An Improved Long-Term File-Usage Prediction Algorithm

Lieutenant Colonel Tim Gibson
HQ USCINCPAC
C4 Systems Directorate (J6)
Camp H. M. Smith, HI, 96861
tgibson@acm.org

Ethan L. Miller*
Department of Computer Science and Electrical Engineering
University of Maryland–Baltimore County
Baltimore, MD 21250
elm@csee.umbc.edu

* Supported by the NASA Ames Research Center under grant NAG 2-1094.

As more computing centers collect files to use in data-mining or data-marts, managing long-term storage space becomes more important. We describe a new way to more accurately predict which files will be used in the future. This new method is an order-of-magnitude more accurate than any current technique. Fifty to eighty percent of all user files can be compressed or moved to tertiary storage with little impact on the user perceived performance. This paper supports these conclusions with an analysis of long-term (5-8 months) data collected on different types of computing environments.

1 Introduction

Disk storage prices continually decrease in price and increase in capacity; people often think disk storage is not a problem. However, this is not necessarily the case. While disk capacity continually increases, so do the number of files and the average size of the files we store on these disks. Administering and maintaining large file systems with many disks is difficult work requiring a skilled administrator. As a result, while storage capacity may not be a problem with newly purchased personal computers, it is a problem for network servers, scientific computing centers, and older workstations. Many of the files on large file systems are unused for long periods of time. Additional storage capacity can be "added" to these systems without adding more disks if the unused files can be compressed or moved to integrated tertiary storage, such as a tape or magneto-optical robot.[1] This paper presents an inexpensive and straightforward method to predict which files on a file system will be used next. The key to this new method is exploiting the reference locality characteristic of an individual file's access patterns. This locality of reference attribute has been observed previously by many researchers [3, 6, 25, 26, 27, 28]. By exploiting reference locality, large numbers of files can be identified for movement to tertiary storage with minimal user impact.

This paper is organized into an introduction and four other sections. In Section 2 we describe the environments we collected data from, how we analyzed the data, and our basic file system activity findings. We explain the different current migration algorithms, describe our new migration technique, apply both the old and the new migration algorithms to our collected data, and analyze the results in Sections 3 and 4. The new technique presented here, called *file-aging*, reduces the number of on-disk misses significantly—a miss occurs when a requested file is not on disk. The file-aging technique takes the activity patterns of individual files into account by rewarding frequently used files and punishing idle files. Punishment in this case equates to migration off disk. File-aging has been used previously to manage file caches in networked workstations [5, 31], but has not been applied to selecting tertiary storage management. Using this new technique, tertiary

[1] Note: The original purpose of this research was to examine new tertiary storage migration algorithms. As a result, we often focus on the migration aspect of the problem. However, these techniques can be applied to integrated file compression techniques.

This paper appeared at the 25th Annual International Conference on Computer Measurement and Performance (CMG '99), Reno, NV, December 1999, pages 639–648.

storage can be a viable way to extend current file system size. In Section 5, we present our conclusions, suggest how to implement the new technique on existing file systems, and discuss our follow-on research.

2 Data Collection, Computing Environments, and Analysis Tools

To collect information we performed nightly traces on different computing systems. Our tracing system was designed with one major goal: gather useful information without requiring operating system kernels to be recompiled. Our trace gathering tool is a modified version of the GNU find utility, which can be used to collect information on all the files in a file system or directory. The collected information includes the file's i-node number, size, name, access time, i-node creation time, modification time, owner, and group. The tracing program can be run any time, and if the output is placed into a directory or file system that is not being traced, the tracing process is invisible to itself. Additionally, the file's name and path can be scrambled to ensure user privacy on public systems while preserving information about the relationships of files in the file system.

The data analyzed in this paper came from two different computing systems at the University of Maryland, Baltimore County (UMBC).[2] Table 1 summarizes the systems.

The UMBC CS system is used primarily by the department's faculty and graduate students for program development, writing technical papers, and doing research. The UCS file systems are used by UMBC's 1,000 undergraduate CS majors and approximately 14,000 other users (faculty, staff, and non-CS majors) for electronic mail, writing papers, and program development.

The analysis tools used in this paper analyze the data collected and provide a wide range of information about both daily and long-term file activity. In addition to collecting file statistics, the analysis tools incorporate a file system simulator designed to test the effectiveness of different tertiary storage techniques. Many of these file system activity results have been reported earlier [8, 10, 12]. The findings can be summarized as follows:

- Normally less than 1% of all files on a file system are used every day, where used equates to being accessed, or modified;
- Files which are used have short inter-access and inter-modification times;
- Files which are used exhibit reference locality, where a file that is used is likely to be used again;
- Files which are not used for several days are unlikely to be used again;
- Files which have never been used are deleted more quickly than files that have been used; and
- The likelihood of a file being used decreases as the time since its last use increases.

Our file system collection and analysis package has some weaknesses. Ousterhout and others [3, 25, 31] noted that 80% of all file creations have a lifetime of less than three minutes. However, our system is designed to gather and analyze long-term disk use and file system activity; temporary files that exist for less than three minutes will not be moved to long-term storage and do not contribute to long-term growth. Hence, they can be safely ignored.

