A trace-driven study of CMS file references

by G. P. Bozman
H. H. Ghannad
E. D. Weinberger

This paper presents a detailed study of file reference patterns by users of a VM/CMS interactive system. The data were collected from two different IBM locations via CMON, a CMS monitoring facility. We present background information about the CMS file system, the CMON program, and our data-reduction programs, as well as a discussion of the results. Some earlier studies of this type have been restricted to a static analysis of the existing files. However, as is shown in this paper, a static analysis does not reliably reflect dynamic file reference behavior. By using both static statistics and dynamic statistics, it is possible to better understand how file systems are used, to evaluate possible changes, and to provide distribution parameters for modeling. More recent studies of other interactive systems have measured dynamic activity patterns. We compare our results with these when appropriate.

Introduction

One of the important performance-sensitive components of any operating system is the file system. Any operating system, whether general- or special-purpose, must provide an efficient way of storing, retrieving, and maintaining files in secondary storage. The importance of the file system has increased because the availability of high-capacity direct-access storage devices (DASDs) now enables diverse populations of users to store data such as programs, electronic mail, graphic data, drafts of technical papers, and even books. This diversity of file usage must be understood by computer systems capacity planners, designers of new operating systems, and those interested in improving the performance of existing systems.

Understanding the workload imposed on a file system in an interactive environment was the main objective of the study described in this paper. A secondary goal was to investigate whether the load patterns could be generated synthetically in order to simulate file systems with larger user loads than we were able to trace.

Some earlier studies of this type [1, 2] have been restricted to a static analysis of the existing files. However, as is shown in this paper, a static analysis does not reliably reflect dynamic file reference behavior. By using both static and dynamic statistics, it is possible to better understand how file systems are used, evaluate possible changes, and provide distribution parameters for modeling. More recent studies of other interactive systems [3–5] have measured dynamic activity patterns. We compare our results with these when appropriate.

Workload studies can be done at several different levels depending on the objective of the study. At the highest level, an application program designer might be interested in understanding the relative frequency of the commands issued by the users of the program—for example, the editing commands issued by a user during an editing session. At the other extreme, a hardware designer might be interested in the frequency of occurrence of each type of machine instruction, such as loading a register from main storage or comparing the contents of one register to another. Our study fell somewhere between: the

commands seen at the interface to the file system. Such commands typically involve opening, closing, reading, writing, erasing, and renaming files, and searching directories for the existence of a file or files.

This study was first done at the IBM Thomas J. Watson Research Center in Hawthorne, New York, and was later repeated more extensively at the IBM Kingston Programming Center. The nature of the systems involved, in terms of the user population, is described later in this paper. We compare and contrast the results from the two systems where appropriate.

A synopsis of our findings is as follows:

- File system activity is bursty and is dominated by reading.
- Most files are read/written sequentially.
- Most read and write activity is for small amounts of data, but access to large files accounts for most of the bytes transferred.
- All of the above characteristics are similar to results found in studies of other file systems, primarily UNIX™.

The VM/370 operating system

The VM/370 system consists of two major components. The control program, or CP, is the manager of the real resources of the system. It contains the scheduler, dispatcher, device management, real storage management, accounting, security, and recovery functions. It provides each logged-on user with a *virtual machine* that is identical in architecture to the real hardware on which CP runs. The Conversational Monitor System, CMS, runs in each virtual machine and provides a command processor, a file system, and a user execution environment.

- *How the control program (CP) handles file activity*

The purpose of the control program is to give each user the appearance of having a full System/370™ CPU with a device configuration at its disposal. From the standpoint of the file system, the most important function CP performs is to divide real DASD volumes into an arbitrary number of *minidisks*. The minidisks consist of a range of contiguous cylinders on a real volume. By adjusting I/O requests issued by the virtual machine, CP ensures that each minidisk appears to begin with a cylinder (Count Key Data DASD) or block (Fixed-Block Architecture DASD) numbered 0 and extends for the appropriate number of cylinders or blocks. CP also restricts access to each minidisk to authorized users.

- *CMS operating system*

CMS provides an environment for running commands and user programs. CMS commands are merely programs stored as executable images, either in primary storage or in the file system. In addition to command resolution and a file system, CMS also provides console support, virtual storage management, and virtual device support.

- *CMS file system*

The files and directories, which are managed by the CMS file system, are stored in the minidisks that are provided by CP. The user identifies a minidisk to be ACCESSed[1] via a mode letter, which is a single alphabetic character. The mode letter "A" usually designates the user's primary (read/write-accessible) work area; the mode letters "S" and "Y" designate the system minidisks (that store programs supplied by IBM). Many installations add other minidisks that contain program libraries of interest to large groups of users. This was the case in both systems that we studied. The "S," "Y," and other local minidisks of this type are ACCESSed read-only and collectively provide systems routines and programs of general interest such as compilers and electronic mail tools.

In effect, each user has his or her own instance of a file system that has access to its own DASD space. The file-sharing capabilities of the CMS system are limited by the fact that the file system runs in the user's virtual machine, allowing each user to change the content or state of his own file system at any time.[2] The major problem is that the file directory and allocation bit map are maintained in the virtual memory of each user who has ACCESSed the minidisk. This aspect of CMS effectively limits file sharing to the situation in which many virtual machines can ACCESS the same minidisk in read-only mode, but at most one virtual machine can ACCESS the minidisk in read/write mode. File sharing at the read/write level is somewhat problematic with the traditional minidisk file system and is not often done.

