FILE ACCESS PATTERNS

Carl Staelin

CS-TR-179-88

September 1988

File Access Patterns

Carl Staelin

Department of Computer Science
Princeton University
Princeton, NJ 08544

ABSTRACT

In this paper we examine patterns of file use which could be exploited by a file system to provide better service. We establish that many files have distinctive histories of access patterns. In particular, we are interested in finding which files are more likely to be accessed in the future, and also those files which are least likely to be accessed. These histories may be utilized by a hierarchical system to optimize file migration algorithms.

Introduction

We present the concept of a file cache and discuss file access patterns which make a file cache viable. A file cache will cache disk files in electronic memory in order to minimize I/O bandwidth requirements. Since users access data with skewed probability a relatively small cache should be able to capture a large fraction of disk accesses. We present results which establish methods for predicting how heavily a file will be accessed and how long a file should kept in a cache after it is closed, by examining past behavior for each file. We show that long term trends for any given file tend to be good predictors of future behavior.

Motivation

This work was motivated by a combination of three trends in computer architecture. First, processors have been increasing in power and speed dramatically. Secondly, the most common form of storage is the magnetic disk, which has been only slowly improving in both latency and transfer time, making the delay relative to the processor longer. Finally, until recently memory has been dramatically dropping in price and increasing in size. All these factors combine to make disk caches a viable approach for breaking the I/O bottleneck.

The first trend driving the need for higher I/O bandwidth and shorter response times is the development of faster processors, particularly in the micro-processor market. Within the last few years the power of the typical micro-processor has increased over ten-fold. These faster processors need at least a corresponding increase in I/O bandwidth or a decrease in I/O response times in order to fully realize their potential. Two standard approaches for increasing the available bandwidth are to either increase the bandwidth of devices, or to increase the number of devices.

Trends in magnetic disk technology are slowly decreasing response times coupled with rapidly increasing storage capacity. These trends lead toward lower bandwidth per byte of storage, which conflicts with newer processors' needs for increased bandwidth. In addition, since a large fraction of response time is attributable to rotational latency, it will be difficult to

improve response time by more than a factor of two† without increasing the rotational speed of the disks. As a result, it is possible to increase the available bandwidth using many small disks, but there is no simple method, other than caching, for reducing the response time. As a result, I/O will soon be a bottleneck for many applications.

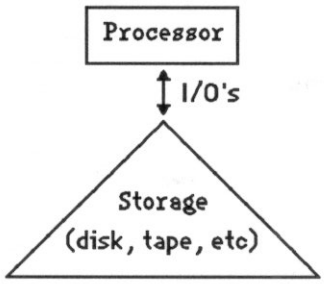
The third trend in computer technology is toward larger and cheaper memory. Even though there has been a recent jump in the price of memory chips, we feel the downward trend will surely re-assert itself as new memory production comes on line. As a result, it is now possible to consider designing computers with massive amounts of memory, which will be available for tasks other than the traditional main memory store. Thus a new resource with both high bandwidth and low latency is available to the computer architect for increasing computer system performance.

Hierarchy

The three trends outlined above drive the need for more I/O bandwidth and less delay, rule out the obvious solution of building faster disks, and offer a possible solution. Ever since people noticed that computers access data in a skewed fashion, computer designers have made use of this fact by building memory hierarchies to solve bandwidth and response time problems. These hierarchies try to provide storage at the cost of the slowest elements with the performance of the fastest elements. The simple solution is to create an analogous storage hierarchy with memory and disk, but this is only a partial solution since there are several storage technologies available.

---

† Since essentially all disks rotate at 3600 revolutions per minute, the average delay due to rotational latency is 8.3 milliseconds. For high performance disks such as the Amdahl 6380J the average seek delay is 12 milliseconds. Ingoring other delays such as the protocol time and the transfer time, nearly half the delay is due to average rotational latency.



Block Diagram of Computer System

We define a hierarchy to be a pyramid with the smallest, fastest, and most expensive storage at the top, and the largest, slowest, and cheapest storage at the bottom. By creating a hierarchy we add the necessity for migrating objects up and down the hierarchy to optimize performance. Since caching seems to provide the most efficient use of the available resources, the introduction of caching algorithms at all levels of the storage hierarchy would be useful.

The traditional model of a hierarchy is the memory hierarchy. All computers have some form of memory hierarchy with registers at the top and main memory below. We call the memory hierarchy traditional because the memory is layered, with each layer having well defined price-performance benefits. As one travels down the hierarchy both performance and price drop, while the size increases. Typically, the size would increase by one or more orders of magnitude, while both the performance and price per byte would decrease by one or more orders of magnitude. This is a simple model because there is a strict ordering of technologies according to performance, so they may be logically stacked to form a hierarchy.

However, a storage hierarchy is more complex than the simple memory hierarchy because there are at least two obvious and independent performance measures, transfer rate and latency. We define the transfer rate to be the number of bytes per second that a device can deliver data, once it has sent the first byte. The latency is the time from the initiation of the I/O request to the delivery of the first byte of data. Obviously, latency can be attributed to a variety of factors, such as protocol overhead or disk seek and rotation, which may depend on both the technology and the manner in which the technology is used. Since transfer rate and latency are independent properties, the various technologies do not fit logically into a conventional

hierarchy since they cannot be strictly ordered according to performance. As a result, the performance may depend on the problem, with one technology outperforming another in certain instances, and the reverse happening in other cases. However, some technologies are clearly superior to others, such as disk is superior to tape in both performance measures.

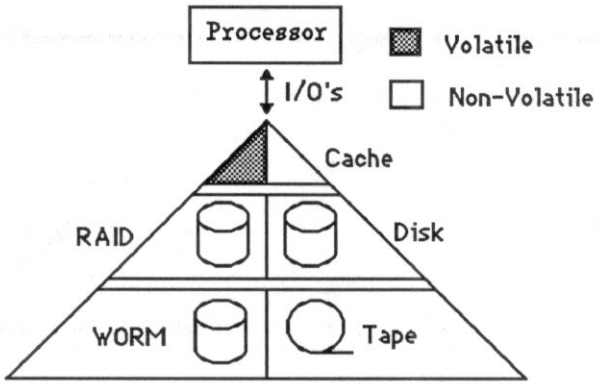
Typically computers have a variety of storage technologies, each of which has its own set of phsyical characteristics, such as storage density or cost per byte. Some of these characteristics are important from a performance standpoint, others are important for political or economic reasons. Some examples of simple characteristics for tape storage are that it is removable and that it is accessed sequentially. An example of a device with an unusual characteristic is the optical write-once-read-many (WORM) drive, where space is not reusable and data can only be written once. Sometimes unusual properties such as this make it difficult to compare two technologies, since the two devices are inherently different.