Additionally, our analysis system cannot determine how many times a file is accessed or modified in a single day. It only notices that an access or modification occurred and when the most recent event happened. Fortunately, what matters from either a long-term storage or a migration perspective is that the file was used on a certain day, not how many times it was used.

A separate but related issue is the politics of collecting data. While detailed information can be collected by changing the operating system kernel and generating a large detailed log file, it severely restricts the number and type of computing installations willing to collect traces. We had a difficult time convincing system administrators to allow us to run our daily collection program; we do not believe modifying multiple operating system kernels and convincing the different system support managers to use the modified kernels—and to provide us the additional log file disk space—is a feasible solution.

[2] We also collected long-term data at the U.S. Army's Aberdeen Proving Ground, and were provided data from the University of California, Santa Cruz. These data sets are not presented in this paper because the Aberdeen file system was very small and the UCSC traces were incomplete.

	University Computing Services (UCS)	Computer Science Dept. (Long Period)	Computer Science Dept. (Short Period)
Average Files on System	1,320,000	230,000	690,000
Disk Capacity (avg. percent full)	35 GB (70%)	11 GB (70%)	28 GB (50%)
Type(s) of file systems traced	User only	User only	User and System
Number of file systems traced	9 user FS	4 user FS	6 user FS, 2 system FS
Type of system	General-purpose	Development	Development
Number of users	~15,000	~500	~500
Trace length	239 days	287 days	157 days

Table 1. Summary of Systems Examined.

3 File Selection Algorithms

The algorithms described in this section focus on selecting when a file should be moved from disk to tertiary storage. The algorithms themselves are simply how a computer program chooses which files to move to free disk space.

3.1 Current Algorithms

We examined four previously tested migration algorithms in this research. While we did not expect to discover anything beyond the information provided by Smith [29, 30], Lawrie [18], and Strange [32, 33], we wanted to corroborate our methodology and findings with their research. The algorithms we examined are first-in, first-out (FIFO), least-recently used, (LRU), size only, and space-time.

The FIFO algorithm is very simple and is a common algorithm better known for its simplicity and ease of implementation than its efficiency. Using this technique, files on the system are migrated to tertiary storage based on how long they have been on the system. For example, files which have been on the file system for twelve months and are used daily will be moved to tertiary storage before a file that has never been used and is eleven months old. While it is simple, this technique has obvious limitations.

The least-recently used (LRU) algorithm moves files that have not been used in the longest time to tertiary storage first. This technique works well for memory systems and performs fairly well as a tertiary storage algorithm. Its biggest drawback is that it treats all files equally, regardless of size, which can cause problems for tertiary storage systems. The problem arises because LRU does not take into account the time the system needs to transfer files to tertiary storage.

This shortcoming is particularly noticeable when file size varies radically. When a tertiary storage system with magnetic tape stores large files (hundreds of MB or larger), the time required to read or write these files is a large part of the total transfer time (the remainder of the time being spent loading the cartridge and fast-forwarding to the file) [21, 22, 23]. However, with smaller files, the read / write time is often negligible, and the time required to mount the tape and find the file dominates the user wait-time [11, 14, 15, 16, 20]. An algorithm that ignores file size may migrate many small files to tertiary storage, instead of one large file with a slightly different migration value. The result is fewer files on-disk, and normally a higher on-disk miss rate and longer user wait times. Despite this drawback, some commercial systems, notably the AMASS software product from E-Mass, use the LRU algorithm exclusively [7].

At the other end of the spectrum from the LRU algorithm is the size-only algorithm. While the LRU algorithm moves files to tertiary storage simply based upon how much time has passed since the file's last use, the size-only algorithm picks migration targets based upon size alone. This algorithm suffers from the same problem as FIFO—a file's past activity pattern has nothing to do with selecting whether a file is chosen for migration. Despite this drawback, the size-only algorithm performs relatively well. Both Smith in 1981 [29, 30] and Strange in 1992 [32, 33], using supercomputing center data and the Berkeley Computer Science Department file systems respectively, found that using only the size as a migration indicator performed nearly as well as the space-time algorithm described in the next paragraph. From a practical standpoint, size-only is not always practical. Because it uses only a file's size, and hence ignores the time required to move files back from tertiary storage, the size-only algorithm can cause problems for users with large files. Large files will be constantly migrated using the size-only algorithm, regardless of how often these large files are used.

The final migration algorithm that was examined by previous researchers is the space-time algorithm, first described by Alan Smith in 1981 [29, 30]. The space-time algorithm uses the product of a file's size and time since its last use, essentially combining LRU and size-only. Additionally, the time component is normally raised to a power. Smith found that raising the time to the power 1.4 was optimal for his data. The complete formula is $Space \times Time^{1.4}$, where Time is the time since last use. A migration system using the space-time algorithm computes a migration value with this formula for every file on the file system. The files with the highest (or lowest, depending on the implementation) migration values are copied to tape, until the necessary disk space is freed.

