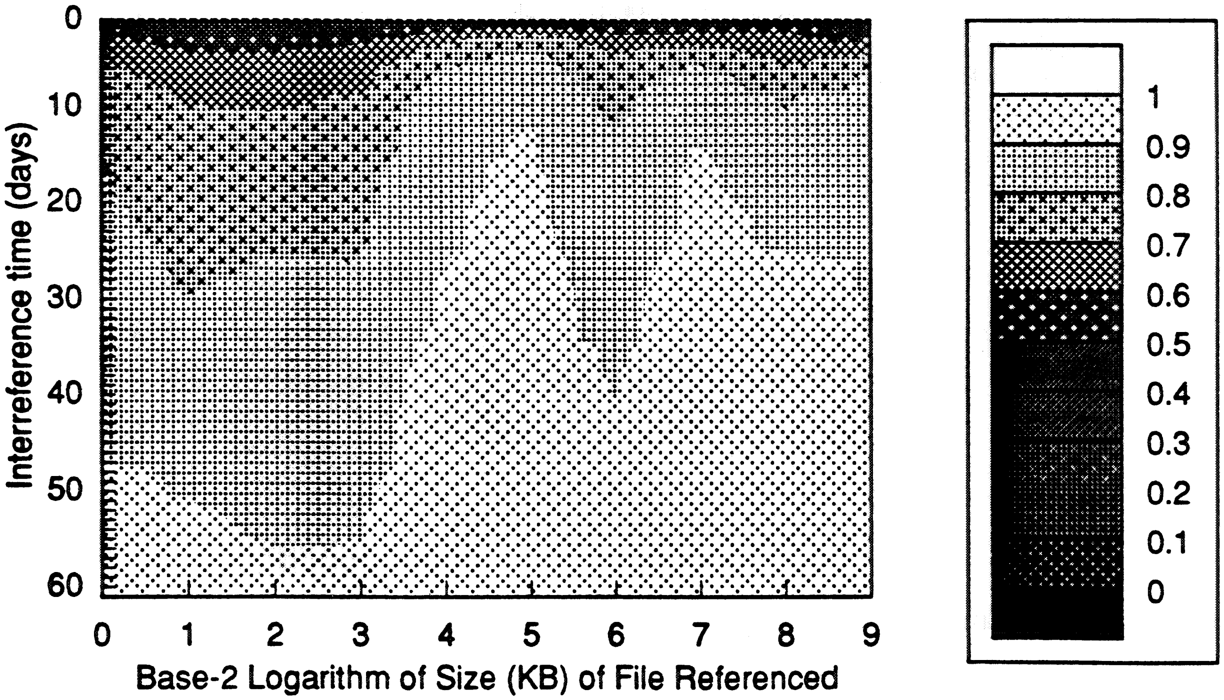
Sprite local - Files Accessed vs. Days



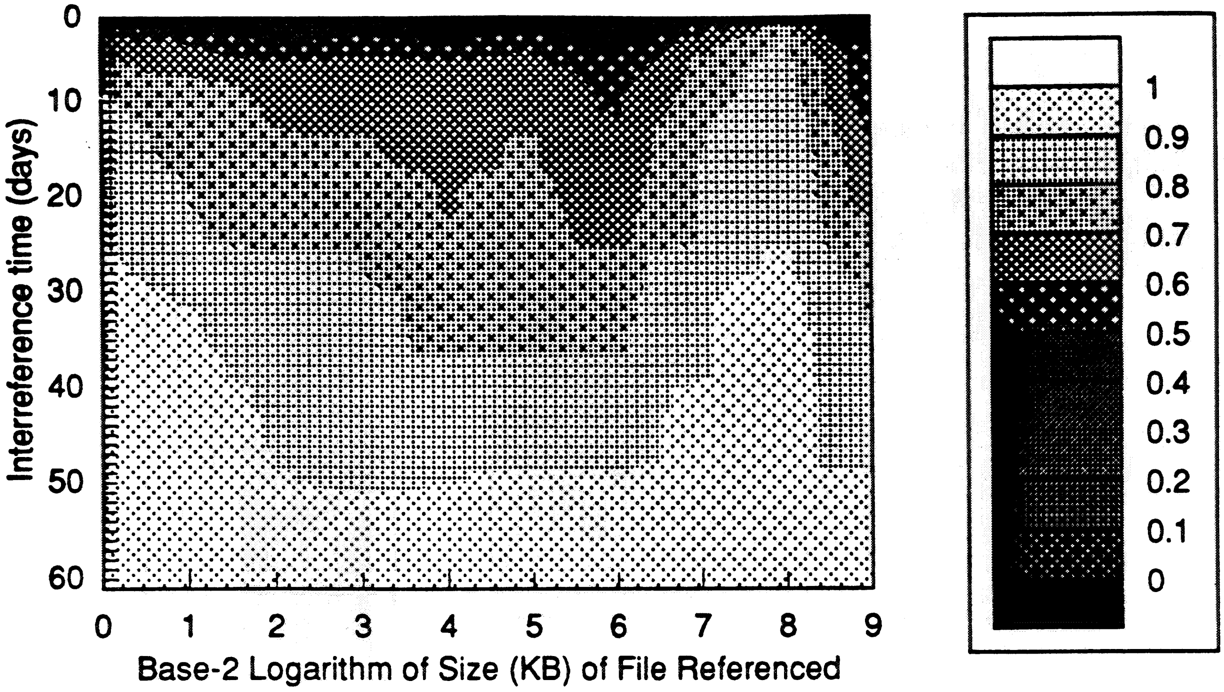
Sprite local - Kbytes Accessed vs. Days



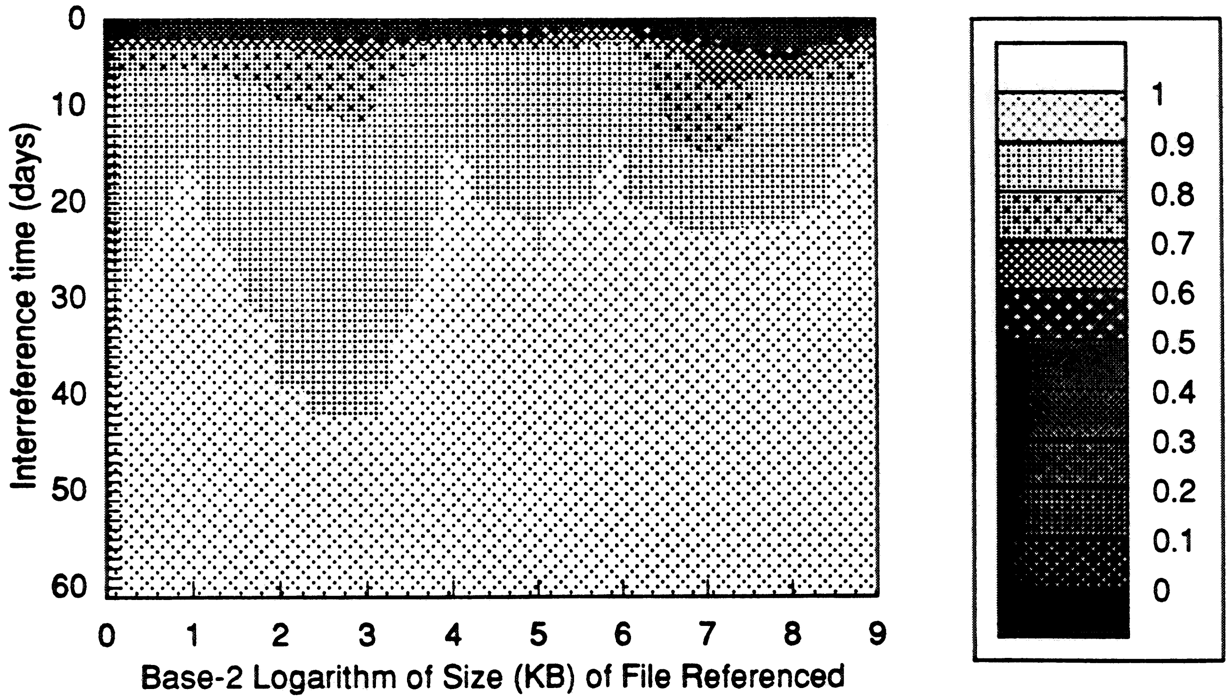

Sprite User1 – File Interreference Distributions – Reads

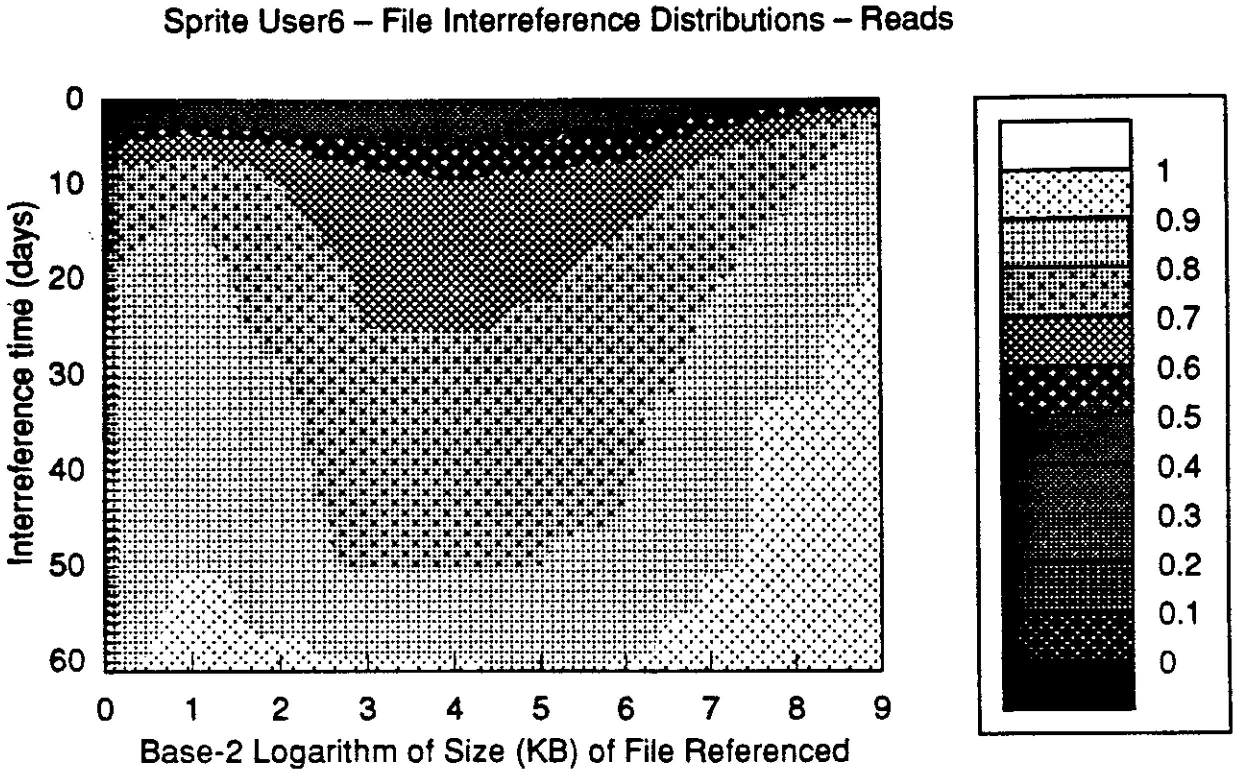

Sprite User1 – File Interreference Distributions – Writes

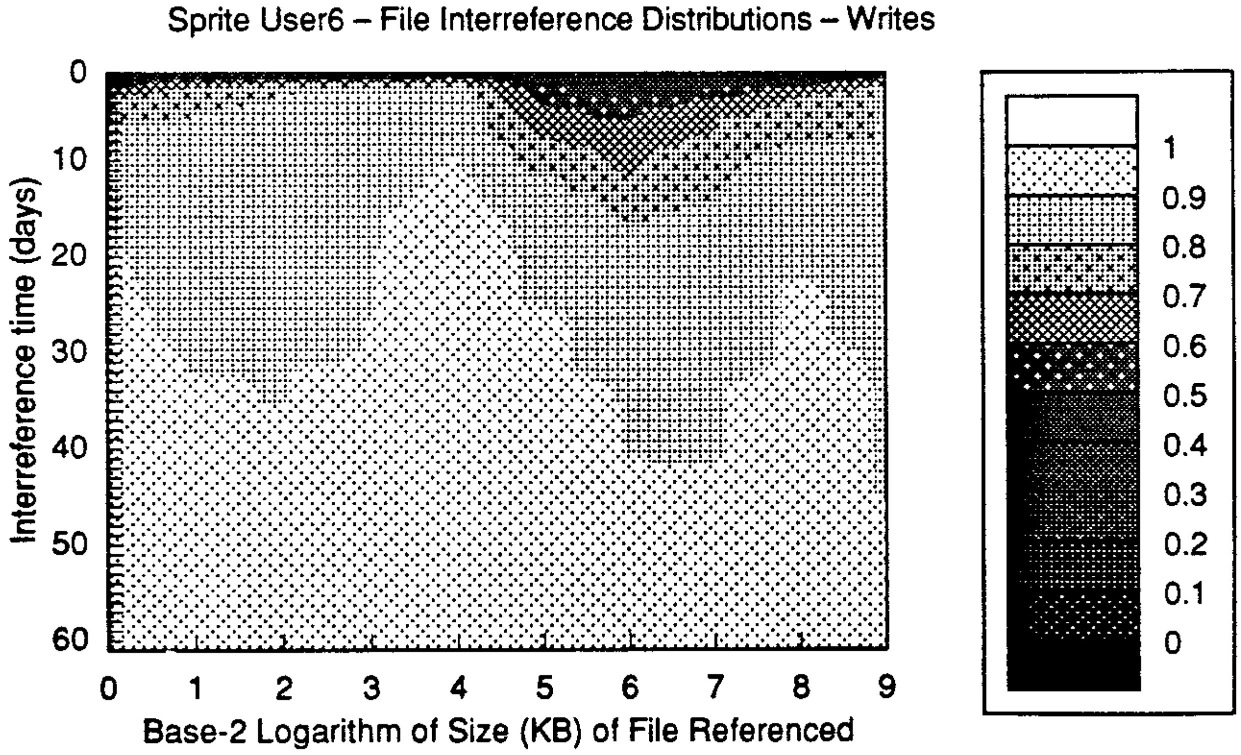