There is another interesting difference between memory hierarchies and storage hierarchies, and that is the issue of volatile versus non-volatile storage. Most operating systems assume that all secondary storage is non-volatile, so that all changes are permanent. This presents problems when caching written material, since most memory systems are volatile and updates may be lost if the system crashes. However, write-through caches would tend to be inefficient† because twenty to thirty percent of all I/O's are writes[1]. Thus the careful inclusion of volatility into a storage design by allowing files to be designated as volatile might dramatically improve performance.

---

† One alternative is turn off caching for certain files either permanently or on certain conditions.

1 [SMITH85], page 190



Sample Storage Hierarchy

Since storage systems are constructed from technologies with vastly different operating characteristics, a more complex storage model is needed to adequately describe the system. An appropriate model might be a tree hierarchy in which technologies which are different and where neither is consistently "better" would be considered siblings, and demonstrably superior technologies would be considered parents. For example, a disk cache with some non-volatile memory for written data would be the root of the tree, and the disks would be children of the cache. In addition, disks which are organized into redundant-arrays-of-inexpensive-disks (RAIDs) would be siblings of normal disks since the average response time for a RAID is longer than a normal disk while the transfer rate is substantially higher than a single large disk. Finally, tapes might be children of both disks and RAID's since tapes are slower than either technology.

Overview of Cache

The concept of a cache is very simple; a small portion of fast storage is used to hold a fraction of the data from a larger, slower storage system. The small store attempts to hold the most frequently accessed elements from the large store so that most accesses will proceed at the speed of the faster store. Caches are common for memory systems, and it is not unusual for memory caches to absorb ninety to ninety-nine percent of all memory accesses.

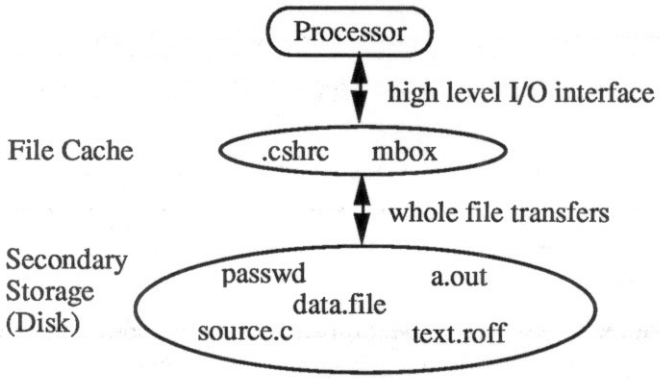
There are two properties that caches exploit in order to achieve the greatest efficiency. The first is the property of spatial locality, and the second is the property of temporal locality. The property of spatial locality relies heavily on the concept of "neighbors". Obviously, different blocks of the same

file could be considered neighbors, whereas blocks residing next to each other on disk but belonging to different files would probably not be considered neighbors. Extending this concept further, one could imagine that files within the same directory might exhibit a weak form of neighborhood. Memory caches take advantage of this property by caching a whole line of words on each cache miss. Storage hierarchies can take advantage of this property in a variety of ways by acknowledging one or more forms of locality.

Caches also try to capitalize on the property of temporal locality. Temporal locality implies that items which have been accessed recently will probably be accessed again in the near future. Caches make use of this behavior by retaining most of the recently accessed data for some time after it has been accessed. Most caches use some form of Least-Recently-Used (LRU) replacement strategy to keep the most recently accessed data in the cache. While this algorithm is simple and usually effective, algorithms which are sensitive to past behavior of particular files might be more successful.

Terminology and Concepts

We present the two new concepts file cache and file temperature. A file cache will cache whole files, unless the file is very large. If the file is too big to handle as an atomic unit, the cache may break the file into smaller pieces as it sees fit. The essential difference between file caching and block caching is the fact that file caching deals with storage on a logical basis (files), whereas block caching deals with storage on a physical basis (disk addresses). The primary advantage of file caching is the fact that caching whole files rather than blocks or tracks allows the cache to deal with fewer objects at a more abstract level. It may store more information about each file, so that it can make better decisions, and it may start to deal with the operating system at a more abstract level. For example, a file cache might know that a particular file is typically very hot when it is opened, and that it is not opened very often. When the file is opened, the cache may pre-stage the file at the open, and flush the file at the close.



File Cache Concept

The potential drawbacks to this method of caching are wasted cache space and wasted I/O bandwidth. Since file caching typically reads a whole file into the cache, it is possible that much of the bandwidth and space required to cache the file will be wasted if only a part of the file is accessed. However, if the whole file is accessed during the course of the open, or if the file is reopened before the file is flushed from the cache, then these resources would not have been wasted.

The second concept is that of file temperature. In cache related literature the terms "hot" and "cold" denote objects, such as words of memory, which are accessed heavily and lightly respectively[1]. We extend this concept and could give a simple definition for file temperature as the number of I/O's the file receives. Since files don't all have the same size, this simple definition is inadequate. For example, if two files each receive a hundred 512-byte I/O's, and one file is a thousand bytes long while the other is a million bytes long, then the thousand byte file is obviously "hotter". A more complete definition of file temperature is the number of bytes transferred divided by the number of bytes in the file. Given this definition the two files from the example above would have vastly different temperatures. We will treat this definition of file temperature as our working definition.

Unfortunately, even our working definition of file temperature is slightly inadequate because we don't take into consideration time intervals and because file sizes are not constant. For example, a file may be accessed very heavily on Monday, and not at all on Tuesday. If only accesses from Monday

---

[1] [BACLAWSKI87], page 3

were counted, then the file would be correctly considered hot. However, if accesses from Monday and Tuesday were aggregated, then the file might be incorrectly considered to have been hot on Tuesday. This problem is not as severe as it might seem, since each level of a hierarchy should take into consideration the appropriate time frame for its temperature computations. A second example might be the temperature of a spool file, which is continually growing and shrinking. Since the file size is not constant, our definition of file temperature is vague. A simple solution to this problem is to approximate the file size as some function of the file sizes at file open and file close.

Prior Work

There are a variety of areas in which the prior work touches the problem of the I/O bottleneck and which therefore touch on the problem of file access patterns. These areas include disk caches, network file systems, and databases, along with a collection of other topics. A large fraction of this work mentions the topic of file access patterns in passing and then focuses primarily on a method for increasing I/O bandwidth. The results from this research can be fit into three categories: basic file access patterns, uni-processor disk caches, and network (typically single user workstations) disk caches.