Strange verified that the space-time algorithm was still optimal in 1992 [32, 33]. Additionally, Strange tried different values for the time factor's exponent, but empirically verified Smith's finding that 1.4 is optimal.[3] The space-time algorithm improves upon the LRU algorithm's success by also considering the file's size when choosing which files should be moved to disk. Currently, space-time is considered to be the best tertiary storage migration algorithm.

Despite this success, space-time does have a problem—it does not take file activity patterns into account. For example, consider two files of identical size are created Monday; one of the files is used daily for a week, and the second file is used only on Sunday. On Sunday evening, both files have similar migration values. An obvious solution to this problem is to recompute the migration value every time the file is used. However, files that are used frequently are still not distinguished from those which are used occasionally. A migration algorithm that takes activity patterns into account should improve on the space-time algorithm's performance.

3.2 The File-Aging Algorithm

The concept of file-aging was first introduced in 1994 by Dahlin [5] as a method for improving network file cache performance (in a file cache, the local workstation retains copies of frequently used files to minimize network traffic). Aging was also examined by Smith and Seltzer [31] in the context of file system benchmarks in 1997. The aging concept changes a file's migration value each day a file is used, so a file that is used regularly becomes a less likely candidate for migration. Similarly, if a file is not used, the migration value is altered so the file is more likely to migrate. The difference between file-aging and the other algorithms is how the migration value is computed and what information is used in the computation.

All the previous algorithms select files to migrate based upon a snapshot of the file's current age and size. None of these algorithms takes into account any past activity. The file-aging technique bases the migration value upon the time of the file's last use, the file's size, and the previously computed migration value. Using the previous migration value gives the file-aging algorithm an advantage over a simple snap-shot view of a file because the file's activity patterns are reflected in the file-aging algorithm's migration value. Files with small migration values are migrated off disk before files with larger values. The general steps the file-aging algorithm follows are:

```
if (the_file_was_created_today)
      Migration_Value = (X/Size) * 0.9;
else
      if
(the_file_was_accessed_or_modified_today
)
            Migration_Value =
Migration_Value + ((X/Size) * 0.9);
      else
            Migration_Value =
                  Migration_Value * 0.9;
```

When a file is created, its original migration value is based solely upon the file's size. After its creation, a file's migration value is based on how many times, and how often, it has been used. The "additive" line, Migration_Value + ((X/Size ...)) ensures that files which are used regularly are rewarded with a decreased likelihood of being moved to tertiary storage. Similarly, the last line of the algorithm punishes files that are not used regularly by gradually decreasing the migration value towards zero. Files that are not used several days in a row are more likely to be migrated than files that are used intermittently, even if the total number of days they have been used are equivalent.

Table 2 illustrates the algorithm on five 150 KB files with different usage patterns. One of the files is used daily and one of them is never used—the other three file's activity patterns lie between these two extremes. Shaded cells in the table indicate an inactive day.

The value of X in the algorithm can be modified so the algorithm is prejudiced towards different file sizes. The value used in this paper is 2,048. A wide range of values were tested; using values between 1,024 and 2,048 worked best. This is because the average (and median) file size on all the traced file systems is in the 1,024–2,048 byte range. More testing of the X variable is needed, but to do so requires data from file systems with different average file sizes. Similarly, the aging factor, in this case 0.9, needs more experimentation. We used different values for the aging factor ranging from 0.5 to 2.5.; the value 0.9 worked the best with our data sets.

[3] We also experimented with different values for the migration exponent, ranging from 0.5 to 2.5, and reached the same conclusion as Smith and Strange. Values of 1.2–1.35 and 1.45–1.6 yield slightly less favorable results than 1.4. Beyond these values, migration effectiveness, when measured by the on-disk miss rate, drops markedly.

Day	Used Daily	Used Alternate Days	Used Later	Used Initially	Never Used
1 (Created)	1.20E-02	1.20E-02	1.20E-02	1.20E-02	1.20E-02
2	2.40E-02	1.08E-02	1.08E-02	2.40E-02	1.08E-02
3	3.60E-02	2.28E-02	9.72E-03	3.60E-02	9.72E-03
4	4.80E-02	2.05E-02	8.75E-03	4.80E-02	8.75E-03
5	6.00E-02	3.25E-02	7.87E-03	6.00E-02	7.87E-03
6	7.20E-02	2.93E-02	7.09E-03	7.20E-02	7.09E-03
7	8.40E-02	4.13E-02	1.91E-02	6.48E-02	6.38E-03
8	9.60E-02	3.71E-02	3.11E-02	5.83E-02	5.74E-03
9	1.08E-01	4.91E-02	4.31E-02	5.25E-02	5.17E-03
10	1.20E-01	4.42E-02	5.51E-02	4.72E-02	4.65E-03
11 (Final Value)	1.32E-01	5.62E-02	6.71E-02	4.25E-02	4.18E-03

Table 2. Effects of the File-Aging Algorithm on Five 150 KB Files with Different Activity Patterns. *Shaded cells in the table denote inactivity.*

The file-aging implementation used in this paper also gives files a one day "grace period." Because files are immediately assigned a migration value based solely upon size when they are created, a large file can be migrated off disk immediately. If a large file is created one day, quickly migrated off because it is large, and used the next day, a disk miss occurs. To prevent this from happening, the file-aging algorithm does not consider a file for migration until at least one day had passed.