Sprite User4 – File Interreference Distributions – Reads



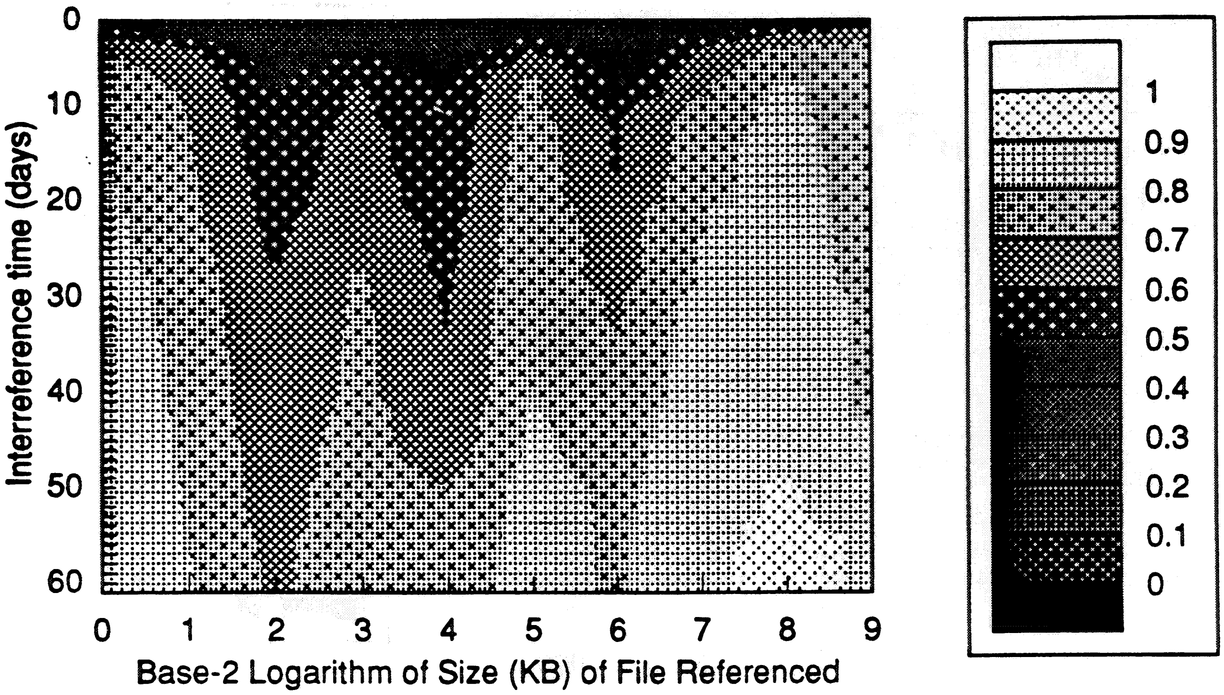
Sprite User4 – File Interreference Distributions – Writes



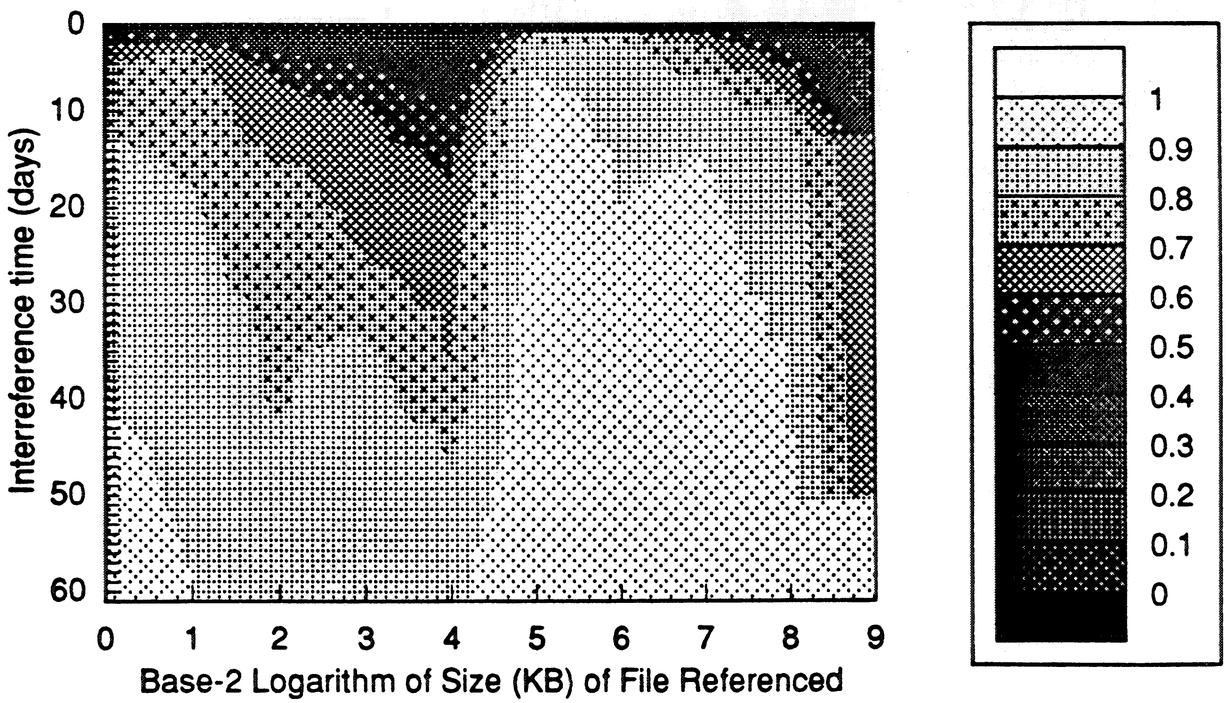

Sprite User6 – File Interreference Distributions – Reads
0
10
20
30
40
50
60
Interreference time (days)
0
1
2
3
4
5
6
7
8
9
Base-2 Logarithm of Size (KB) of File Referenced
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0