Objectives

Our primary objective was to understand the load imposed on a file system in an interactive time-sharing environment. Some specific questions we had upon undertaking this study were the following:

1. What is the relative rate for each type of file activity?
2. What is the size distribution of file accesses?
3. What is the frequency of accesses to the same file?
4. What is the distribution of interarrival times for file system requests?
5. What are the static and dynamic distributions of file sizes?

In addition, we wanted to explore the feasibility of generating synthetic file system workloads so that, for

[1] The ACCESS command in CMS connects the minidisk to a *mode* letter which is in a search path. Additionally, it reads the file directory and, if the user has write capability, the allocation bit map into the user's virtual memory. In this paper we *capitalize* ACCESS when we are referring to the CMS command.
[2] The Shared File System (SFS) available in CMS SP/6 provides file sharing and other new facilities through a file server. The traditional minidisk file system remains accessible.

example, systems with very large user populations might be modeled.

The next several sections deal with data collection. Readers not interested in this aspect of the study may skip to the section on analysis of the results.

Data collection

- *Static data collection*

For the Hawthorne study, we wrote a program which was invoked when a user logged on and recorded the file sizes for all of the minidisks which the user had ACCESSed at that time. This missed some of the minidisks which were dynamically ACCESSed during a user's session but, nevertheless, gave us a large and representative sample of the static file size distribution on the system. We also made a simple modification to CP so that we could uniquely identify each minidisk. Thus, the files on minidisks ACCESSed by more than one user were only counted once.

Because this program took up to several minutes to execute for users who had a large number of files ACCESSed, we refined this procedure for the Kingston study. At Kingston we identified beforehand most of the minidisks which were shared among users, and modified the program to avoid repetitively collecting statistics for these minidisks. Instead we ACCESSed these minidisks during one of our own sessions and ran the program once.

- *Dynamic data collection*

VM/370 has a monitoring capability in the CP component of the operating system. This monitor provides information about the operation of the system and about the state and resource consumption of each virtual machine. Unfortunately, it is not feasible for the monitor to gain knowledge of what is happening within a virtual machine. The monitor also lacks information about the current state of file directories and has no allocation maps from which to infer the meaning of file system operations based on the I/O operations to minidisks. Thus, while it would seem that CP would be a logical central point from which to collect file system data, a collector that interfaces directly with CP is not practicable.

- *CMS monitor (CMON)*

CMON is a CMS monitor, developed by David N. Smith of the IBM Thomas J. Watson Research Center, that allows the collection of information about the activities within CMS for a particular user. It collects trace information about specific events within the CMS machine being monitored. For this study CMON was extended to provide information at the occurrence of each logical I/O operation as well as during other lower-frequency operations such as OPEN, CLOSE, and STATE (which determines the existence of a file with a given identifier on a given minidisk). We could therefore determine the actual number of bytes read or written by each access.

- *Data-reduction programs*

The data reduction of both the static data and the (dynamic) CMON user traces was done by a set of Pascal programs written for this purpose. The reduction of the CMON data was more involved than one might think. CMON monitored exactly what the file system did at the file access level, which means that the implementation details of CMS functions could affect the statistics that were gathered.

A good example of such a detail is the process by which files are updated via certain text editors (e.g., XEDIT). These editors copy the file to virtual memory, where the file undergoes modification by the user. At the end of the editing session, in order to perform an *atomic* update, the modified version is written to a temporary disk file, the previous, unmodified version of the file is erased, and the temporary file is renamed appropriately. To collect data on *generic* file system activity, the reduction program must realize that all writes to the temporary file are actually writes to the file being edited.

Difficulties such as this were overcome by searching for these known patterns of file system activity as a part of the first-level data-reduction program. For example, a "write a new file" event that results from editing a previously existing file was distinguished from the genuine creation of a new file by looking for a characteristic "write/erase/rename" pattern of file activity. The output of the first-level data-reduction program is a report of a single user's generic file system activity by minidisk.

We also wrote several other Pascal and REXX programs to further reduce data collected from individual users, to obtain cumulative statistics, and to answer specific questions that came up during the study.

Selected systems and users

The monitoring of the two systems that were used for this study was done about eight months apart. The first system which was studied (Hawthorne) was the one that the authors used daily. It was used for a variety of work such as program development and testing, preparing papers for publication, and sending and receiving electronic mail.

Twenty-one users were selected randomly from the directory of enrolled users. The users selected had to be contacted individually, since they had to agree to the installation of CMON; this was the factor most responsible for the small size of the sample.