The first area of research which addresses the problem of the I/O bottleneck and file access patterns is the design of disk caches. Much of this research is focussed on the choice of cache size, and cache line size. The results regarding cache size are both relevant and important. However, the results for optimal cache line size are only applicable for caches with a fixed line size since most work in this area tends to ignore the underlying file structure in favor of physical addresses. The most notable papers in this area are [OUSTERHOUT85], [GROSSMAN85], [SMITH85], [FLOYD86], [MCKUSIK84], [FRIEDMAN83], and [BASTIAN82].

A second area of work which is strongly influenced by the I/O bottleneck is the design of networked file systems, particularly in a diskless workstation environment. Since most common networks have a bandwidth comparable to many disks and since that bandwidth must be shared by several

processors†, the bottleneck is even more pronounced than in the stand-alone world. Thus, most network file systems incorporate some form of local disk cache to reduce network traffic and delay. The most notable papers on network file systems are [HOWARD88], [NELSON88], [LAZOWSKA86], [SCHROEDER85], [SATYANARAYANAN85], and [HAGMANN87].

A third topic which is heavily influenced by the I/O bottleneck is database design. Many database systems incorporate some form of internal caching (typically using some form of buffer management), and some research has been done to analyze predictive caching methods for database systems. The most interesting paper in this field is [SMITH 78], which discusses predictive prefetching of data for sequential scans.

Finally, there are a diverse assortment of other topics which touch briefly on the problem of the I/O bottleneck and file access patterns. The most similar topic is algorithms for the automatic migration of files between tape and disk. Another similar topic is the management of electronic disks, which typically have massive electronic memories with reserve battery power and which talk to the host via a standard disk interface. Some relevant papers from these and other topics are [PATTERSON88], [HENLEY87], [SATYANARAYANAN81], [BASTIAN81], [FRIEDMAN83], [SMITH81], and [VOLDMAN83].

The first group of results on basic file access patterns were obtained by experiment and they can be summarized as follows. First, sixty to seventy percent all opens access the whole file in a UNIX environment[1]. Secondly, twenty to thirty-five percent of all I/O's are writes[2]. Thirdly, fifty percent of all newly created files are deleted within five minutes on a UNIX system[3].

The results regarding disk cache design were obtained both from experiment and modelling, and they can be summarized as follows. Block

---

† Ethernet can handle at most 1.25MBytes per second while disks can handle up to 3.0MBytes per second

1 [OUSTERHOUT85], page 15

2 [SMITH85], page 190, and derivable from [OUSTERHOUT85] data on number of read-only, write-only, and read-write opens.

3 [OUSTERHOUT85], page 15

prefetching schemes can reduce miss ratios substantially[4]. Caches closer to the processor which store data from several devices provide better hit ratios per byte of cache than several smaller caches located further from the processor[5]. Caches that are roughly one percent of the total storage space can give hit ratios of ninety percent[6].

The last group of results focus on network disk caches. It has been shown that a small local disk caches, on the order of a few megabytes, can dramatically reduce the network bandwidth required to support a distributed file system[7]. The Sprite Network File System reports eighty-nine percent hit ratios for reads with four megabyte caches[8]. In addition, it has been found that each user needs an average bandwidth of a few hundred bytes per second, but the peak requirements are significant compared to the total available bandwidth in that system.

Data

In order to design a smart file cache we conducted some experiments to determine if there were any useful file access patterns which would allow us to predict future behavior more accurately than a simple LRU algorithm. Amdahl Corporation generously allowed the author access to trace data and computing resources necessary to conduct this research during the spring of 1988. We were primarily interested in dynamic access patterns. The utility which generated the data was IBM's System Management Facility (SMF), which provided us with trace data for each file open and close in IBM's MVS/XA operating system.

We used data from two MVS sites using IBM's SMF. Most of the analysis was done on SMF trace data using SAS® and Merrill's MXG®† Package. Essentially, SMF records file activity information on each file close. In addition, at certain intervals SMF records information for each active process

---

4 [SMITH78], page 243

5 [SMITH85], pages 171-172

6 [MENON88], page 150

7 [NELSON88], page 141

8 ibid.

† SAS is a trademark of the SAS Institute Inc. and MXG is a trademark of Merrill Consultants

and open file. This data is transformed into SAS data sets by Merrill's MXG package. However, SMF data is somewhat unreliable, and not all the data we desired is recorded by SMF, so we could not answer certain questions.

The SMF traces contain tremendous quantities of data, most of which is not relevant to our current work. Primarily, we can generate a table of file opens. For each open we know: which file was opened, roughly how many I/O's were done during the open, when the file was opened, when the file was closed, how many tracks were allocated to the file at the time of the close, the name of the job which opened the file, and the logical name the job assigned to the file.

The first limitation of SMF data is the fact that we cannot compute file temperature using our working definition of bytes transferred over bytes in file. Since SMF only records the number I/O's to a file and the number of tracks allocated to the file, we must redefine file temperature as the number of I/O's to the file over the number of tracks allocated to the file. This new definition is used with our experimental data, and it is the "temperature" referred to in our results.

The most glaring shortcoming of SMF data is the fact that it collects summary statistics for each open. There is no record of the timing between various reads, nor even which blocks or tracks were accessed. In addition, SMF merely keeps a record of the number of I/O's to the file, it has no reliable indication of the distribution between reads and writes. For those opens which are read only or write only we can correctly assess the number of reads or writes, but for those opens which are read/write we cannot distinguish between reads or writes. Thus we have difficulty both comparing file caching to block caching and assessing the impact of various write strategies.

A second problem with the SMF data is that fact that it only records the number of tracks allocated to a file, not the actual space used by the file. In MVS the user specifies at file creation time the initial size for the file. This size is the space allocated for the file. The true file size is the number of bytes of data stored in the file. Since our definition of file temperature is sensitive to file size, and since we only know the allocated size of a file and not the space actually occupied by the file, there is some experimental error in our temperature calculations. In addition, since most of the small files have

allocations of one cylinder or a few hundred thousand bytes, and since the smallest unit of allocation is one track or thirteen thousand to forty-seven thousand bytes[1], the problem seems to be more severe with small files. Because most files are small our expected performance calculations will be biased. Note that the errors will tend to lower the file temperature.

Another problem with the SMF data is the fact that we only have data on those files accessed during the measurement period. This means that we cannot tell how large the entire file system is, nor can we tell how long it has been since each of the cold files has been accessed.