Sprite User6 – File Interreference Distributions – Writes
0
10
20
30
40
50
60
Interreference time (days)
0
1
2
3
4
5
6
7
8
9
Base-2 Logarithm of Size (KB) of File Referenced
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

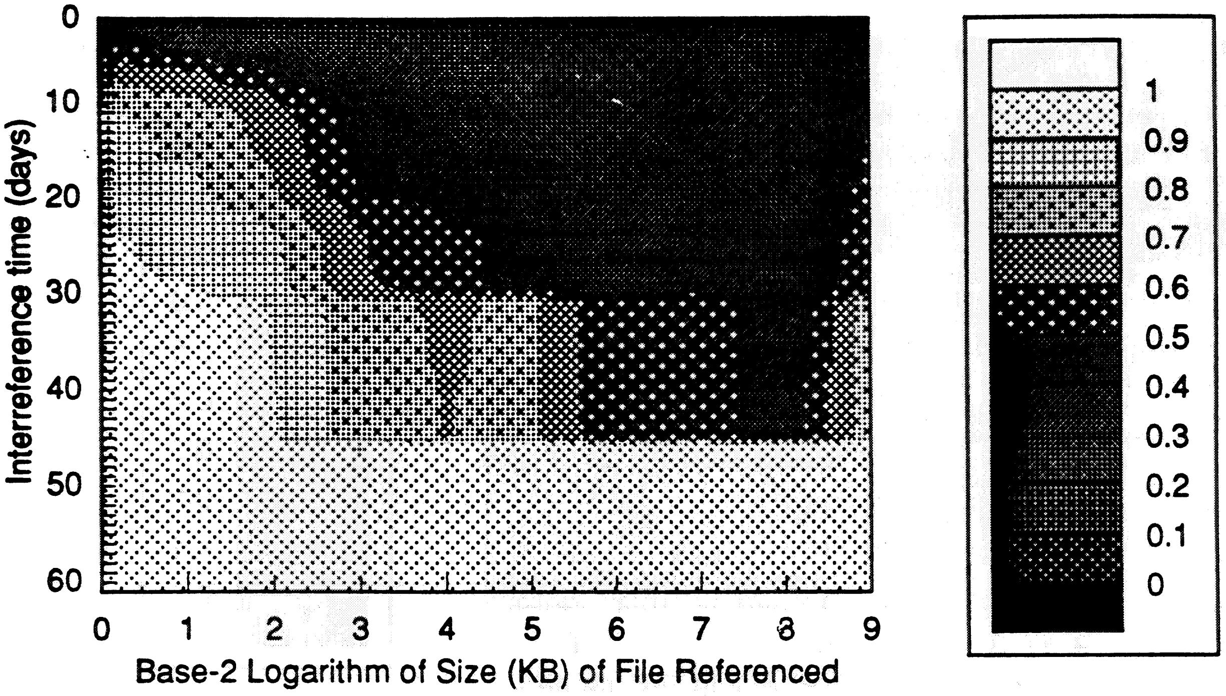
Ginger Users – File Interreference Distributions – Reads



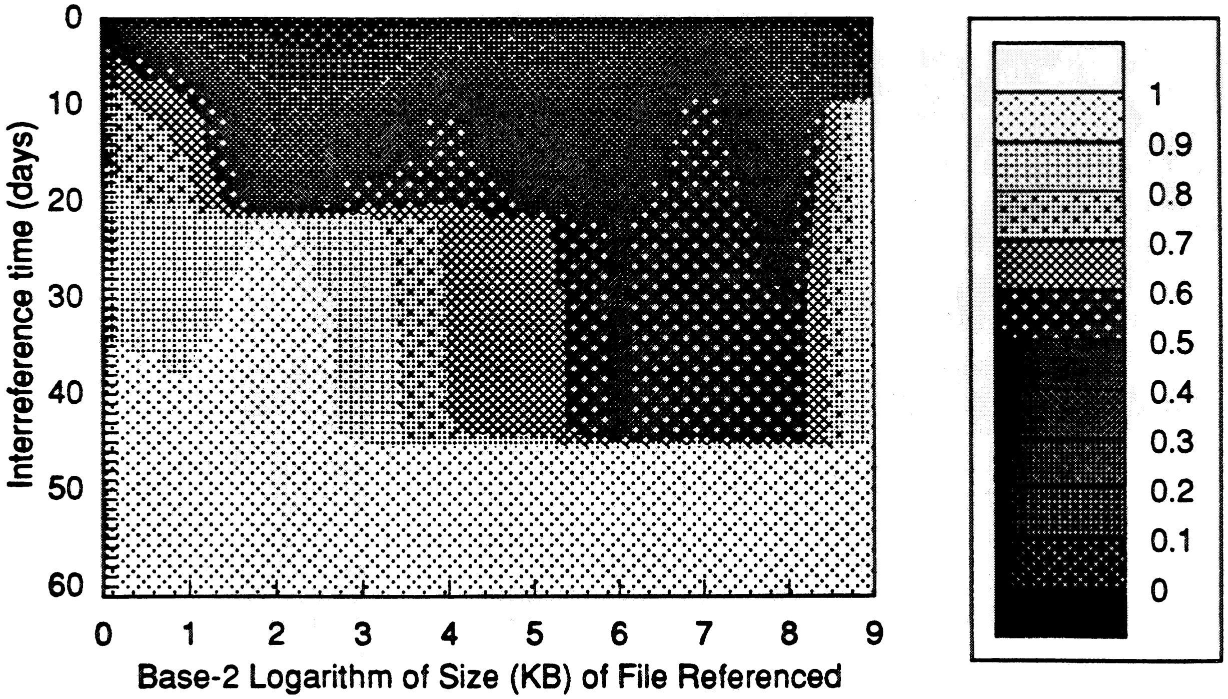
Ginger Users – File Interreference Distributions – Writes



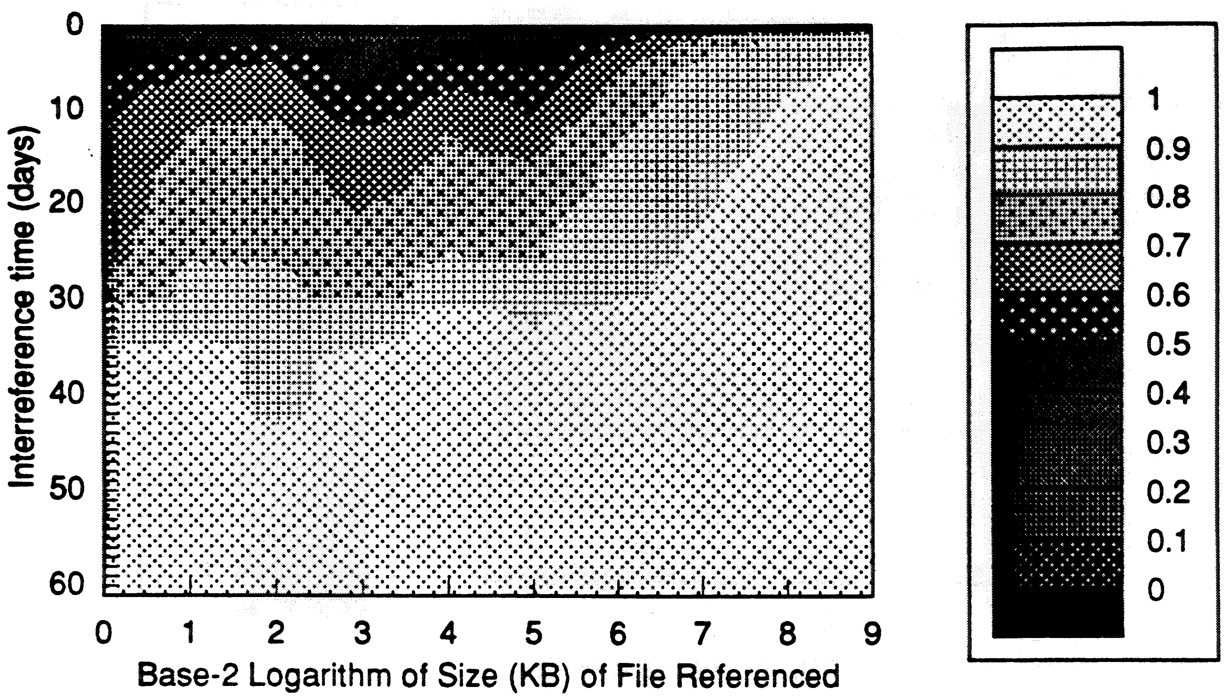
Sprite pcs – File Interreference Distributions – Reads