Although we had no good *a priori* reason for believing that a sample of 21 users would be adequate, we employed several *a posteriori* techniques to verify that our data were representative. A visual inspection of the traces revealed

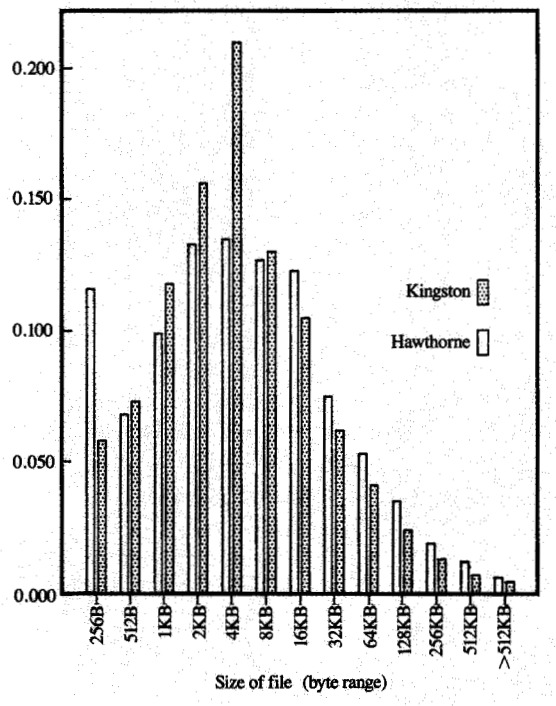

Figure 1

Static distribution: fraction of all files by file size.

usage patterns that seemed to be representative on the basis of our prior experience with VM. Comparing our findings with those of a previous, less detailed study [6] performed on the IBM Yorktown system, we noted excellent agreement in every case where comparison was possible. We computed the coefficient of variation (the ratio of the standard deviation to the mean) for each of the statistics we gathered, and found that this ratio was quite small (typically 0.1). For the statistic with the largest coefficient of variation, we computed the 95% confidence intervals assuming a student's T distribution, and discovered that they were well within one standard deviation.

The resulting user population was composed of a wide variety of user types, including a secretary, a new user, an expert user working hard to meet a software development deadline, and three service machines. (A service machine is a virtual machine that is not directly associated with a user but provides function for a set of users via communication among virtual machines.) After the users agreed to participate, they were monitored until we obtained traces for a day that they thought was typical for them. This was usually the first day of monitoring, but occasionally an unanticipated interruption would create an atypical session.

The selection of the users and system in the second case (Kingston) was done differently, since we had the experience of a preliminary study. To better determine the sample size, we used the largest coefficient of variation from the first study to compute the sample size for the second study. We used a confidence interval method and calculated the sample size (50) which gave us 95% confidence that our mean would fall within one percent of the true mean. Since we did the confidence interval analysis with the largest coefficient of variation, the above confidence level would also be a lower bound with all of the other statistics we observed from the first study. However, the sample size (number of users to be monitored) that we computed was valid only if the two populations were the same, or if all of the coefficients of variation which would be observed in the second study remained smaller than the largest from the first. Of course, we did not know this, and therefore we wanted to get a larger sample, if possible, as a contingency.

The IBM Kingston location, which was the site of our second study, had 22 VM systems. We were especially interested in a VM system that supported PROFS, which is an IBM office-system product in wide use. Since many VM systems are dedicated to running PROFS, we were interested in discovering how it might affect file system load.

We were able to select a system in Kingston that had significant PROFS use. We obtained two weeks of accounting data from this system in order to ensure that our sample was selected from the set of users who used the system on a regular basis. We then randomly selected users and were able to get 60 acceptable volunteers.

Preparation and problems encountered

The programs required for the study resided on a minidisk of a virtual machine that we maintained. This minidisk was ACCESSed by the users in the study when they logged on, and activity traces were begun at that time. The trace was terminated when the user logged off. Because of this, we advised users to disconnect[3] instead of logging off when they were discontinuing their sessions for short periods (e.g., for lunch or a meeting).

At Hawthorne, once users were set up for monitoring, we received traces until we requested them to delete our initialization command from their log-on procedure. At Kingston, we enhanced our routines to enable us to add and delete users to be monitored without their awareness, as long as they had agreed to participate in our study and had installed an EXEC[4] which we had sent them.

[3] This process disconnects the user's terminal from the system without terminating the session.

[4] EXECs are command procedures that are very frequently used to package together other programs with conditional execution based on run-time parameters. In CMS there are three different EXEC interpreters: EXEC, EXEC2, and REXX.

We encountered no major problem in gathering data from users in Hawthorne. However, this was not the case at Kingston, where we encountered two problems that we had not anticipated. The I/O interrupt vector used by CMON was used in an incompatible manner by a version of APL and by a compress/decompress routine that was optionally invoked by some users to compress their files upon writing them onto their minidisks and decompress them when they were read. Neither of these programs had been used at Hawthorne. We therefore lost the subset of our volunteers who used either of these tools.

When we started to reduce the trace data, we found that some of the volunteers did not log on at all (or for only very short periods), for unexpected business or personal reasons. Because of these problems, we ended up with 45 acceptable sessions from 34 users.

After reducing all the data and using the largest coefficient of variation, we repeated the confidence interval analysis and were able to conclude that our 95% confidence level remained valid with this set of sessions.

Analysis of the results

The amount of data that was gathered was very large, and, depending upon one's objectives, could be reduced and presented in many different ways. Here, we present results which we believe are of general interest.

- *Static data*

We were interested in gathering static statistics (i.e., statistics about the CMS files as they reside on disks). Static statistics provided us with information on the mean file size and what percentage of secondary storage is occupied by files whose size falls within a given range.