Because the file-aging algorithm considers a file's past activity when computing the file's migration value, file-aging is slightly more resource intensive than the other migration algorithms. Specifically, the previous migration value, a 32-bit floating point number, must be kept for the file-aging algorithm to use the next day while the other algorithms only require the system to check information the file system already maintains. Adding this field to the existing inode structure is straightforward; similar modifications have been made in the past [1, 4, 13, 35]. The file-aging algorithm has the same computational complexity as the space-time algorithm.

4 Algorithm Performance

This section provides an analysis of the old and the new algorithms using our file system simulator, and compares and contrasts the different algorithms' performance using several factors. The different algorithms' performances are compared as the amount of disk space available to the system decreases. This is done by having the simulator use only a percentage of the actual file system's size, and having it migrate the files which will not fit on the "smaller" disk to a tertiary device. Using this methodology, the previously tested algorithms have the same performance other researchers reported, while the file-aging algorithm performs better in every respect than the older techniques. The most commonly used measure of effectiveness is the number of files 'missed' (i.e., not on disk) when the file is required. Additionally, the simulator provides other performance measures, such as the number of files and bytes moved from disk to tape, the number of mid-day migrations forced by disk buffer overflows, and by distinguishing whether missed files and missed bytes were due to a file access or a file modification. The file-aging technique provides a large improvement over all of the other algorithms in every category.

The results shown in this section are normally from the most active sub-sections of the 239 day UMBC University Computing Services (UCS) traces and the 157 day Computer Science (CS) Department traces (see Table 1). Trends and initial findings observed in these file system sub-sections were confirmed by running the simulator on the entire UCS and CS file systems. The reason for limiting the scope of the experiments is the simulations running time. One simulation run, on either the CS or UCS traces, requires four hours on a single-user 400MHz Pentium II, with 512 MB of RAM and the Linux operating system. To fully evaluate the algorithms we ran a battery of 340 simulations, using with different migration algorithms, disk size restrictions, and other factors. The different simulation variations are briefly summarized in Table 3. Running this test battery on the complete set of UCS file system traces would have required approximately 55 days.

Test	Parameters	Migration Algorithms Tested
Basic	Disk capacity reduced from 90% to 10%, by 10% increments. Disk capacity further reduced from 10% to 2%, by 1% increments. Low water mark (disk buffer) set to 10%, high watermark set to 50%.	FIFO, LRU, Size-only, Space-time, File-aging
Pre-migration	Tests the effect of copying X of the most likely files to migrate to tertiary storage at night. X set to 0, 1,024, 2,048, and 4,096. Other parameters the same as the Basic test.	File-aging
Watermark tests	Uses 20% of original disk capacity. Low watermark set to 10%, 20%, 30%, 40%, and 50%. High water mark set to 10%–90% for each low water mark in 10% increments	File-aging
File Size Restriction	Uses 20% of original disk size, 10% low water mark, 50% high water mark. Restricts file migration to file larger than 0 KB, 1 KB, 2 KB, and 4 KB.	Space-time and File-aging

Table 3. Summary of the combinations in the simulator test battery.
Note: Because of space limitations, the test results on pre-migration, watermarks, and size restrictions are not shown in this paper.

To overcome this run-time limitation, we chose the most active sub-sections of the UCS file system and the CS file system, and conducted the entire test battery on these two file system sub-sections. When the simulations on the file system sub-section with the test battery revealed possible trends, we ran portions of the test battery on the entire file system trace to corroborate the findings. The sub-sections we chose to use were the dsk5 and faculty5 sub-sections, from the UCS and CS file systems respectively. A summary of the dsk5 and faculty5 characteristics is shown in Table 4.

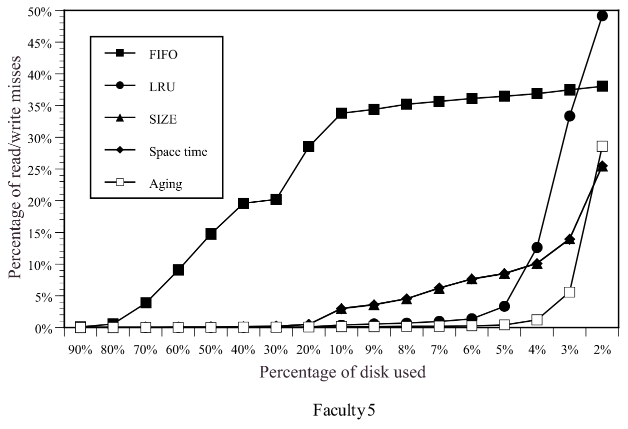
	Average Number of Files	Average Daily Transactions
All UCS file systems	1,329,000	53,635
dsk5 (UCS sub-section)	148,500 (11% of total)	7,542 (13.5% of total)
All CS file systems	690,523	30,345
faculty5 (CS subsection)	130,894 (19% of total)	15,372 (51% of total)

Table 4. Summary of dsk5 and faculty5 activity.