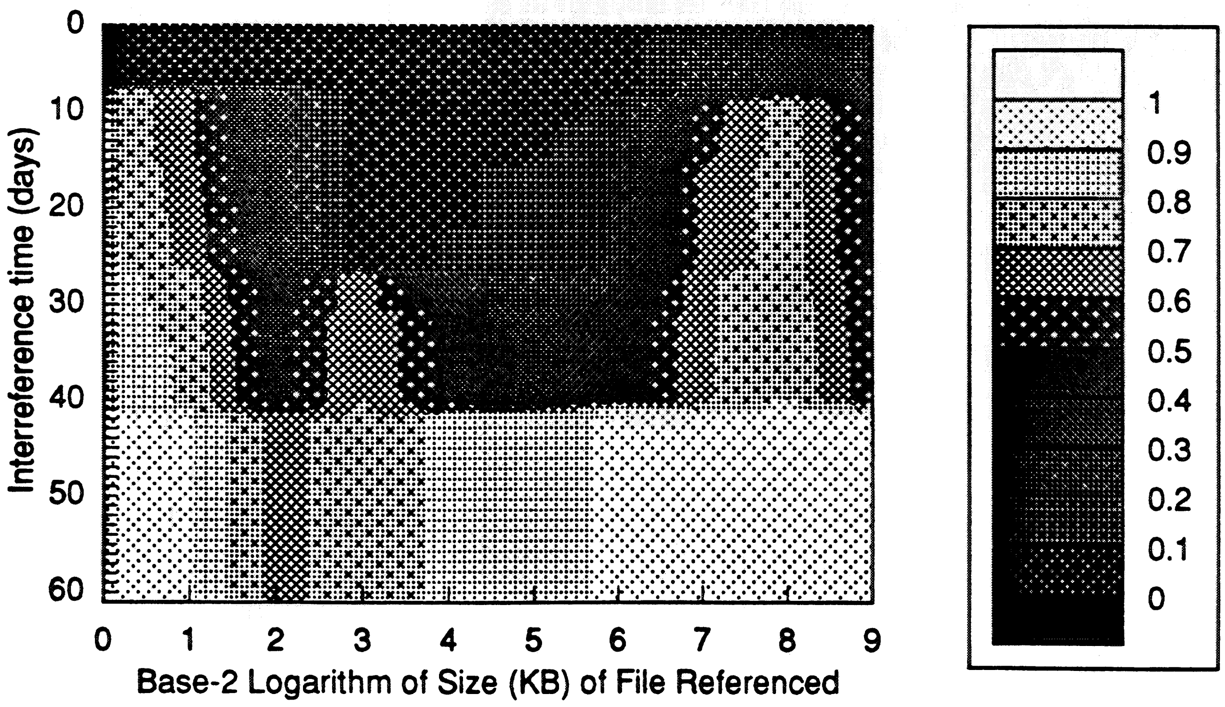
Sprite pcs – File Interreference Distributions – Writes



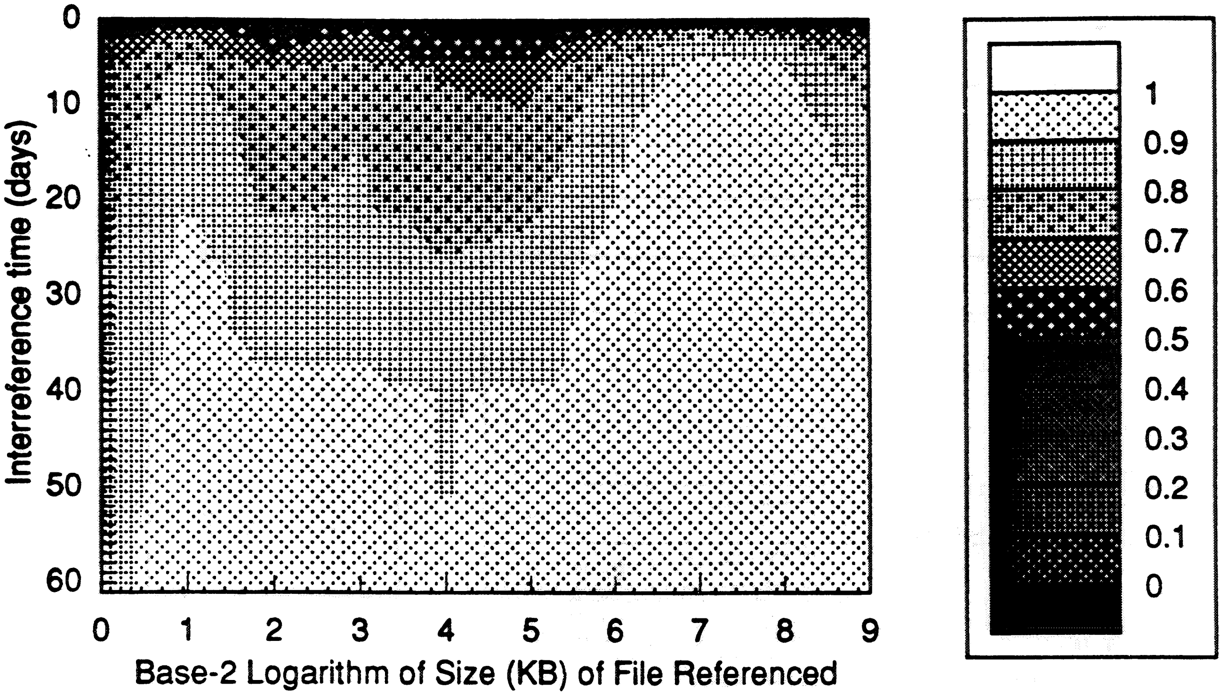
X11 – File Interreference Distributions – Reads



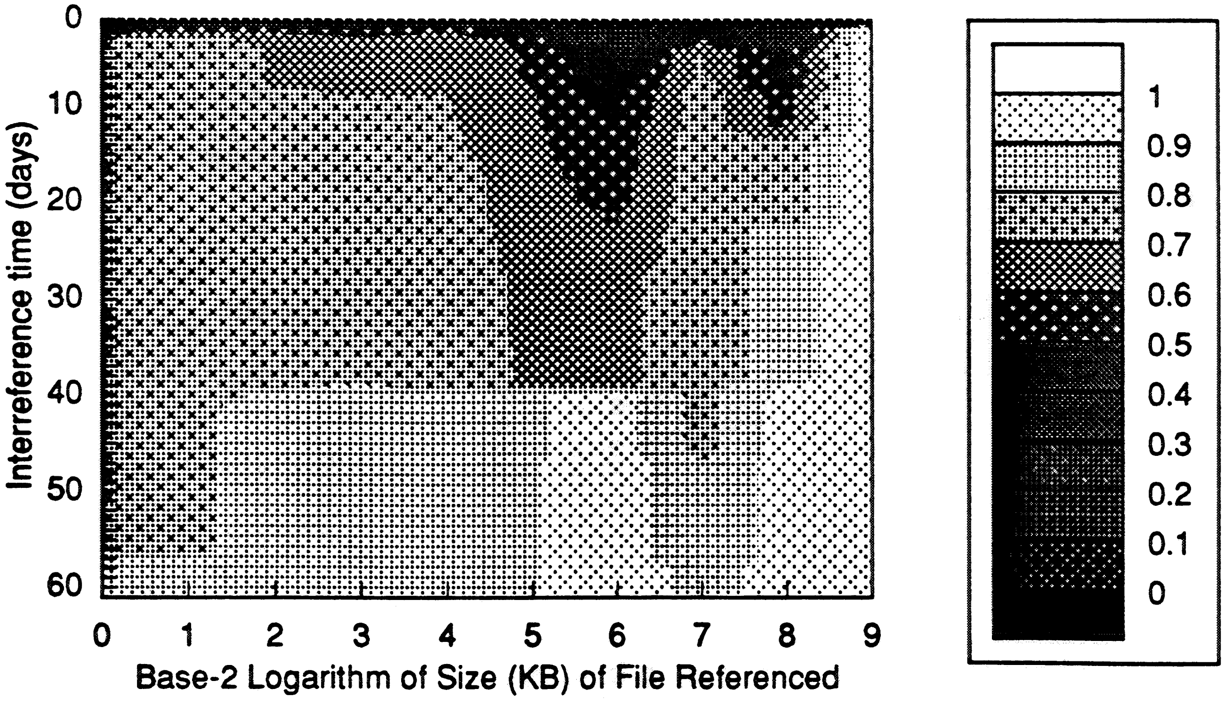
X11 – File Interreference Distributions – Writes



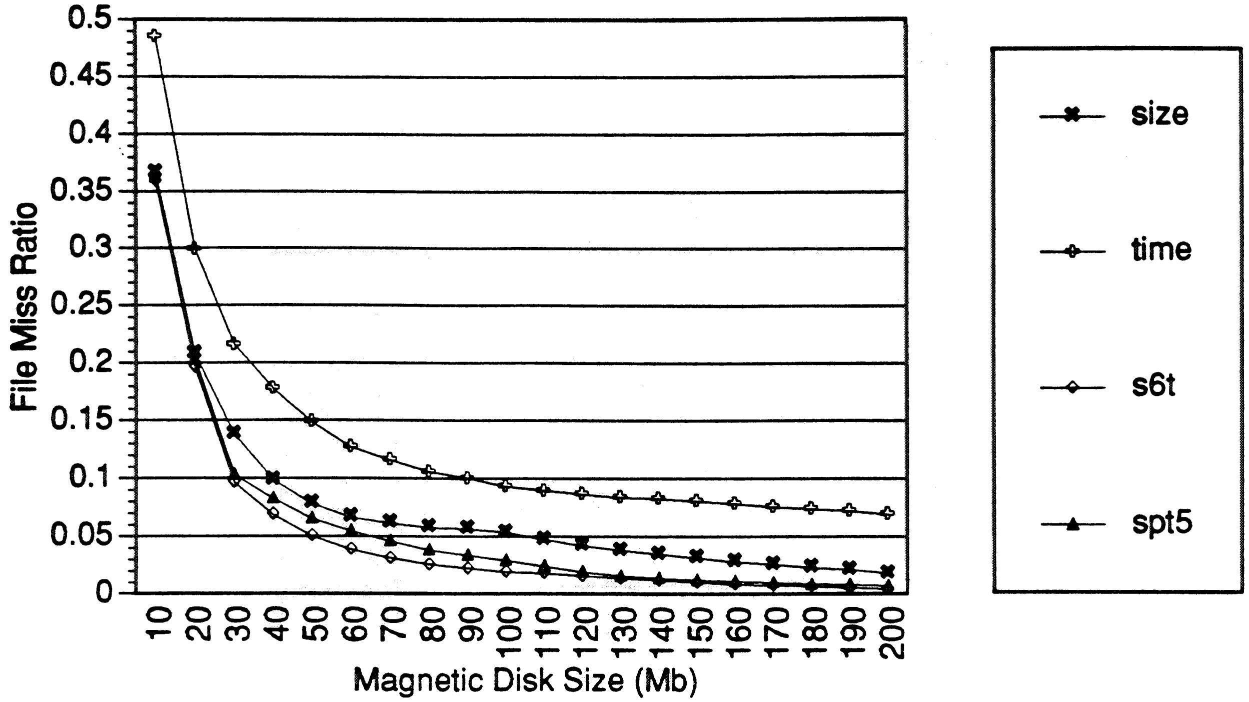
Sprite local – File Interreference Distributions – Reads