We observed that, in general, the data from the two systems correspond. **Figure 1** depicts the distribution of the number of files by file size for the two systems. We broke up the range of possible sizes into bins bounded above by successive powers of two. (The first bin was associated with accesses of 256 bytes or less, the next bin was associated with accesses of 257–512 bytes, the next with accesses of 513–1024 bytes, etc.) Therefore, the horizontal axis is the range of size in bytes by bin. The number below each bar represents the upper limit for the size, and the exclusive lower limit is the number below the previous bar. The vertical axis is the ratio of the number of files of a given size to the total number of files counted (but not all files) on each system.

For both systems, the median file size was less than four kilobytes (KB) which is the recommended block size for minidisks (for I/O performance).[5] This is similar to the distribution found on the TOPS-10 system by Satyanarayanan [2], where 50% of the files were less than 2880 bytes (five blocks of 128 36-bit words each).

[5] Four kilobytes is the largest block size supported by the CMS file system.

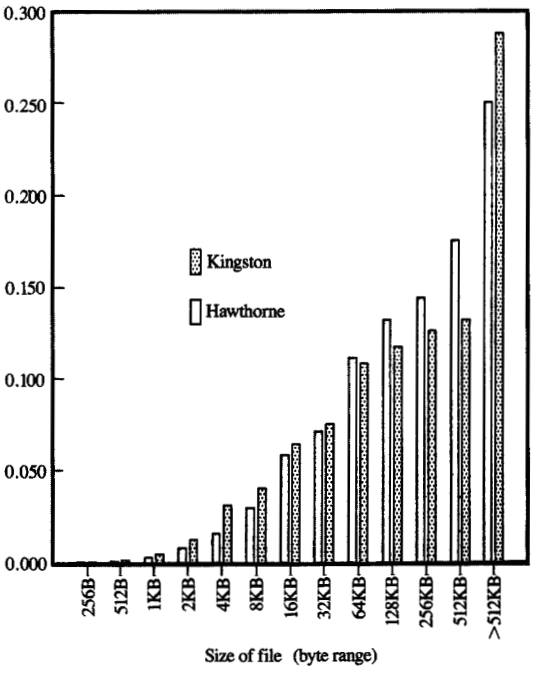

Figure 2

Static distribution: fraction of storage utilization by file size.

For many of the user minidisks which we examined, the median was about half of that, or about 2 KB. The mean file size was considerably larger than the median in both systems, indicating that the relatively few very large files occupied a significant percentage of the allocated secondary storage. The overall mean static file size was 24 409 bytes. This distribution is comparable to that found in [2], although the exact mean is not available for comparison.

The distribution of the percentage of storage occupied versus file sizes is shown in **Figure 2**. Although at 4KB blocking there is a considerable amount of *wasted* space in the large number of small files, the space utilization is so dominated by the small number of large files that, overall, the wasted space is only about 10% of the total space used.

The static data also allowed us to investigate any correlation between the static distribution of file sizes as they reside on the minidisks and the dynamic activity distributions such as the size of the read accesses.

In **Figures 3** and **4** the fraction of files for each size range is compared to the fraction of files read in the same range. The correlation is strong for the Hawthorne data;

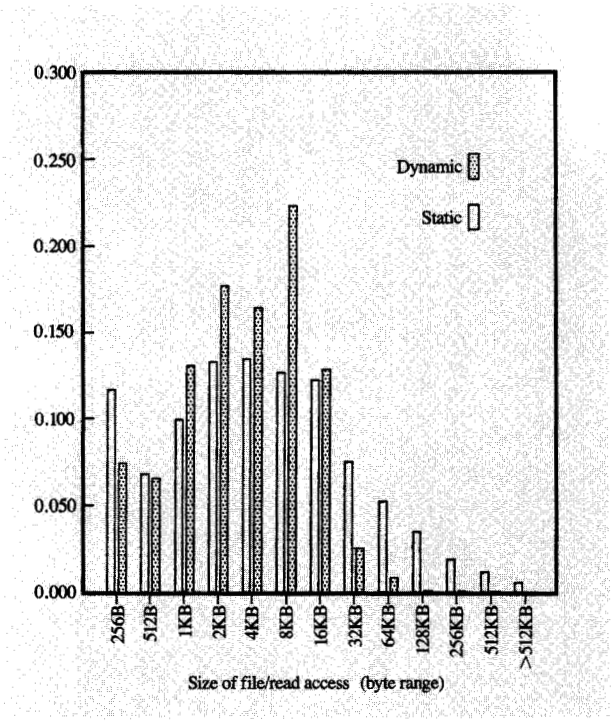

Figure 3

Hawthorne: static fraction of all files by file size vs. dynamic fraction of total files read by read event size.