The nine different sub-sections of the UCS file system are closely balanced in terms of total files, and the dsk5 portion is only slightly more active than any of the others. In contrast, while the faculty5 sub-section holds its proportionate share of files (there are seven other sub-sections in the CS file system), it is very active and has more than half the entire CS file system's total transactions.

The most common method of measuring the performance of migration algorithms is to count the number of file misses. A file miss occurs when a file that has been moved to tertiary storage and erased from the magnetic disk is requested. This miss causes the migration system to copy the file back from tertiary storage to disk. File misses can be categorized as read misses, when the system needs to access a file and it was on tape; or write misses, when the system needs to both access and modify a file.

The overall miss rate on dsk5 and faculty5 is shown in Figure 1. In all four cases, the FIFO algorithm performs the worst. The LRU algorithm performs well, until a certain point is reached in the usable disk size—4% for the faculty5 trace and 10% for the dsk5 trace—below which the LRU algorithm's performance rapidly erodes. Space-time and size-only are indistinguishable from one another. This is true for nearly every measurement: file misses; byte misses; both the number of files and bytes moved to tape; and the number of migrations required.



Faculty5

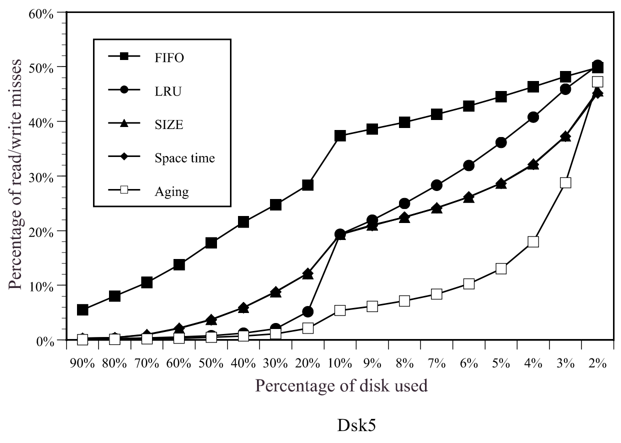


Figure 1. Performance of the different migration algorithms on the dsk5 and faculty5 file systems with decreasing amounts of disk space, 90%–2%. Shown by the percentage of on-disk misses. *Note scale change in the X-axis.*[4]

The file-aging algorithm performed uniformly better than any of the other algorithms in all but one instance. This exception occurred when the usable disk space was below 2% and is more clearly shown in Figure 2.

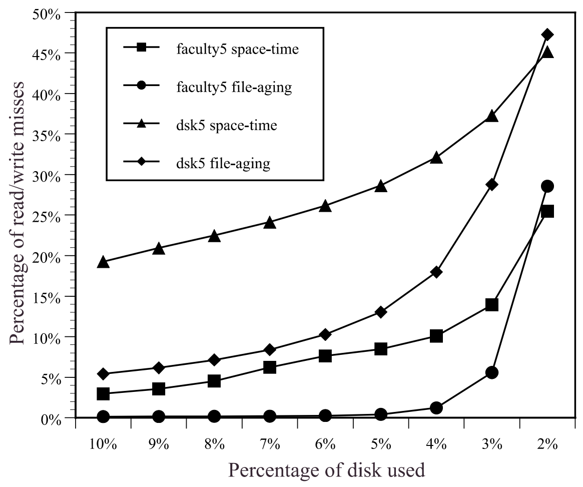
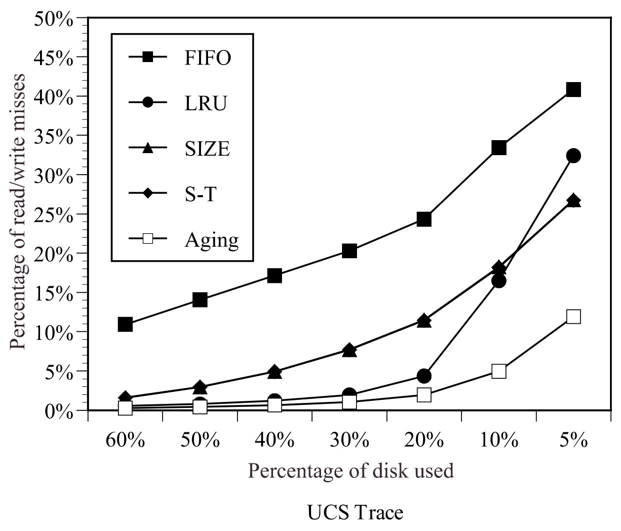


Figure 2. Performance of the space-time and file-aging algorithms on the dsk5 and faculty5 file systems with disk space between 10%–2%.