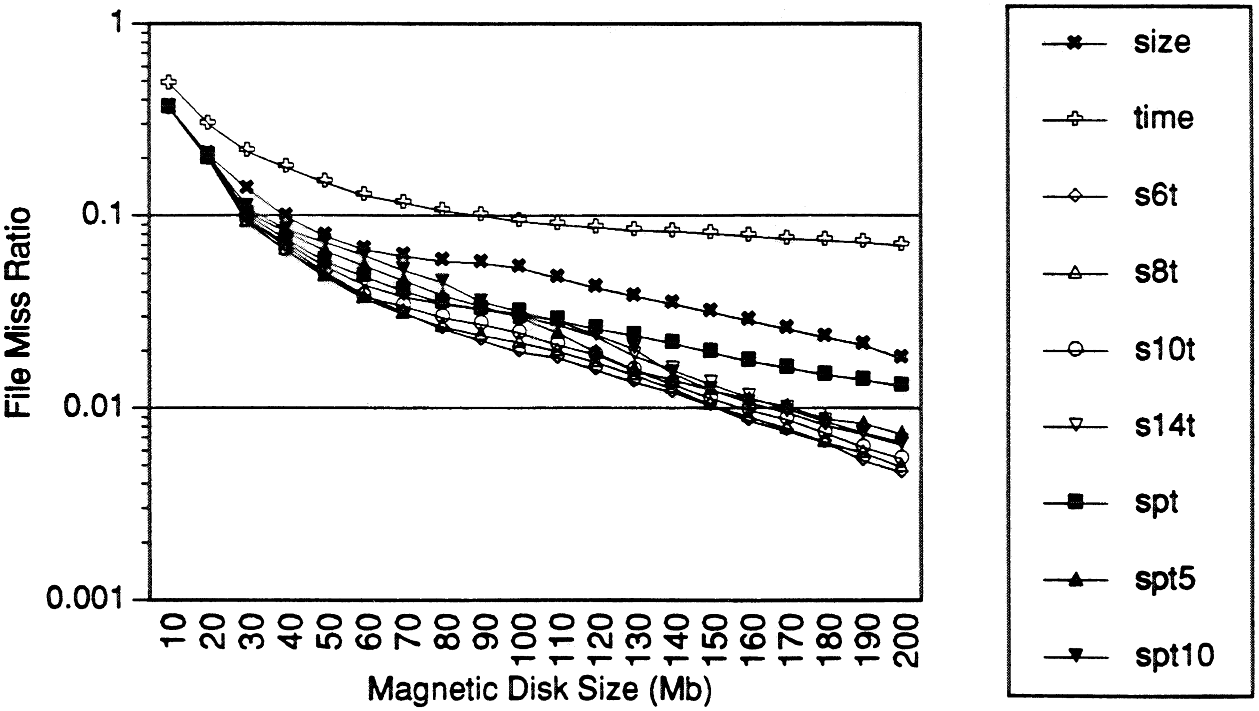
Sprite local – File Interreference Distributions – Writes



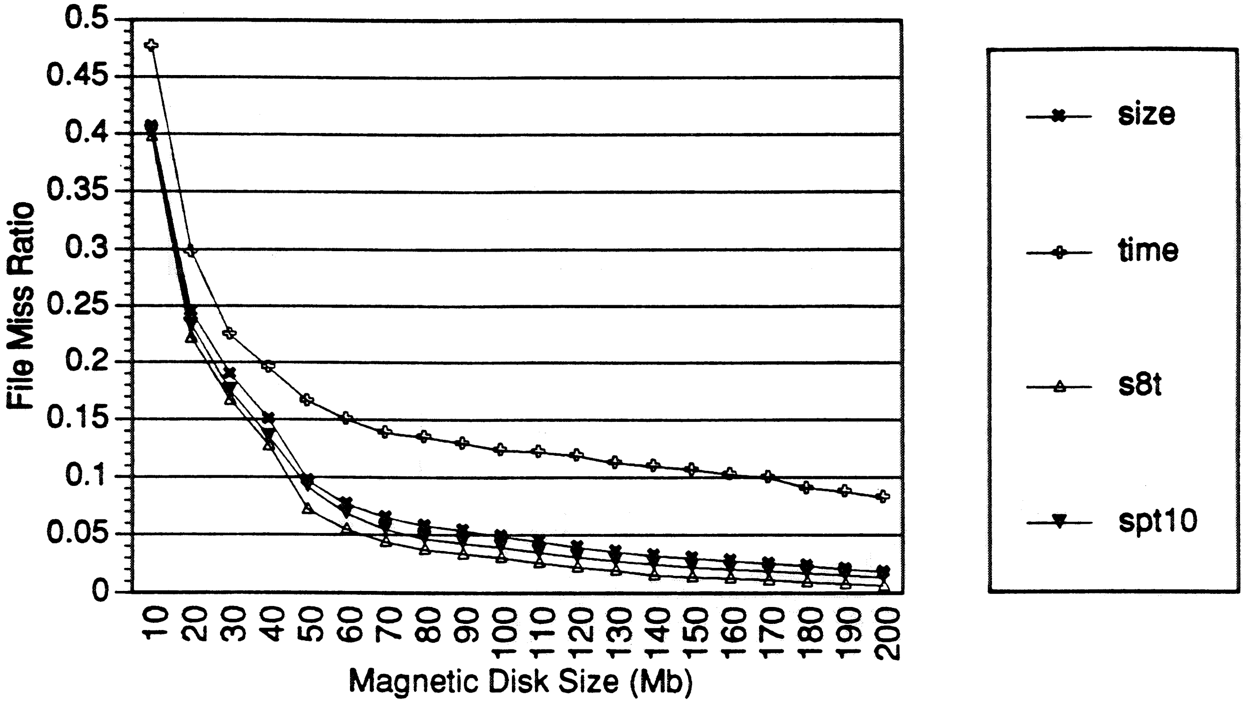
Sprite - User 1 (subset)



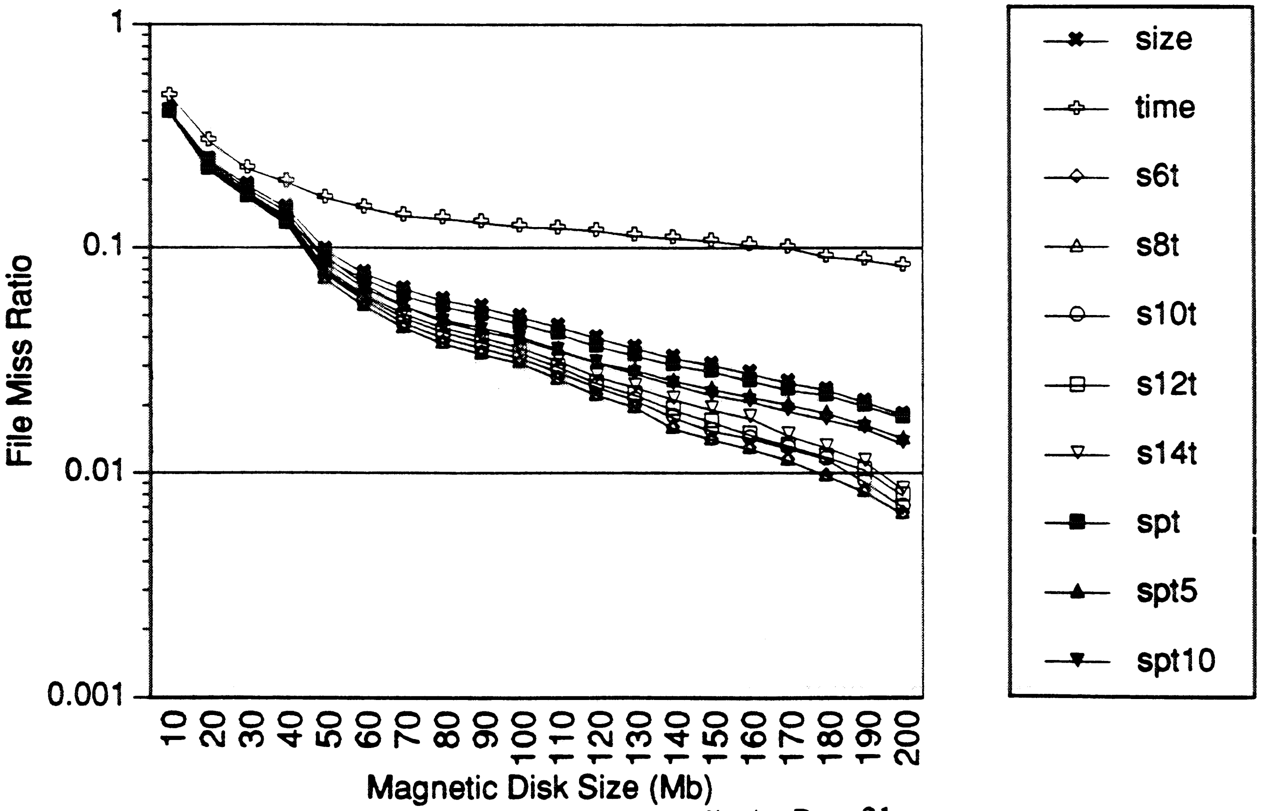
Sprite - User 1 (full set)