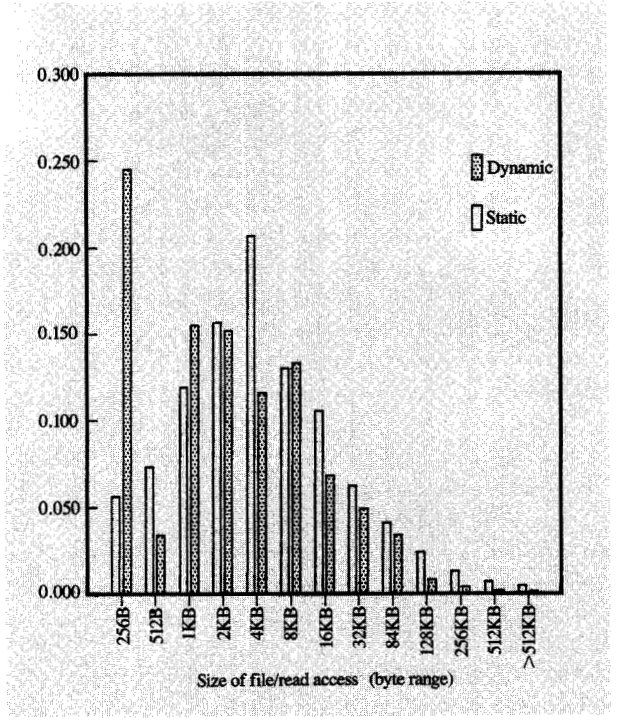

Figure 4

Kingston: static fraction of all files by file size vs. dynamic fraction of total files read by read event size.

the coefficient of correlation is 0.90. The correlation for the Kingston data is not as strong; the coefficient of correlation is 0.60. However, if the range 1–256 bytes (smallest files) is excluded, the coefficient of correlation is 0.89.

Figures 5 and **6** compare the fraction of total storage occupied to the fraction of total bytes read for each size range. Here the coefficients of correlation are −0.22 for Hawthorne and 0.60 for Kingston. We investigated why there is more space being used by large files than being read dynamically. Many of the large files are really libraries of macros (MACLIBs) and run-time routines (TXTLIB and LOADLIB). This is an anomaly of the way in which the CMS file system supports MVS libraries. All of the members of these libraries are placed in one file along with an imbedded directory. When they are invoked by a compiler or loader, typically only a small subset of the members are actually read. Therefore, a read access to one of these very large files will appear as a partial read of significantly fewer bytes than the file size.

• *Dynamic file system activities*

Dynamic file system activities are the results of application requests to the file system. In addition to reading and writing files, they include directory searches, directory listings, and erasing and renaming files.

File read events

We discovered that most user commands caused a surprising number of files to be read in a relatively short burst of activity. We knew, of course, that the common use of EXECs often caused more file reading. The surprise was how much the EXECs were nested with other EXECs and programs. The *tool-builders* had used modular techniques to the extent that commands that we anticipated might read a few files would read 10 to 20. Even experienced CMS users were surprised when we showed them traces of their own activity. Burstiness was also seen on the UNIX systems studied by Ousterhout et al. [5] and Floyd [4].

On both systems, most files were read in their entirety and sequentially. This was true for 86% of the files at Hawthorne and 88% at Kingston. This is greater than the results found by Floyd [4] and Ousterhout et al. [5] on UNIX systems, where 68% and 67% of the files opened for reading were completely read. Most of the nonsequential accesses in CMS were almost sequential in the sense that only a few records were read out of order or more than

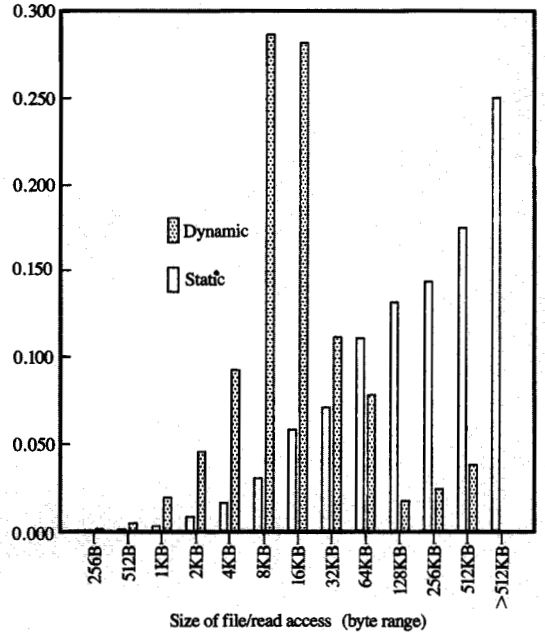

Figure 5

Hawthorne: static fraction of storage utilization by file size vs. dynamic fraction of total bytes read by read event size.

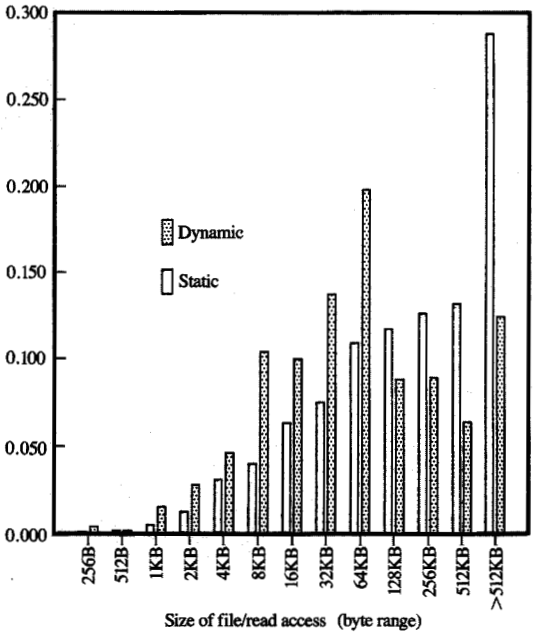

Figure 6

Kingston: static fraction of storage utilization by file size vs. dynamic fraction of total bytes read by read event size.

once. This situation arises frequently in the processing of EXECs. The three CMS EXEC language processors [EXEC, EXEC2, and REXX] are called in a fixed order to identify the language in which the EXEC is written by reading the first record of the file. If the wrong command processor gets the EXEC file first, the command processor will terminate without closing the file. Thus, the first record of the file is often read more than once, but the rest of the file is read sequentially. The sequential nature of reading on CMS is largely due to compilers, EXEC processors, text processors, and editors reading the entire file into virtual memory. However, we note that the CMS file system does not offer an indexed access method.