An abbreviated set of simulations for the entire UCS file system and CS user files provides results similar to those shown in Figure 1 for the dsk5 and faculty5 sub-set. These are shown in Figure 3.

[4] All figures in this section use the following simulator parameters: a low watermark of 10%, a highwater mark of 50%, files less than 2 KB are not migrated, and 1,048 files are pre-migrated nightly.



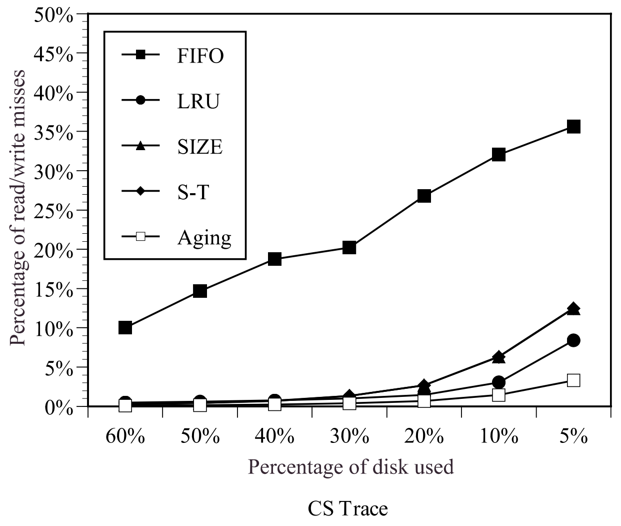
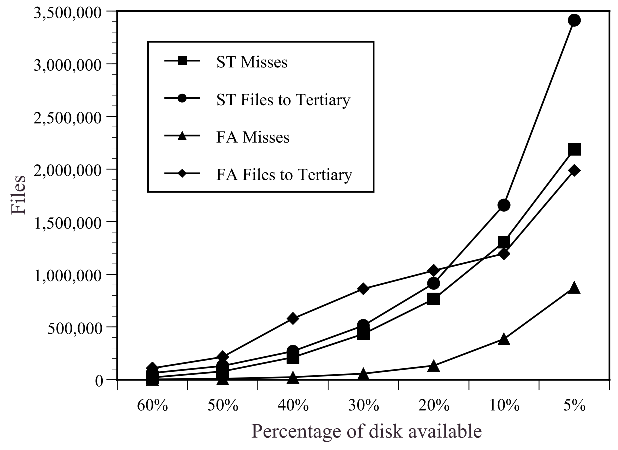


Figure 3. Performance of the different migration algorithms on the entire UCS file system and the CS file system's user space. Usable disk space decreases from, 60%–5%.
Note scale change in the X-axis.

The CS system-level files (e.g., /usr and /usr/local) are not part of the statistics shown in Figure 3 for two reasons. First, prudent system administrators would not keep system-level files on tertiary storage because these files are used too often. Furthermore, placing the system-level files, with their higher activity level, on tertiary storage would result in a much higher miss rate. Simulation runs show that this is indeed the case; the system-level file miss rate is normally three to five times higher than the user-level file miss rate for the same percentage of disk capacity.

An interesting observation about the space-time and file-aging algorithms comes from comparing their performance using the simulator's misses and moved statistics. The misses statistic is a count of how many files (or bytes) an algorithm moves to tertiary storage when it should have left the file on disk. The total files or bytes moved statistics is the number of files (or bytes) moved from disk to tertiary storage during the entire simulation. Figure 4 shows these statistics for the UCS traces (see Table 1). The number of files moved to tertiary storage and back to disk when they are required is quite large using either algorithm. The file-aging algorithm clearly outperforms the space-time algorithm because it has fewer misses, measured in both the number of files or bytes. Moreover, the file-aging algorithm moves significantly fewer bytes to tertiary storage than the space-time algorithm does. However, the file-aging algorithm occasionally moves larger numbers of files to tertiary storage than the space-time technique does. This is because it does a better job of selecting small files for migration. The combined result is more files—but fewer bytes—moved to tertiary storage and fewer on-disk file misses.



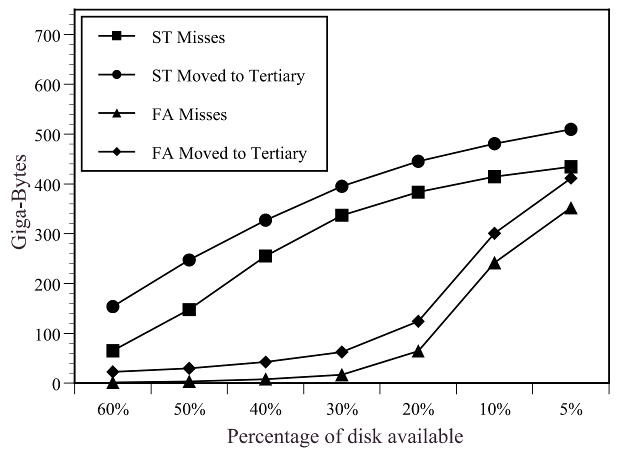


Figure 4. Performance of the space-time and file-aging algorithms using total number of file (and byte) misses and total number of files (and bytes) moved to tertiary storage, using the UCS data set. (15,000 user over 239 days)
Note scale change in the X-axis.