In MVS a partitioned data set is a file which consists of a set of member files. A final problem with the SMF data is the fact that it is impossible to tell which member or members of a partitioned data set were accessed on a given open. The SMF data only records the name of the partitioned data set not the name of the member which was accessed. Essentially, this means that logically separate files are being treated as a single unit. As a consequence, the temperature of the group is the average of the individual temperatures, and the temperature for each open is computed as if each member had been opened. One may think of this as diluting the temperature of the hottest members, producing a uniform lukewarm temperature. Since some members will probably be hotter than others, this dilution will tend to downplay the existence of hot spots, which will tend to bias to our results against caching.

The data we used for our experiments came from two installations which we will call customer "S" and customer "Z". The only information we received from these shops was a trace of SMF data from some period of time. The first trace, from customer "S", contained a week's worth of data and 170,437 disk file opens. The second trace contained three days of data with 142,795 disk file opens.

One interesting feature of the data is that there are thousands of temporary data sets which consume ten to twenty five percent of the total I/O's and whose cumulative size is fifty to eighty percent of the total space consumed by all files. However, the space used by temporary files at any given time is relatively small. This fact produces an interesting accounting

---

[1] For the IBM 3330 and 3380 devices respectively.

problem from the standpoint of file system size since there is no easy way to account for the space used by temporary files. The primary problem is that the space is reused by the system, and it is not obvious how to attribute the I/O's to particular locations. As a result we ignored temporary files, with the assumption that they require little space.

Research Questions

Given the available data and the working design of a cache, we tried to develop better caching strategies than a simple LRU algorithm. Our work focussed on three areas: file temperature, spatial locality, and temporal locality. In each case we tried to develop methods for predicting behavior, particularly by examining past behavior.

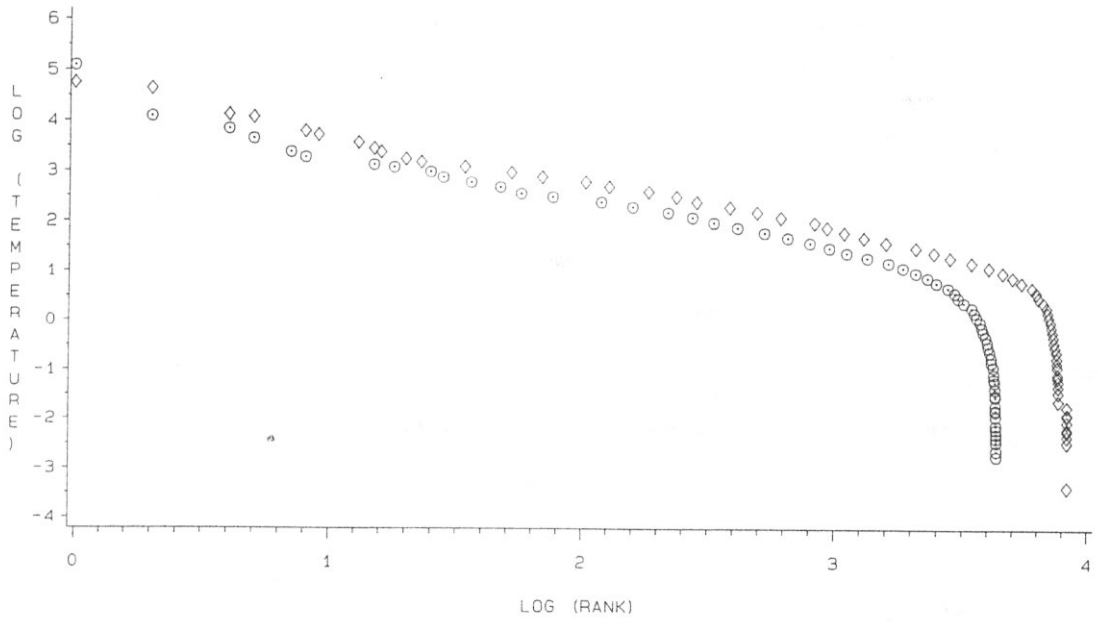
The first question was whether file temperatures were skewed. If file temperatures were not skewed, then disk caches would not work because each file would be accessed with equal probability. However, file temperatures are strongly skewed. In figure one we show the temperature curve of the file systems for both customers. The graph shows the log of the temperature versus the log of the rank of the file. The rank is an index from one to the number of files, with the hottest files having the lowest rank. The temperature is simply the number of I/O accesses the file received during the experiment over the number of tracks allocated to the file. The interesting point is the fact that the temperatures vary over five orders of magnitude, with the majority of files having lower temperatures.

The second property we tried to establish was file open temperature consistency. We define *file open temperature* as the temperature of a file for a given open. If files had the same temperature on each open, then the cache replacement strategy would be simplified because there would be a constant ranking of files. As expected, however, file open temperatures are not constant in most cases, and are in fact very random in nature. Figure two shows the file open temperature for a representative file over sequential opens. It is obvious that the file open temperature for any given file is not constant.

Since file open temperatures are not constant, the file open temperature probability density function (PDF) is an important property for