It is important to observe that some of the frequently used tools in CMS (most notably the most popular editor, browser, and text processor) exist in shared memory, and their use is not reflected as file system activity. This tends to understate CMS file activity in comparison with a system that must access all data via the file system.

Table 1 shows read event statistics for both systems. (A read event is the read activity of any file from open to close.) The first row is the total number of reads observed. The values from the two systems cannot be compared, since the number of users was different. The second row is this value normalized to the average number of read events per user per session. The third row is the standard deviation of user activity. In order to discount differences in session length (e.g., the sessions of two users at Hawthorne were much longer than average), we also normalized for time. The fourth row shows the average number of read events per user per hour. The Hawthorne users read 48% more files per session and 21% more per hour. This was largely a result of the following:

Table 1 File read events.

	Hawthorne	*Kingston*
Total	21 861	31 692
Average per user	1041.00	704.77
Standard deviation	834.43	665.61
Average per user per hour	149.28	123.19
Percent of all events	7.94	11.65
Bytes read per user per hour	1.42 MB	1.00 MB

1. At Hawthorne, two very active users accounted for a disproportionate share of the file system activity (e.g., they read 43% of the total bytes accessed). These two

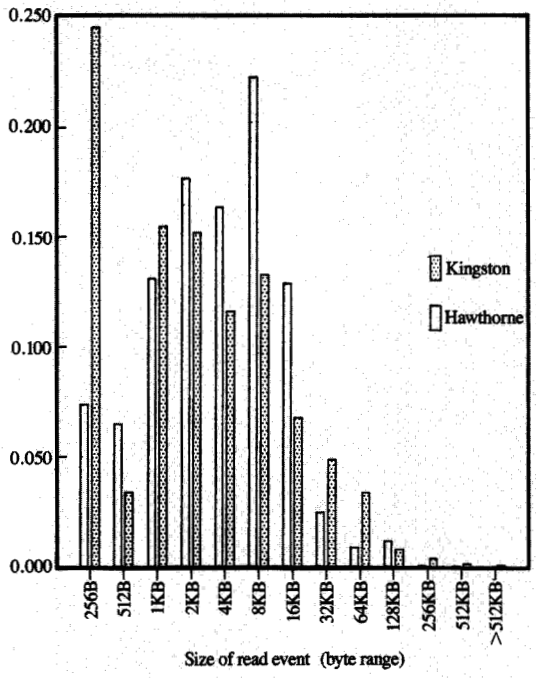

Figure 7

Fraction of total files read by read event size.

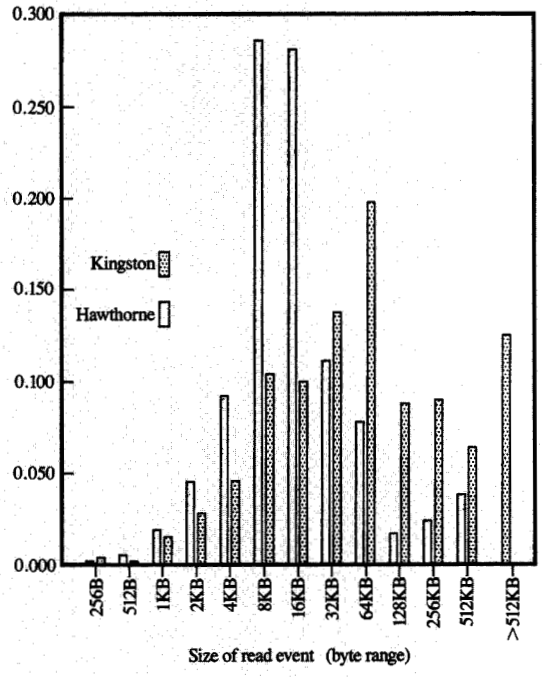

Figure 8

Fraction of total bytes read by read event size.

users were each developing a (different) large software system and compiling a large number of programs. This dominance of activity by a small subset of the active users seems to be typical of the Research Center distribution: A sample of daily accounting data showed that 1.3% of users performed 30% of the I/O events. While a similar distribution was seen at Kingston, the traces were not dominated as much by a few very active users.

2. To a greater extent than their counterparts in Kingston, the tool builders at Hawthorne had developed tools from many smaller, modular components. This resulted in long bursts of read activity as the result of a single user command. We almost always had difficulty moving Hawthorne tools to Kingston, because there were usually EXECs and programs being called that were not present on the Kingston system.

The fifth row shows the percentage of all file system events that were read events. We later discuss the overall file system event distribution further.