To corroborate our results with existing algorithms we compared our results with those of earlier researchers. The data in Figures 1, 2, and 3 for FIFO, LRU, size-only, and space-time are in agreement with the earlier findings of Smith [30] and Strange [32, 33]. For example, Strange [32] reported that a 5% miss rate for user files (i.e., not system-level files) could be expected using disk capacities of 12–22% and the space-time algorithm.
Both Smith and Strange found that attaining a miss rate of 1% using the space-time algorithm requires a disk capacity of 40–60% of the true disk capacity [29, 30, 32, 33]. File-aging allows a miss rate of 1% with only 10–20% of the true disk capacity. This is a substantial improvement.

5 Conclusion

We believe the file-aging algorithm provides significant improvements over any existing method for pre-determining which files will be used in the future. It is better than any previously implemented algorithm. For example, it is 2–20 times more effective than the space-time algorithm when judged using the number of files missed. It is also more effective in terms of reducing the number of files and bytes moved back and forth between disk and tertiary storage. The file-aging algorithm is no more computationally complex than the space-time algorithm; our file system simulator runs with nearly identical times using either the file-aging or the space-time algorithms. As a result, we believe it can be effectively used as a basis for tertiary storage systems.

As high capacity, low access time, near-line tertiary storage systems become available from companies like TeraStor [34], having a reliable method to accurately choose which files to place on tertiary storage becomes more important. We believe the file-aging algorithm can do this.

Implementing the file-aging algorithm is relatively straightforward. Systems which already incorporate integrated tertiary storage can substitute the file-aging algorithm for any current algorithm [2, 4, 13, 35]. The only additional requirement is to store the previous migration value for future use. For example, the AMASS system can easily use the file-aging algorithm because AMASS already keeps an extensive database of all the files on the system [7]. Adding a migration value to the database is a minor change.

We are currently researching migration algorithms based upon both file activity patterns and file owner activity patterns. We hope to increase our prediction accuracy by examining user behavior and studying the effects of migrating directories to tertiary storage. We are also studying existing real-time trace data from University of California, Berkeley and Carnegie-Mellon to study how our system performs with real-time instead of periodic trace data.

References

[1] C. J. Antonelli and P. Honeyman, “Integrating Mass Storage and File Systems,” *Twelfth IEEE Symposium on Mass Storage Systems*, Monterey, CA, 1993, pages 133–138.

[2] Edward R. Arnold and Marc E. Nelson, “Automatic Unix backup in a Mass-Storage Environment,” *USENIX–Winter 1988*, February 1988, pages 131–136.

[3] Mary G. Baker, John H. Hartmon, Michael Kupfer, Ken W. Shirriff, and John K. Ousterhout, “Measurement of a Distributed File System,” Operating System Review **25**(5), *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, 1991, pages 198–212.

[4] Ronald D. Christman, Danny P. Cook, and Christian W. Mercier, “Re-engineering the Los Alamos Common File system,” *Proceedings of the Tenth IEEE Symposium on Mass Storage Systems*, May 1990, pages 122–125.

[5] Michael D. Dahlin, Clifford J. Mather, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson, “A quantitative analysis of cache policies for scalable network file systems,” *Proceedings of the SIGMETRICS ‘94 Annual Conference on Measurement and Modeling of Computer Systems*, Nashville, Tennessee, May 1994.

[6] Bob Devine, Sunita Sarawagi, and Kannan Muthukkaruppan, “Caching for Tertiary Storage,” Unpublished paper, University of California, Berkeley, CA, 9 December 1992.

[7] The E-Mass Corporation maintains a full listing of their software, tape-robot, optical disk, and magnetic tape technical specifications at http://www.emass.com, 6 February 1999.

[8] Timothy J. Gibson, Ethan L. Miller, and Darrell D. E. Long, “Long-term File Activity and Inter-Reference Patterns,” *CMG98 Proceedings, 24th International Conference on Technology Management and Performance Evaluation of Enterprise-Wide Information Systems*, Computer Measurement Group, Anaheim, CA, December 1998, pages 976–987.

[9] Timothy J. Gibson, *Long-term File System Activity and the Efficacy of Automatic File Migration*, Doctoral Dissertation, University of Maryland, Baltimore County, May 1998.

[10] Timothy J. Gibson and Ethan L. Miller, “Long-Term File Activity Patterns in a UNIX Workstation Environment,” *Fifteenth IEEE Symposium on Mass Storage Systems*, Greenbelt, MD, March 1998, pages 355–371.

[11] John J. Gniewek, “Towards Improved Tape Storage and Retrieval Response time,” *Fifteenth IEEE Symposium on Mass Storage Systems*, Greenbelt, MD, March 1998, pages 81–94.