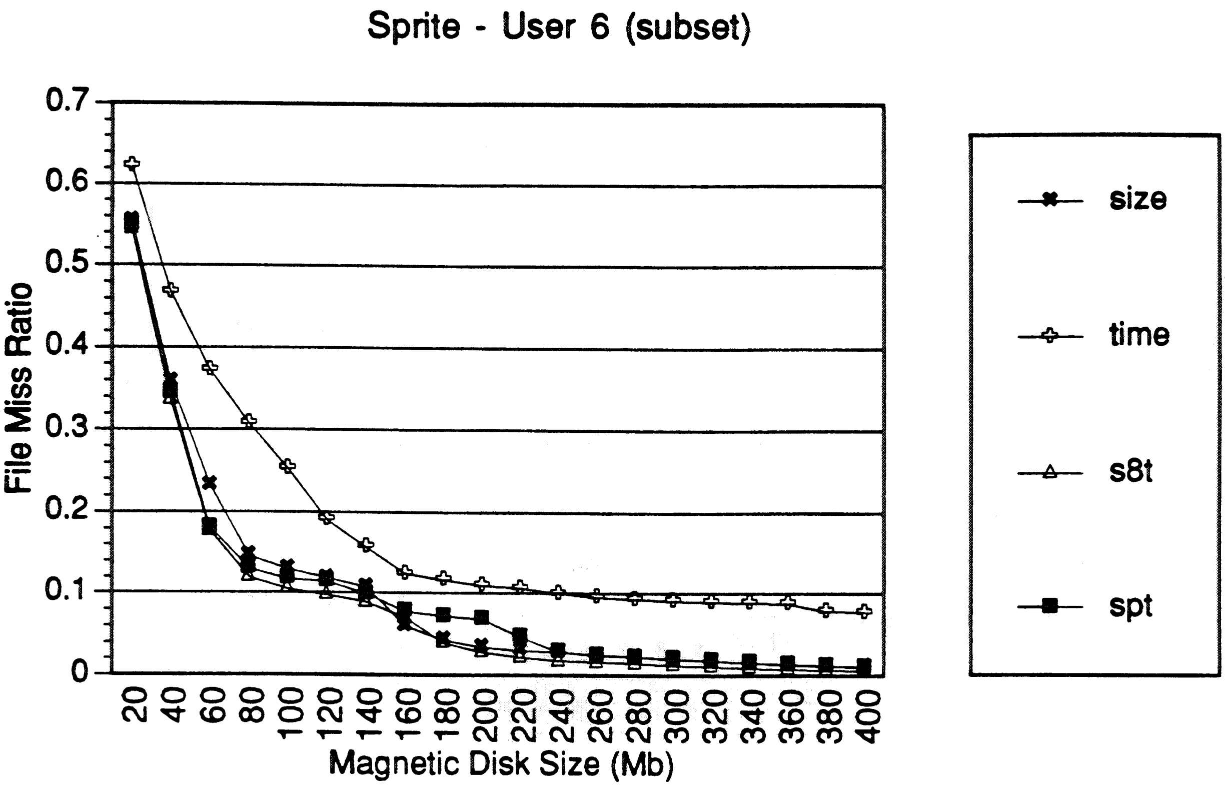


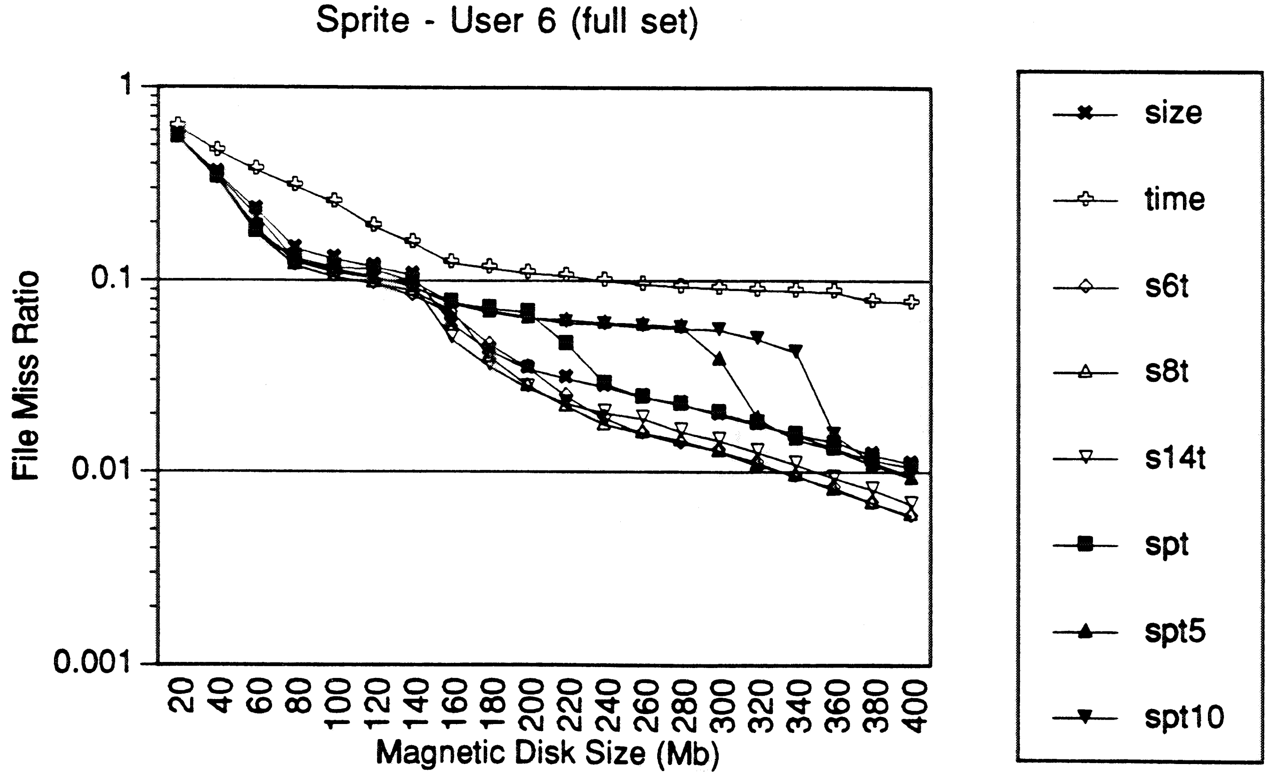
Sprite - User 4 (subset)



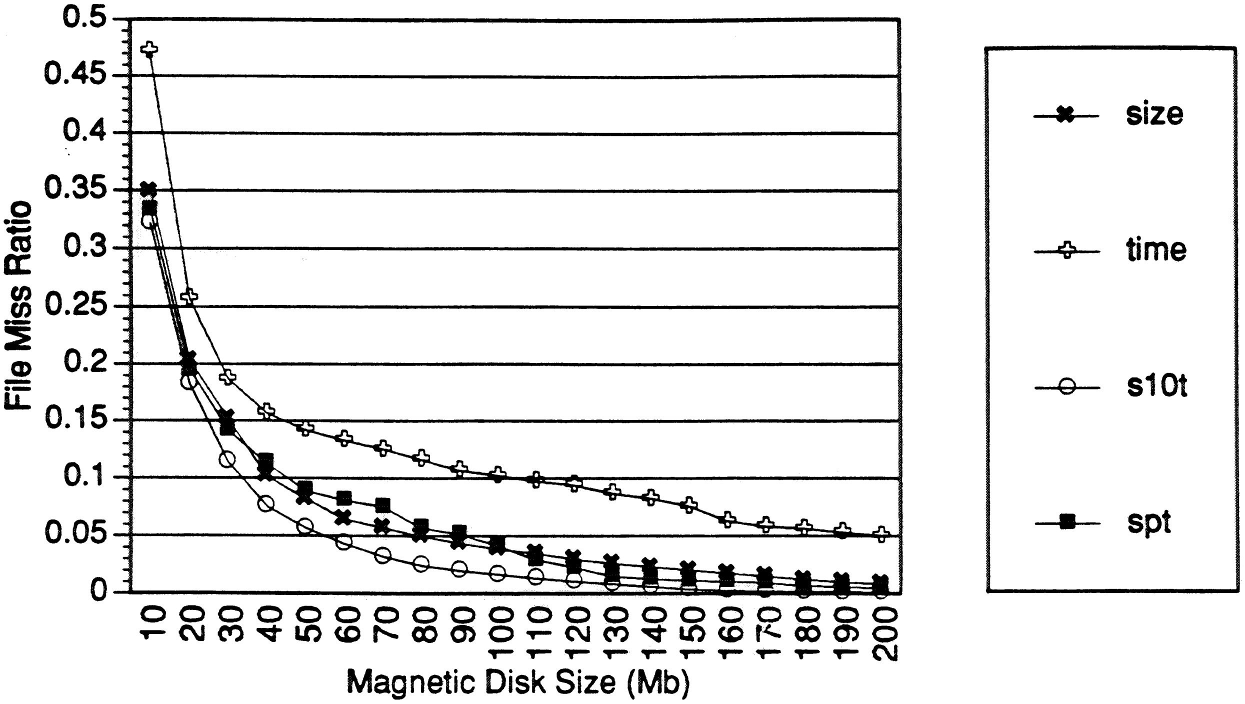
Sprite - User 4 (full set)



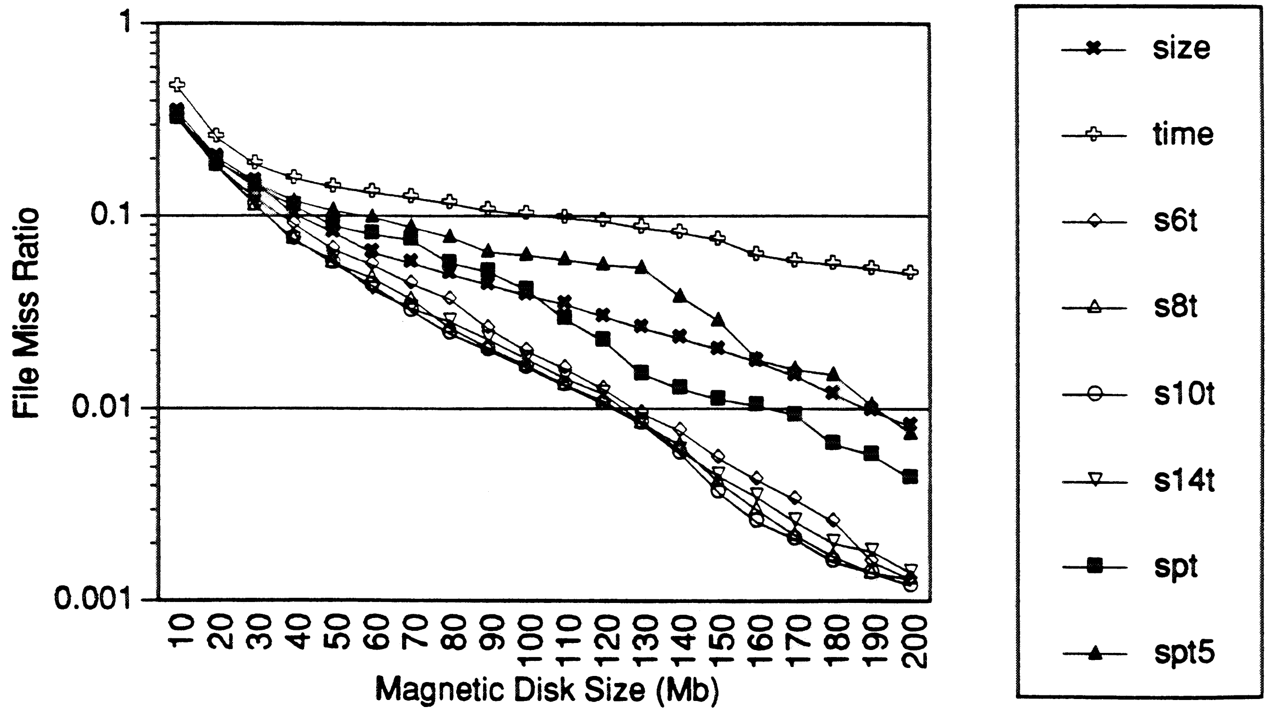

Sprite - User 6 (subset)
File Miss Ratio
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
380
400
Magnetic Disk Size (Mb)
size
time
s8t
spt