The last row of Table 1 shows the number of bytes read per user per hour. This indicates that in addition to reading more files per hour, the users at Hawthorne also read somewhat larger files on average (9741 bytes per read access at Hawthorne and 8312 bytes per read access at Kingston). This too, was largely accounted for by the dominance of the two active users there. Also, we noticed a greater amount of text processing (involving rather large files) at Hawthorne. **Figure 7** shows the relative fraction of read accesses by the size of the access. For example, very small read accesses (up to 256 bytes) accounted for only 7.4% of the total file accesses at Hawthorne, but over three times that amount at Kingston.

Although most of the read activities involved a small number of bytes (4 KB or less), a large percentage of bytes read per hour were done by read requests that involved large sizes. **Figure 8** shows the distribution of bytes accessed per hour by a given access size.

The read event/write event ratio was 9.4 at Hawthorne and 10.2 at Kingston. The bytes read/bytes written ratio was 5.2 at Hawthorne and 7.4 at Kingston. The 4 KB block read/write ratio was 4.9 at Hawthorne and 7.3 at Kingston. This is very similar to Floyd's byte read/write ratio on UNIX of 5.25 [4].

We also found that, at both sites, approximately 50% of the files that were read, were read more than once. Approximately 90% of all read requests were done to the

Table 2 File write events.

	Hawthorne	*Kingston*
Total	3029	3133
Average per user	144.24	69.62
Standard deviation	122.19	98.39
Average per user per hour	15.92	11.92
Percent of all events	1.10	1.15
Bytes written per user per hour	274 KB	135 KB

Table 3 File write events by event type.

	Hawthorne	*Kingston*
Average per user per hour	15.92	11.92
UPDATE	8.61	4.85
REPLACE	1.59	1.63
NEW	5.71	5.44
Bytes written per user per hour	274 KB	135 KB
UPDATE	8 KB (3%)	22 KB (17%)
REPLACE	47 KB (16%)	32 KB (24%)
NEW	219 KB (82%)	80 KB (60%)

above files. This indicated that a data cache would be beneficial, and a later study confirmed that this was indeed the case [7].

File write events

The statistics for file write events are given in **Table 2**. The Hawthorne users were more active writers as well as readers. The average number of bytes written per access at Hawthorne was 17 624, compared with 11 597 bytes written per access at Kingston. This was also largely a result of the dominance of two active users at Hawthorne.

Table 3 shows the write event comparison broken down by the type of write event. An update event involved the replacement of a part but not all of the file or, more commonly, an append to the end of a file. A replace event involved the replacement of the entire file due to an edit or copy operation (editors in CMS read the entire file into memory and replace it in its entirety on a *file* or *save* operation). A new event involved the creation of a file.

The lifetime distribution of new files is shown in **Figure 9**. This graph shows the cumulative fraction of files created during a session that were either erased or replaced in their entirety. Recall that CMS editors replace the entire file. New files in CMS have a somewhat longer lifetime than those found in [5], where 80% of the new files in UNIX environments were deleted or replaced in less than 200 seconds. In our study less than 60% of the new files were erased or replaced within 200 seconds.

Figure 10 shows the same distribution at the block level. A greater fraction of the space used by new files is being

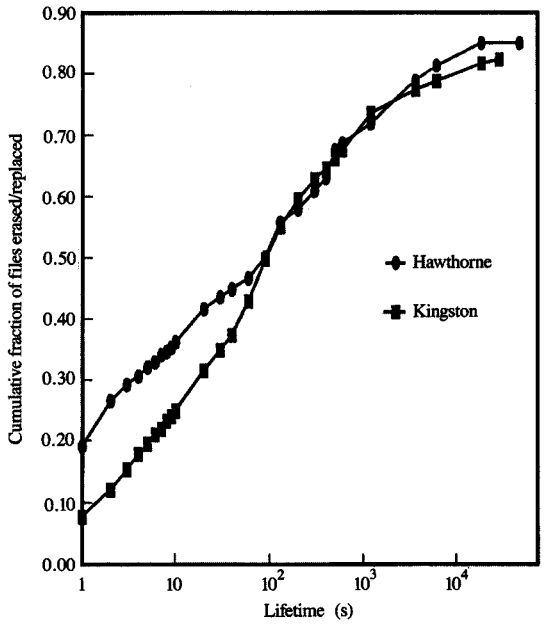

Figure 9

Fraction of files erased/replaced as a function of lifetime.

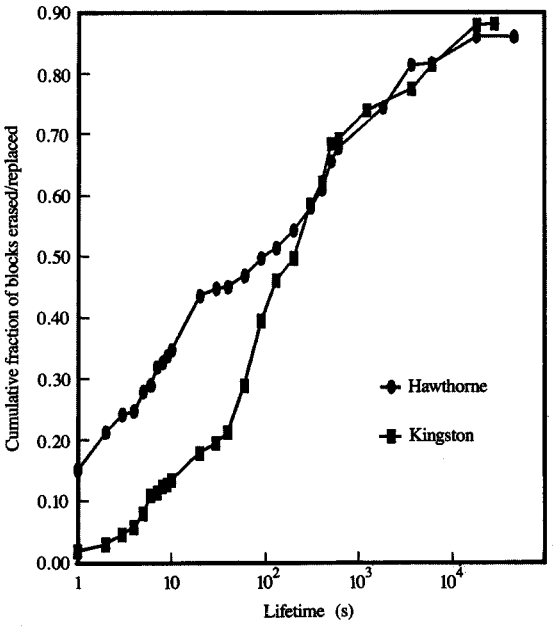

Figure 10

Fraction of written blocks (excluding append writes) erased/replaced as a function of lifetime.