FILE TEMPERATURE

NON-VSAM



○ CUSTOMER Z
◇ CUSTOMER S

FILES ARE RANKED BY DESCENDING TEMPERATURE

FIGURE 1

FILE OPEN TEMPERATURES FOR A SAMPLE FILE

NON-VSAM

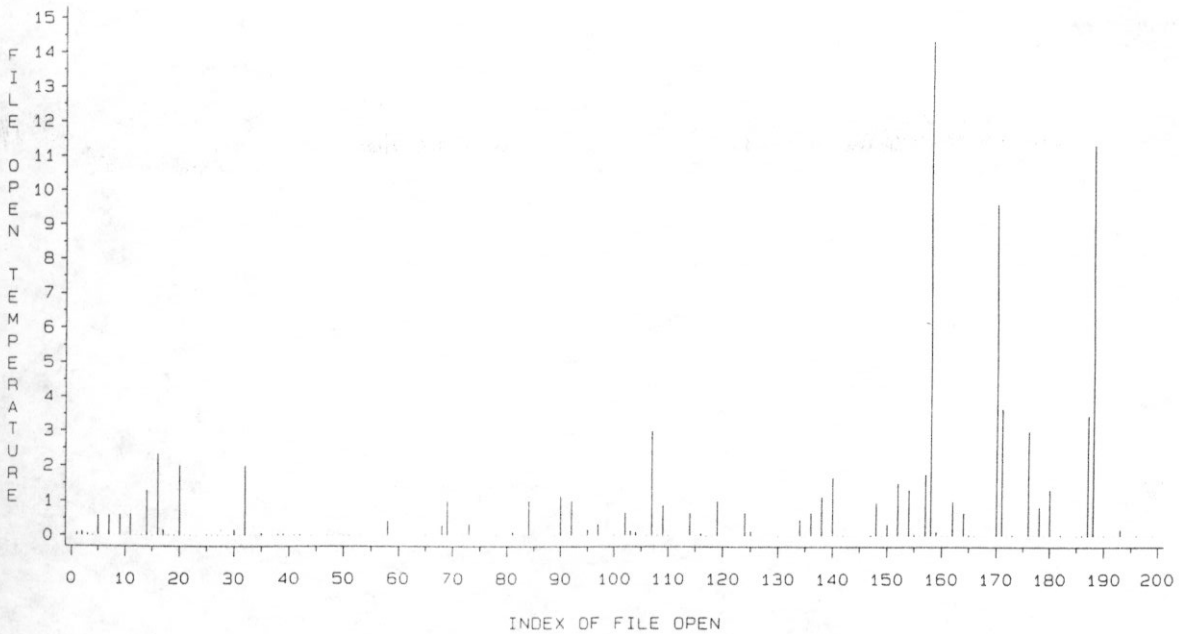


FIGURE 2

each file. If the PDF is "regular" in some sense, then the cache could guess more accurately which files to promote. A "regular" PDF might be an exponential or uniform distribution. Since PDF's are not directly observable, we assume that the histograms of file open temperatures provide a reasonable approximation to the PDF.

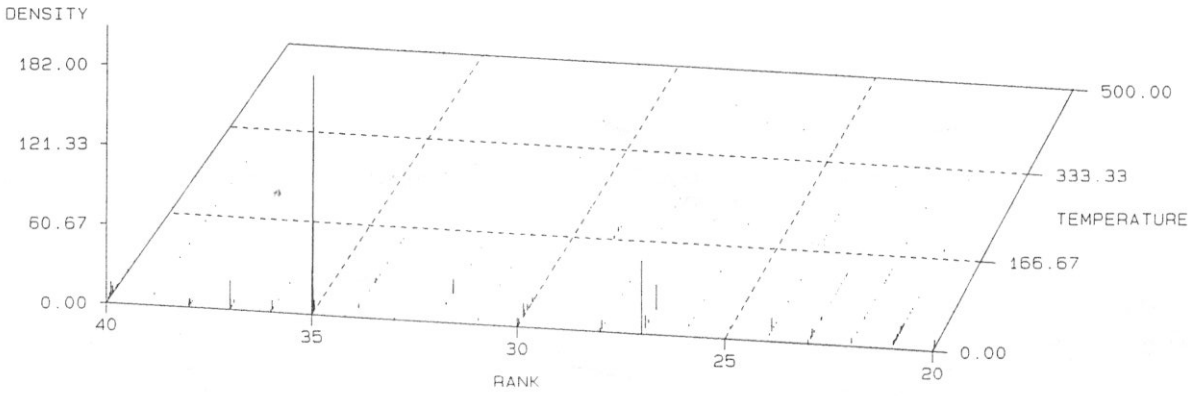
In figure three we show some file open temperature histograms for a sampling of files. This is a three dimensional plot, with the files arrayed on the "rank" axis, the file open temperature "buckets" on the "temperature" axis, and the number of opens per "bucket" on the "density" axis. The "buckets" are really the conversion of the real valued file open temperatures to a discrete axis. The rank is the identical to the rank in figure one, and all temperature values associated with a given file have the same rank. However, many file open temperature histograms seem to be random scatter plots rather than regular functions, so in general this avenue would appear to be a dead end.

We then tried to establish whether file temperature was at all consistent. Essentially, file temperature consistency implies that file temperatures are similar during consecutive time periods. It turns out that long term file temperature is reasonably consistent, and that temperature over a period of a few days can predict temperature for subsequent days. Since we only had a week's worth of data, it is not possible for us to see how temperature varies over periods of more than a week, nor were we able to determine how well long term temperature predicts temperatures further in the future.

In figure four we show how long term temperature can predict temperatures over subsequent days. The graph represents the cumulative percentage of the I/O's absorbed by the files during each of the three time frames. The first time frame, called the calibration period, was used to establish a ranking of files according to file temperature. This period was three days long. The two remaining curves show what fraction of the I/O during the two subsequent days is absorbed by the hottest files from the calibration period. Looking at the graph one may see that long term temperature is a reasonable predictor of future temperature, since most of the I/O's are absorbed by the hottest files from the calibration period. It is interesting to note that the calibration period predicted the fifth day's accesses better than it predicted the fourth day's accesses.