[12] Steven D. Gribble, Gurmeet Singh Manku, Eric A. Brewer, Timothy J. Gibson and Ethan L. Miller, “Self-Similarity in File-System Traffic,” *SIGMETRICS ‘98/PERFORMANCE '98, Joint International Conference on Measurement and Modeling of Computer Systems*, Madison, WI, June 1998, pages 141–150.

[13] 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, CA, January 1989, pages 65–84.

[14] Bruce K. Hillyer and Avi Silberschatz, “Scheduling Non-Contiguous Tape Retrievals,” *Fifteenth IEEE Symposium on Mass Storage Systems*, Greenbelt, MD, March 1998, pages 113–123.

[15] Bruce K. Hillyer and Avi Silberschatz, “On the Modeling of the Characteristics of a Serpentine Tape Drive,” *Proceedings of the 1996 ACM SIGMETRICS Conference on Measurement and Modeling of Computing Systems*, Philadelphia, PA, May 1996, pages 170–179.

[16] Theodore Johnson and Ethan L. Miller, “Benchmarking Tape System Performance,” *Fifteenth IEEE Symposium on Mass Storage Systems*, Greenbelt, MD, March 1998, pages 355–372.

[17] David W. Jensen and Daniel A. Reed, “File Archive Activity in a Supercomputer Environment.” *Technical Report UIUCDCS-R-91-1672*, Department of Computer Science, University of Illinois, Urbana, IL, 1991.

[18] D. H. Lawrie, J. M. Randal, and R. R. Barton, “Experiments with Automatic File Migration,” *IEEE Computer*, July 1982, pages 45–55.

[19] Daniel Menascé, Odysseas I. Pentakalos, and Yelena Yesha, “An Analytic Model of Hierarchical Mass Storage Systems with Network-Attached Storage Devices,” *Proceedings of the 1996 ACM SIGMETRICS Conference on Measurement and Modeling of Computing Systems*, Philadelphia, PA, May 1996, pages 180–189.

[20] Ethan L. Miller and Theodore Johnson, “Performance Measurements and Models of Tertiary Storage Devices,” *Proceedings of the 1998 Conference on Very Large Databases (VLDB’98)*, New York, NY, August 1998.

[21] Ethan L. Miller and Randy H. Katz, “An Analysis of File Migration in a UNIX Supercomputing Environment,” *USENIX Winter 1993 Conference*, San Diego, CA, January 1993, pages 421–434.

[22] Ethan L. Miller and Randy H. Katz, “Analyzing the I/O Behavior of Supercomputer Applications,” *Eleventh IEEE Symposium on Mass Storage Systems*, Monterey, CA, 1991, pages 51–55.

[23] Ethan L. Miller and Randy H. Katz, “Input/Output Behavior of Supercomputing Applications,” *Proceedings of Supercomputing ‘91*, Albuquerque, NM, 1991, pages 567–577.

[24] L. Mummert and M. Satyanarayanan, “Long-term Distributed File Reference Tracing: Implementation and Experience,” *Software—Practice and Experience*, Volume 26(6), June 1996, pages 705–736.

[25] John K. Ousterhout, Herve Da Costa, David Harrison, John Kunze, Mike Kupfer, and James Thompson, “A Trace-Driven Analysis of the UNIX 4.2 BSD File System,” Operating System Review **19**(5), *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, 1985, pages 15–24.

[26] R. Hugo Patterson, Garth A. Gibson, Eka Ginting, Daniel Stodolsky, and Jim Zelenka, “Informed Prefetching and Caching,” Operating System Review **29**(5), *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, 1995, pages 79–95.

[27] M. Satyanarayanan, “A Study of File Sizes and Functional Lifetimes,” *Proceedings of the 8th Symposium on Operating systems Principles, Association of Computing Machinery*, 1981, pages 96–108.

[28] Ken W. Shirriff and John K. Ousterhout, “A Trace-Driven Analysis of Name and Attribute Caching in a Distributed System,” *USENIX Winter 1992 Conference*, San Francisco, CA, January 1993, pages 315–332.

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

[30] Alan Jay Smith, “Long term file migration: development and evaluation of algorithms,” *Communications of the ACM* **24**(8), 1981, pages 521–532.

[31] Keith A. Smith and Margo I. Seltzer, “File System Aging—Increasing Relevance of File System Benchmarks,” *Proceedings of the 1997 SIGMETRICS Conference*, June 1997, Seattle, WA, pages 203–213.

[32] Stephen Strange, “Analysis of Long-Term Unix File Access Patterns for Application to Automatic File Migration Strategies,” *Technical Report UCB/CSD-92-700*, Computer Science Division (EECS), University of California, Berkeley, California, 1992.

[33] Stephen Strange, A Survey of Storage System Design Employing Automatic File Migration, submission for CS266, University of California, Berkeley, CA, 11 May 1991.

[34] The TeraStor Corporation provides a full listing of their new products at http://www.terastor.com, 16 January 1998.

[35] David Tweten, “Hiding Mass Storage Under Unix: NASA’s MSS-II Architecture,” Tenth IEEE Symposium on Mass Storage Systems, Monterey, CA, May 1991, pages 140–145.