Sprite - User 6 (full set)
File Miss Ratio
1
0.1
0.01
0.001
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
380
400
Magnetic Disk Size (Mb)
size
time
s6t
s8t
s14t
spt
spt5
spt10

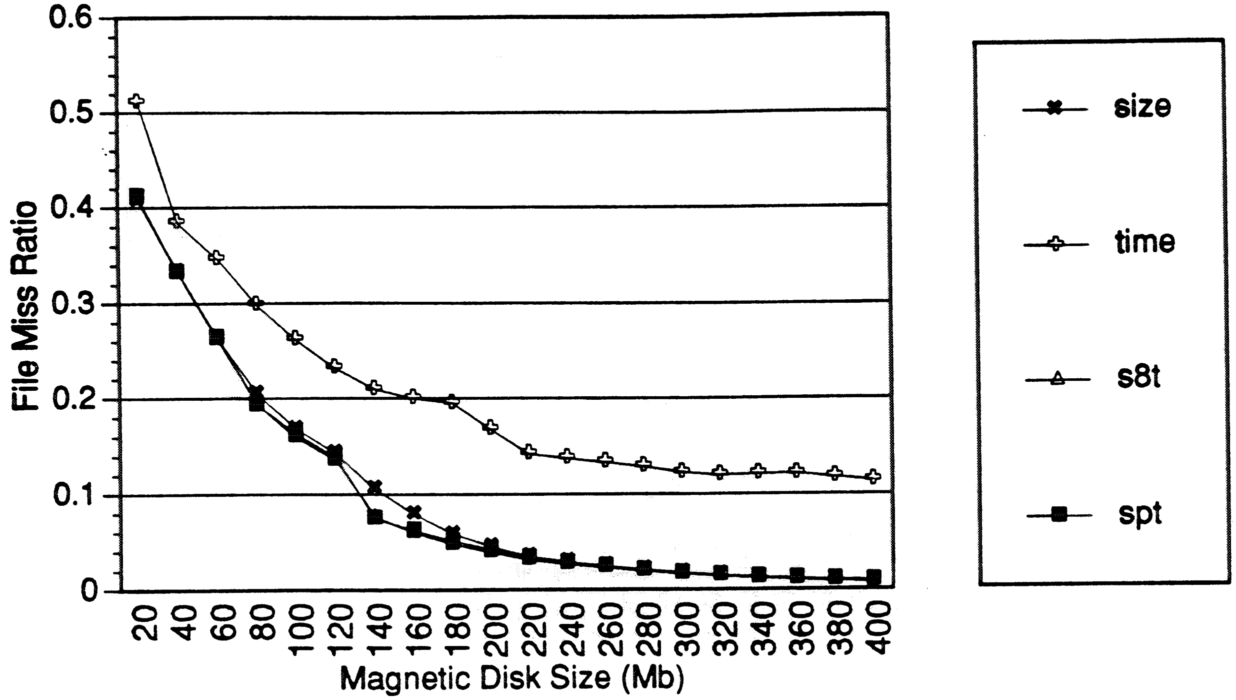
Ginger Users (subset)



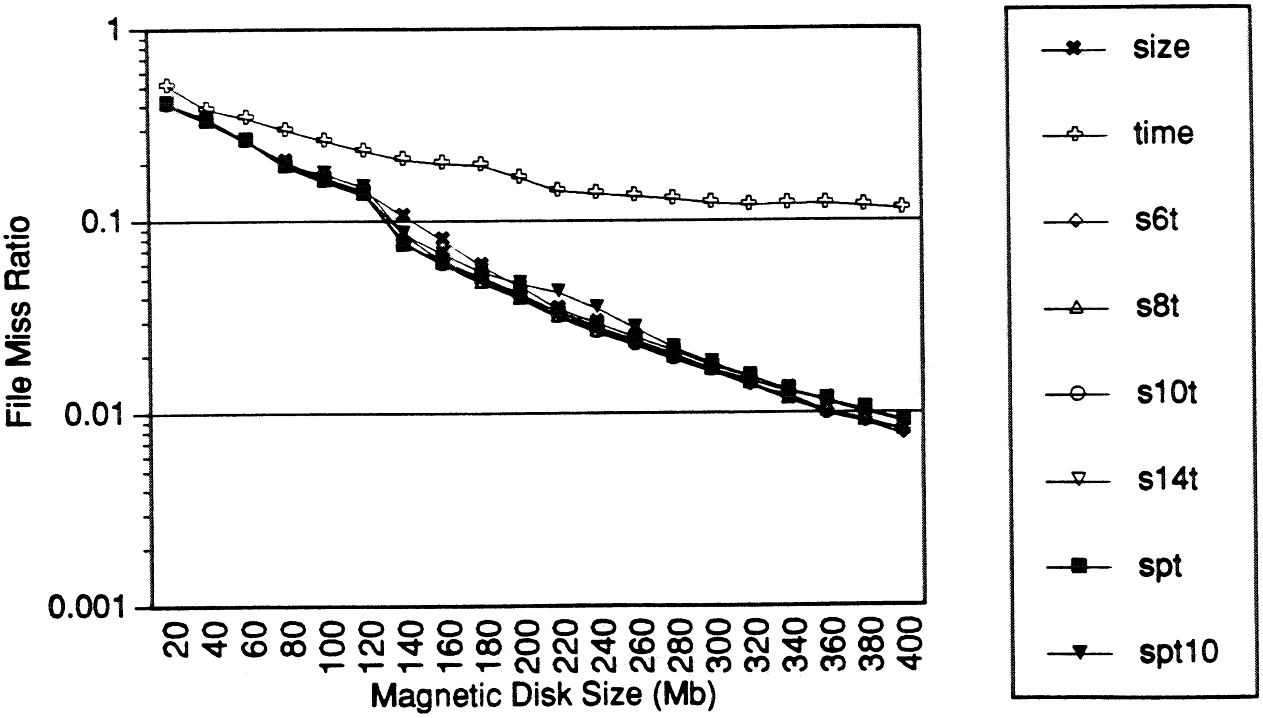
Ginger Users (full set)