FILE TEMPERATURE FOR 21 FILES
HISTOGRAM OF TEMPERATURE

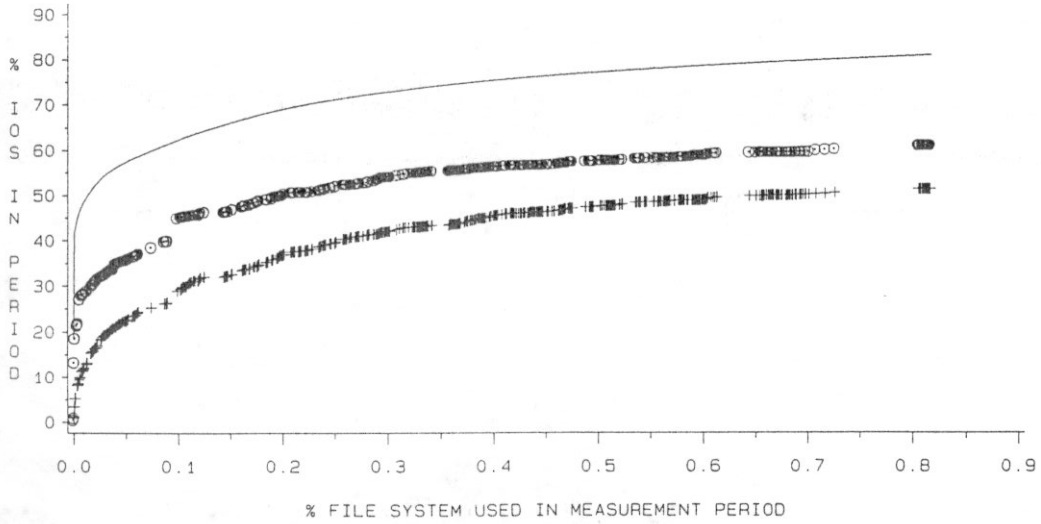
CUSTOMER "S"
NON-VSAM



CLIPPING VALUE FOR TEMPERATURE: 500
POSITIVE TEMPERATURE VALUES ONLY
GRANULARITY OF 1 EXCPS/TRACK
FIGURE 3

CUMULATIVE %IOS VS CUMULATIVE %FILESYSTEM
MEASURING PERSISTENCE

CUSTOMER "S"
NON-VSAM



\- MEASUREMENT PERIOD OF 0 ... 71 HOURS
\+ TRIAL PERIOD OF 72 ... 95 HOURS
o TRIAL PERIOD OF 96 ... 119 HOURS

HOTTEST 1000 FILES FROM MEASUREMENT PERIOD
FIGURE 4

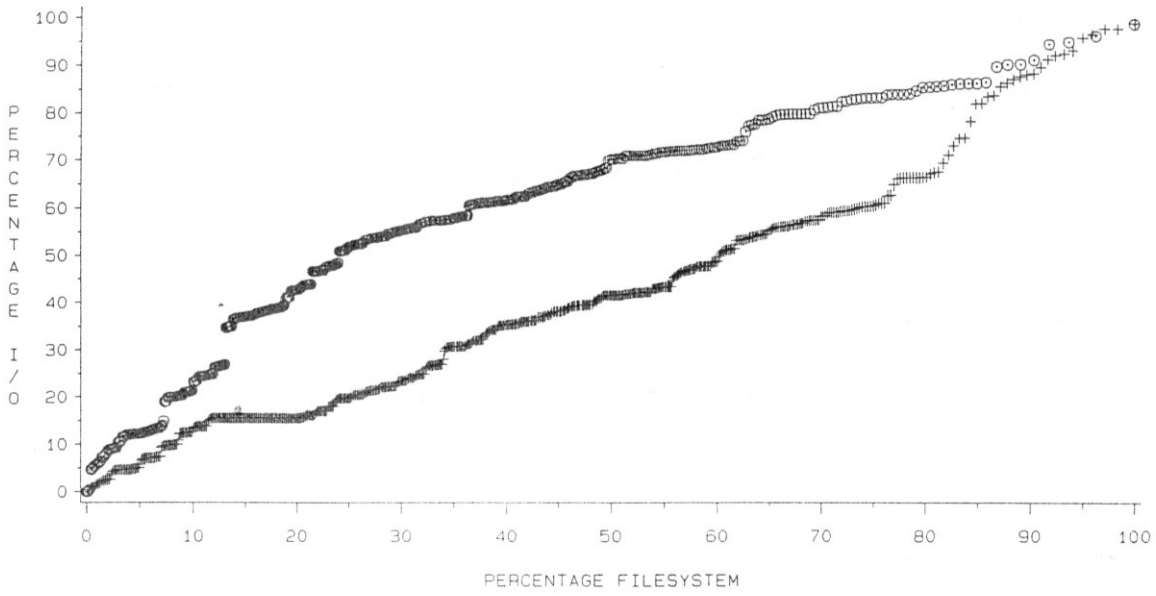
A third property we investigated was a possible linking between file temperature and file size. If there were a strong link between a file's size and it's temperature, then the replacement strategy would be relatively simple. However, it appears that any such linking is weak at best. In figure five we have plotted cumulative percent of file system space versus cumulative percent of total I/O's for both customers, with the files ranked by file size. If there were no correlation between the two variables then the curves should be relatively straight lines from the origin to (100,100), but the curves do not follow this pattern. One curve is slightly concave, which implies that the smaller files are hotter than the larger files, while the other curve doesn't follow any simple pattern. However, both curves exhibit similar behavior for the first ten percent of the file system, which consumes fifteen to twenty percent of the I/O activity. As a result, very small files seem to be moderately hotter than other files.

Another question was whether shared files tended to be hotter than non-shared files. Apparently, shared files are significantly hotter than other files, consuming roughly sixty percent of the total I/O's. In addition it seems that files which are shared by interactive users tend to be hotter than those files shared by batch jobs. While this distinction is important in the MVS world, there doesn't seem to be a corresponding distinction in the UNIX world.

In figure six we show the cumulative percentage of the total I/O versus the cumulative space used for all files, files shared by users, files shared by batch jobs, and non-shared files. Each curve represents the activity of a set of files, with the x-axis as the cumulative space used by the files expressed in terms of the total space consumed by all files, and with the y-axis as the cumulative I/O absorbed by the files relative to the total I/O to all files. The top curve, in the solid line, shows the cumulative percent of I/O versus the cumulative percent of file system space for all files, and is provided for reference. The second curve, with the plus signs, shows the curve for those files which are used by more than one user. In order, the remaining curves are for files used by more than one batch job, files used by more than one user and by more than one batch job, and files used by at most one batch job and user. Note that one may not simply add two curves together because it does not account for the space consumed by each file. Also note that the set of files shared by users and the set of files shared by batch jobs are not disjoint, and

CORRELATION BETWEEN FILE SIZE AND TEMPERATURE

CUMULATIVE %IOS VERSUS CUMULATIVE %FILESYSTEM
NON-VSAM



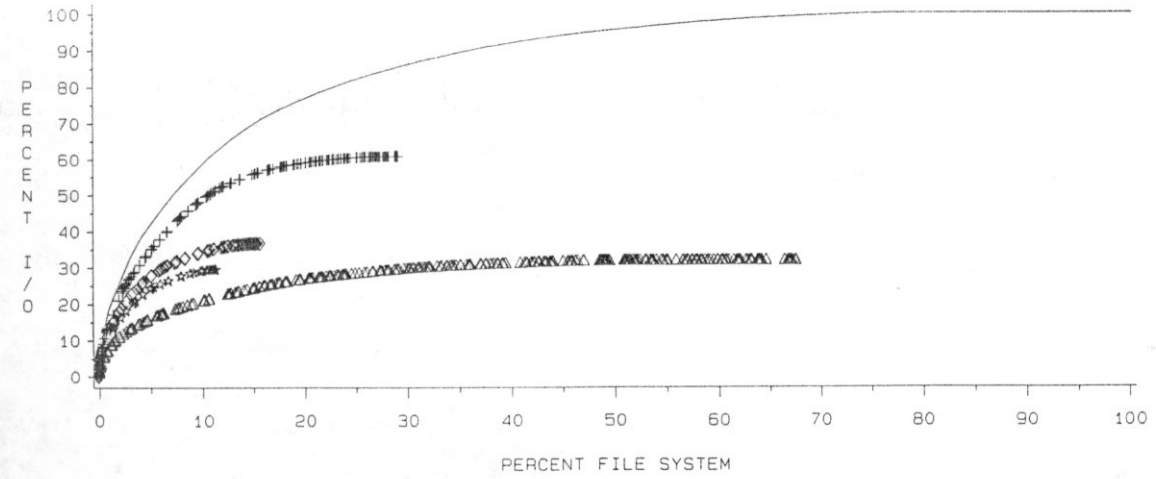
FILES ARE RANKED BY INCREASING SIZE

\+ CUSTOMER S
○ CUSTOMER Z

FIGURE 5

ANALYSIS OF FILE SHARING
%IOS VS %FILESYSTEM

CUSTOMER "S"
NON-VSAM



FILES ARE RANKED BY IO ACTIVITY PER TRACK FOR THE WEEK

\- ALL FILES
\+ FILES SHARED BY USERS
◇ FILES SHARED BY BATCH JOBS
★ FILES SHARED BY USERS AND BATCH JOBS
△ NON-SHARED FILES

FIGURE 6

that within each set files are ranked by descending temperature. Clearly, files which are shared by users are significantly hotter than unshared files.

Another property of interest was whether file opens are bursty. If the opens for each file occurred in bursts, then caching a whole file on the first open, and flushing it after the last close would drastically reduce the bandwidth needed for that file. However, as expected, things are a bit more complex than this simple model. It turns out that opens do tend to occur in bursts, but we can't provide a simple model for this phenomena.

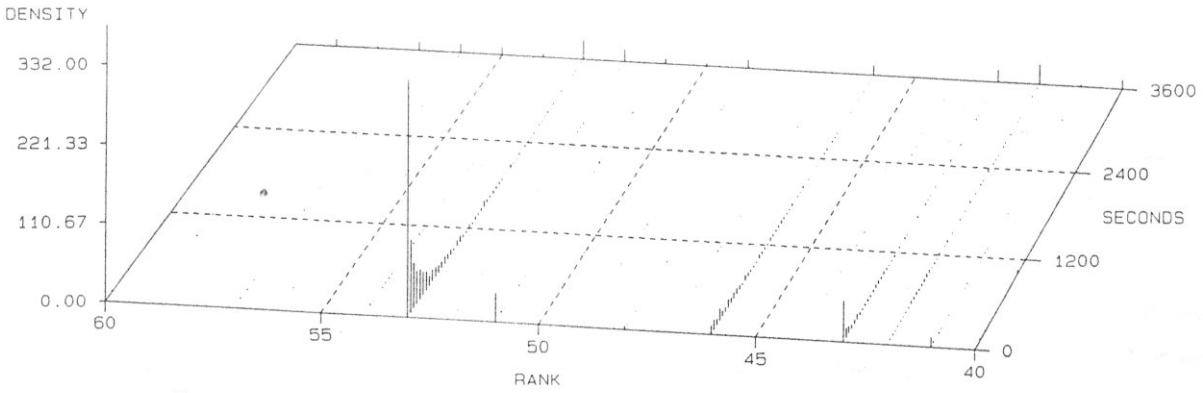
An interesting file property is the PDF for the time from a file's close to that file's next open. This could be used by the replacement strategy to flush files with a lower probability of being re-opened. Again, since we cannot observe PDFs, we approximated them using histograms. It turns out that some files exhibit exponential-like distributions, whereas others exhibit more uniform distributions. It would be a relatively simple matter to store some form of the PDF for the time to next open by storing a histogram of the times along with a total of the number of opens. Then the replacement strategy could compute the "probability of being accessed within some time frame" for each file every few minutes using the PDF, and rank the files according to that probability.

In figure seven we show the histograms for a handful of files. Figure seven is a three dimensional graph, similar to figure three, with axes "rank", "seconds" and "density". The axes "rank" and "density" have meanings similar to the axes in figure three. The axis "seconds" represents the number of seconds between each file close and the next open. A negative value would imply that the file was reopened before it had been closed. However, we discarded the negative values. In addition, the "seconds" axis has been clipped at one hour, and the density at one hour is really the probability density for all time greater than one hour.

Many files seem to have recognizable, or at least reasonable PDF's for the time from last close to next open. For example, files forty-three, and fifty-three seem to have exponential-like distributions. Unfortunately, many files, such as files fifty-one and fifty-two, exhibit more complex behavior. As a result, it seems that we should be able to predict with reasonable probability whether certain files are likely to be reopened.

TIME BETWEEN CLOSE AND OPEN
HISTOGRAM OF TIME BETWEEN OPENS

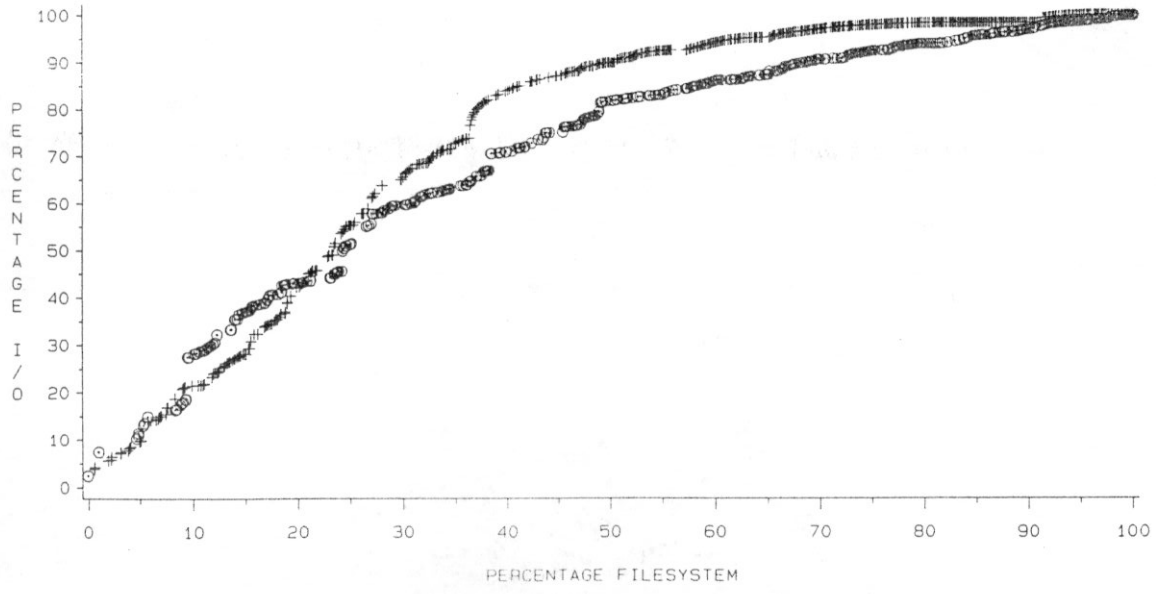
CUSTOMER "S"
NON-VSAM



CLIPPING VALUE FOR TIME: 3600 SECONDS
POSITIVE TIME VALUES ONLY
GRANULARITY OF 60 SECONDS
FIGURE 7

CORRELATION BETWEEN FILE OPEN FREQUENCY AND TEMPERATURE

CUMULATIVE %IOS VERSUS CUMULATIVE %FILESYSTEM
NON-VSAM



FILES ARE RANKED BY DECREASING NUMBER OF OPENS

+ CUSTOMER S
o CUSTOMER Z

FIGURE 8

A final question was whether file temperatures could be linked to file open frequency. The basic premise is that files which tend to be opened frequently should tend to have a higher temperature. It appears that this is in fact true, but the connection is weak compared to the linking between past long term temperature and future temperature. In figure eight we show the cumulative percentage of I/O versus the cumulative percentage of file system space for both customers with the files ranked by open frequency. There is some correlation because nearly ninety percent of the I/O went to fifty percent of the space, using this ranking. However, this is still a fairly weak correlation, and is nearly useless from a practical standpoint.