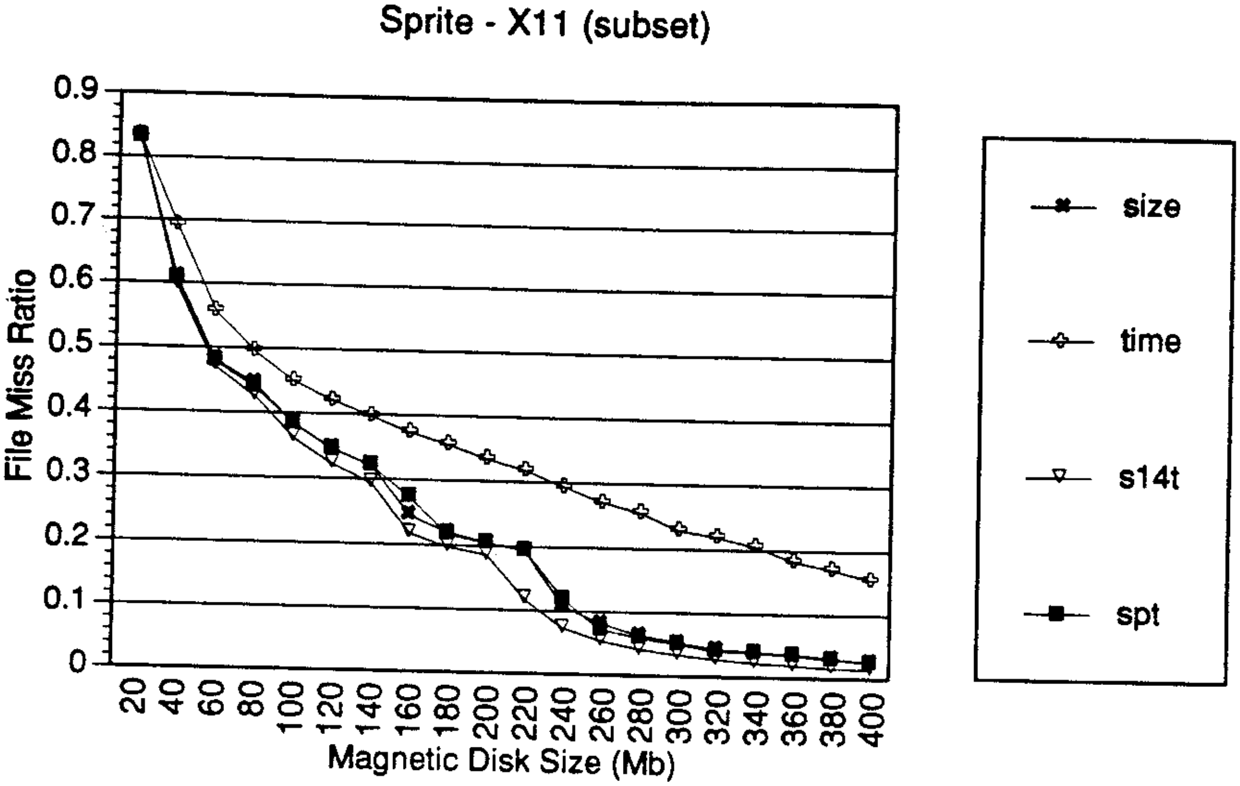


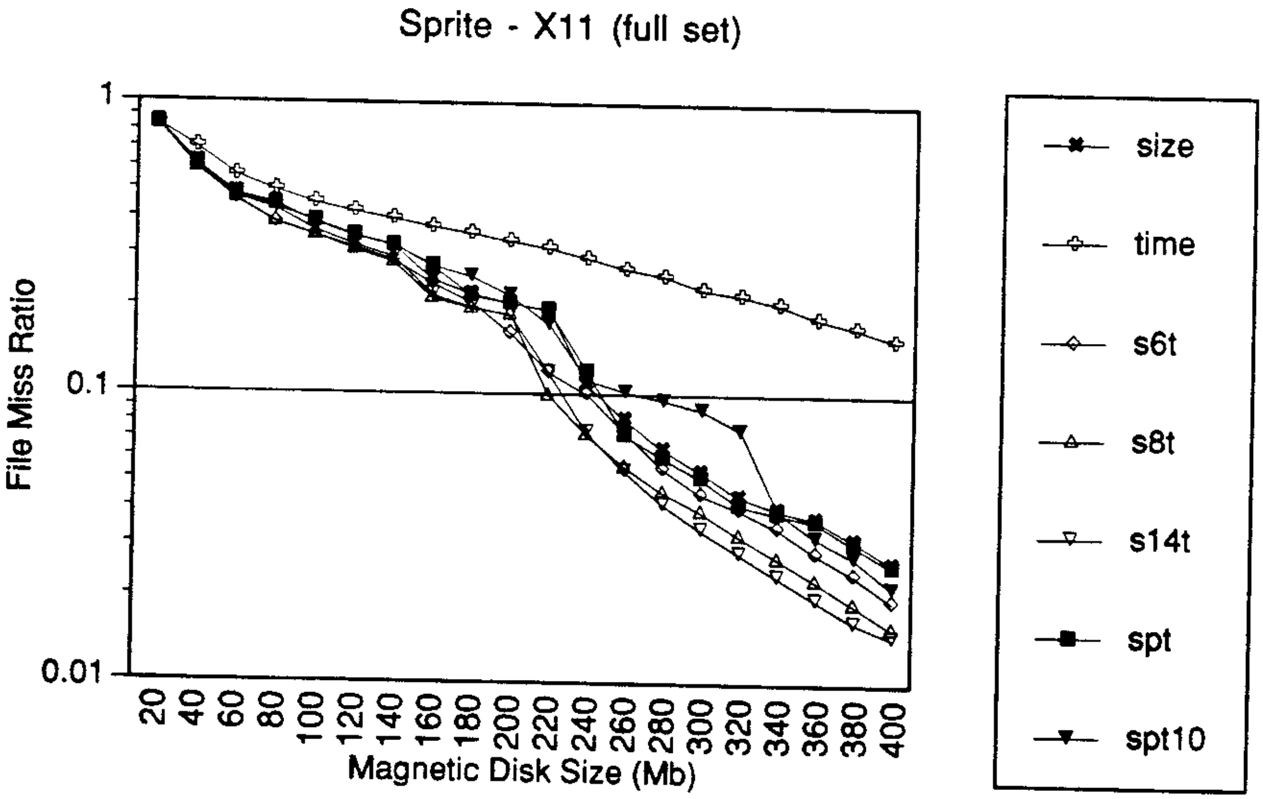
Sprite - pcs (subset)



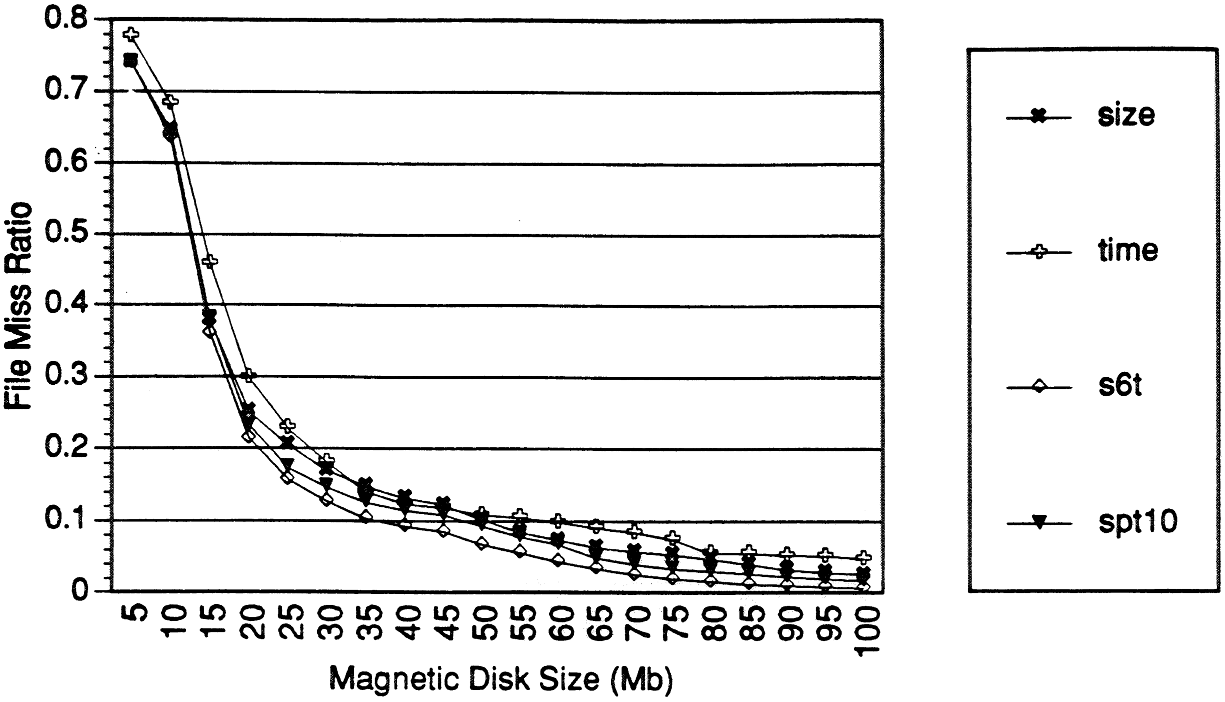
Sprite - pcs (full set)



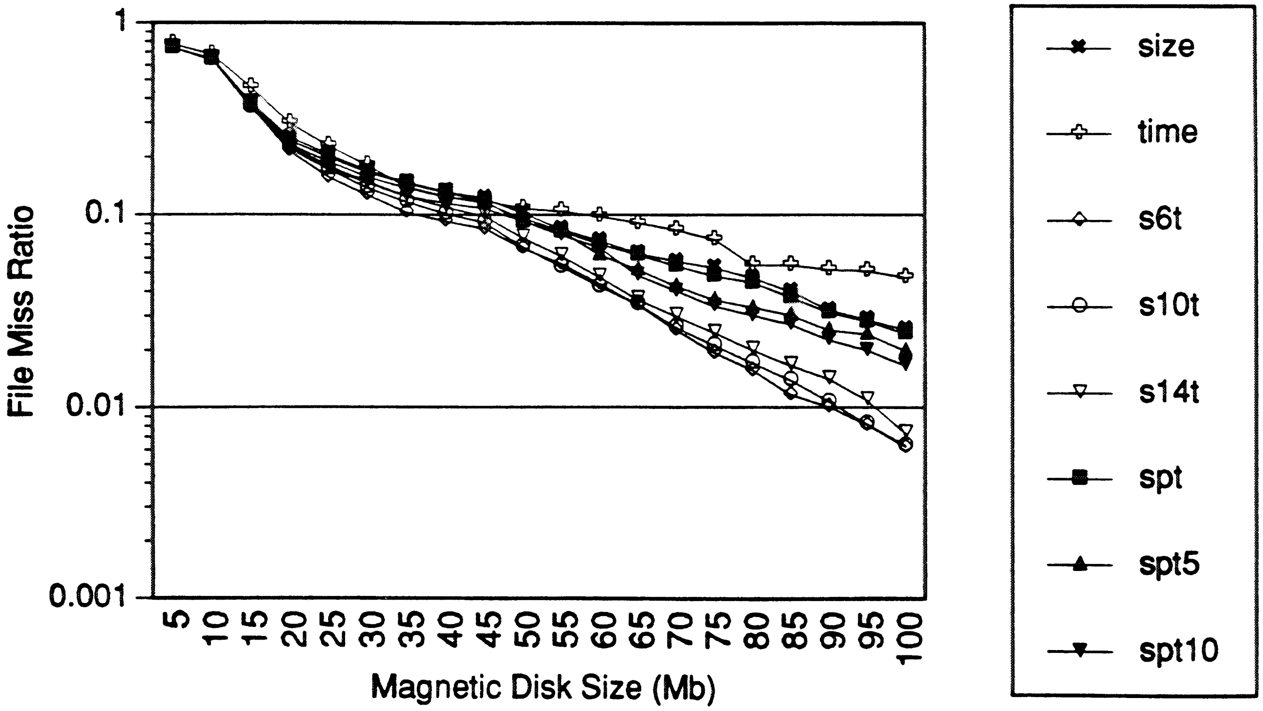

Sprite - X11 (subset)
File Miss Ratio
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
380
400
Magnetic Disk Size (Mb)
size
time
s14t
spt


Sprite - X11 (full set)
File Miss Ratio
1
0.1
0.01
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
380
400
Magnetic Disk Size (Mb)
size
time
s6t
s8t
s14t
spt
spt10

Sprite - Local (subset)



Sprite - Local (full set)


File Miss Ratio
Magnetic Disk Size (Mb)
size
time
s6t
s10t
s14t
spt
spt5
spt10