Conclusions

In conclusion we have presented our concept of a file cache and some results regarding basic file access patterns which may be useful for constructing an efficient file cache. We tried to find patterns which would allow us to predict file temperature, which in turn would help predict the value of promoting a given file. We also tried to find patterns which would help predict the time from a file's close until it's next open, in order to predict the value of keeping a file in the cache after the file is closed. We found that long term file temperatures are reasonable predictors of future long term temperature. In addition, we found that for many files the time from close to next open can be predicted using a probabilistic model based on past behavior with some reasonable success.

Acknowledgements

I would like to thank those who made this work possible. First of all, I would like to thank Amdahl Corporation for providing me with the opportunity and resources to investigate this area. I would also like to thank Dieter Gawlick, Dick Wilmot and Bill McCormack for their technical assistance and advice.

Bibliography

[BACLAWSKI87] Kenneth Baclawski, "A Stochastic Model of Data Access and Communication", *Technical Report NU-CCS-87-8*, Northeastern University, Boston, Mass., 1987

[BASTIAN81] A. L. Bastian, J. S. Hyde and W. E. Langstroth, "Characteristics of DASD Use", *Proceedings of the CMG 12th International Conference* (New Orleans, Dec., 1981), ACM, New York, pp. 107-109, 1981.

[BASTIAN82] A. L. Bastian, "Cached DASD Performance Prediction and Validation", *Proceedings of the CMG 13th International Conference* (San Diego, Dec., 1982), ACM, New York, pp. 174-177, 1982.

[DENNING68] Peter Denning, "The Working Set Model for Program Behavior", *Communications of the ACM*, Vol. 2, No. 5, May 1968.

[FLOYD86] Rick Floyd, "Directory Reference Patterns in a UNIX Environment", TR-179, Computer Science Department, University of Rochester, Rochester, New York, 1986.

[FRIEDMAN83] Mark Friedman, "DASD Access Patterns", *Proceedings of the 1983 CMG International Conference* (San Diego, Cal., Dec., 1983), ACM, New York, pp. 51-61, 1983.

[GROSSMAN85] Carol P. Grossman, "Cache-DASD Storage Design for Improving System Performance", IBM Systems Journal, Vol. 24, Nos. 3/4, pp. 316-334, 1985.

[HENLEY87] Martha Henley and Ingrid Liu, "Static Vs Dynamic Management of Consistently Very Active Data Sets", *Proceedings of the International Conference on Management and Performance Evaluation of Computer Systems* (Orlando, Fla., Dec. 7-11, 1987), The Computer Measurement Group, pp. 208-216, December 1987.

[HOWARD88] John Howard et al., "Scale and Performance in a Distributed File System", *ACM Transactions on Computer Systems*, Vol. 6, No. 1, pp. 51-81, February 1988.

[LAZOWSKA86] Edward Lazowska, John Zahorjan, and David Cheriton, "File Access Performance of Diskless Workstations", *ACM Transactions on Computer Systems*, Vol. 4, No. 3, pp. 238-268, August 1986.

[MCKUSIK84] Marshall McKusick, William Joy, Samuel Leffler and Robert Fabry, "A fast File System For UNIX", *ACM Transactions on Computer Systems*, Vol. 2, No. 3, pp. 181-197, August 1984.

[MENON88] Jai Menon and Mike Hartung, "The IBM 3990 Disk Cache", *Proceedings of the IEEE Compcon Spring 1988* (San Francisco, Cal., Feb. 29 - Mar. 4, 1988), pp. 146-151, 1988.

[NELSON88] Michail Nelson, Brent Welch, and John Ousterhout, "Caching in the Sprite Network File System", *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988.

[OUSTERHOUT85] John Ousterhout et al., "A Trace Driven Analysis of the UNIX 4.2 BSD File System", *Proceedings of the 10th ACM Symposium on Operating System Principles* (Orcas Island, Wash., Dec. 1-4, 1985), ACM, New York, pp. 15-24, 1985.

[PATTERSON88] David Patterson, Garth Gibson and Randy Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", SIGMOD International Conference on Management of Data (Chicago, Ill., June 1-3, 1988), ACM, pp. 109-116, 1988.

[SATYANARANAN81] M. Satyanaranan, "A Study of File Sizes and Functional Lifetimes", Proceedings of the 8th ACM Symposium on Operating System Principles (Pacific Grove, Cal., Dec. 14-16, 1981), pp. 96-108, December 1981.

[SCHROEDER85] Michael Schroeder, David Gifford, Roger Needham, "A Caching File System for a Programmer's Workstation", *Proceedings of the 10th ACM Symposium on Operating System Principles* (Orcas Island, Wash., Dec. 1-4, 1985), ACM, New York, pp. 25-34, 1985.

[SMITH78] Alan Smith, "Sequentiality and Prefetching in Database Systems", *ACM Transactions on Database Systems*, Vol. 3, No. 3, pp. 223-247, September 1978.

[SMITH81] Alan Smith, "Analysis of Long Term File Reference Patterns for Application to File Migration Algorithms", *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 4, pp. 403-416, 1981.

[SMITH85] Alan Smith, "Disk Cache - Miss Ratio Analysis and Design Considerations", *ACM Transactions on Computer Systems*, Vol. 3, No. 3, pp. 161-203, August 1985.

[THISQUEN86] Jean Thisquen, "6880 Cache Controller Feature and 6680 Electronic Direct Access Storage Planning and Tuning Guide", Publication MT-A01077-003, Amdahl, Sunnyvale, CA, 1986.

[ZIPF49] George K. Zipf, "Human Behavior and the Principle of Least Effort", Addison-Wesley Press Inc., Cambridge, Mass., 1949.