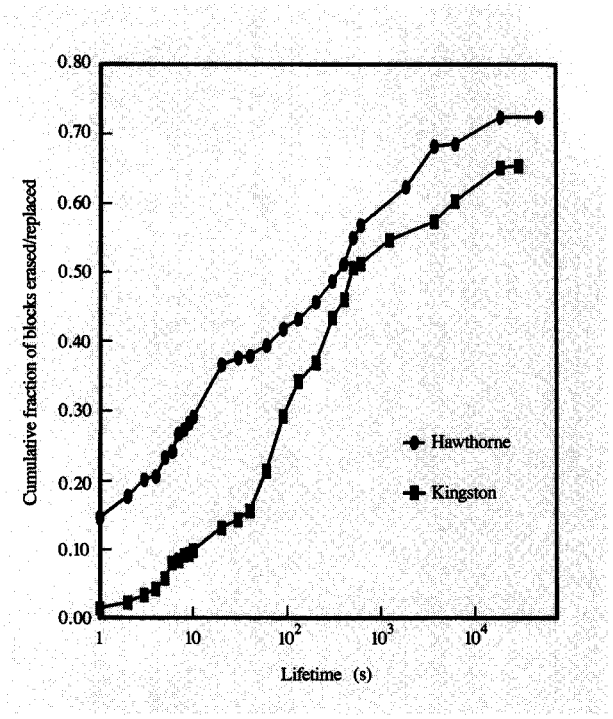

Figure 11

Fraction of all written blocks erased/replaced as a function of lifetime.

released than the files themselves. **Figure 11** is the same as Figure 10, except that all written blocks are considered. That is, blocks appended to existing files are included as well as blocks written as new or totally replaced files. This is of interest when considering the upper bound of I/O activity that a cache can avoid [8]. As shown here, in the CMS environment, an *infinite* cache could avoid 96% of I/O activity (at the block level) in Kingston and 95% at Hawthorne (100% of reads and the fraction of replaced or erased blocks adjusted by the read/write ratio).

Files were also generally written sequentially. This was true of 93% of the files at Hawthorne and 97% at Kingston. We considered append writes that extended a file to be sequential. This is very close to the UNIX data in [5], where 97% of the write-only accesses were sequential.

In both systems the locality of reference of file writes was nearly the same. In both cases, a little less than 30% of the files were written more than once, and approximately 70% of the write requests were made to those files.

Figure 12 shows the ratio of write events by a given byte size range to all write events. It corresponds to Figure 7 for read events. The Kingston and Hawthorne distributions are similar, and in both cases a very large number of accesses involved a small number of bytes. At Hawthorne 58% of the write events involved 256 bytes or less. At Kingston this was true of 37% of the write events. We found that this was largely due to the use, in CMS, of 1) small files to keep track of the state of many variables associated with user sessions (e.g., whether they have perused the latest bulletin board articles) and 2) the IBM VNET internal network for file and mail transfer. Sending and receiving network files causes a short entry to be appended to an activity log on the user's private minidisk.

Figure 13 shows a comparison of the distribution of bytes written per hour by the size of the write event. A very large percentage of bytes written per hour involved large files, although a majority of write events were 4 KB or less in size. This was significantly different from the corresponding read distribution shown in Figure 8. We found that these very large files were largely APL workspaces, compiler work files, and program listings at Hawthorne. This type of file was also present at Kingston, but the largest files were the temporary work files of an application program unique to that environment.

In summary, most file access events transferred a relatively small number of bytes, while most of the bytes transferred were in accesses associated with larger files. At Hawthorne 61% of the read events per hour were for 4 KB or less, but 84% of the bytes read per hour were in events that transferred more than 4 KB. At Kingston 70% of the read events per hour were for 4 KB or less, but 91% of the bytes read per hour were in events that transferred more than 4 KB. There was an even greater difference in the write distribution. At Hawthorne 83% of the write events per hour were transfers of 4 KB or less, while 98% of the bytes transferred per hour were in events that transferred more than 4 KB. At Kingston 70% of the write events per hour involved transfers of 4 KB or less, while 96% of the bytes transferred per hour were in events that transferred more than 4 KB. Similar distributions were found in UNIX [5], where 80% of all file accesses were to files less than 10 KB, while 70% of all bytes transferred involved files larger than 10 KB, and [4] where 75% of all opened files were under 4 KB but 67% of all bytes were read from files more than 20 000 bytes long. Bach and Gomes [3] also found a similar event distribution in UNIX, where 75% of the open files were smaller than 4 KB, but they did not measure the bytes-accessed distribution.

Summary of all file system activities

One of our objectives was determining CMS logical file activity rates. These rates are shown in **Table 4** for all file system activities. Write events are listed by type. This table shows the rate of all file system commands.

FULIST and FILELIST are full-screen file directory presentation programs that are very popular. Many users

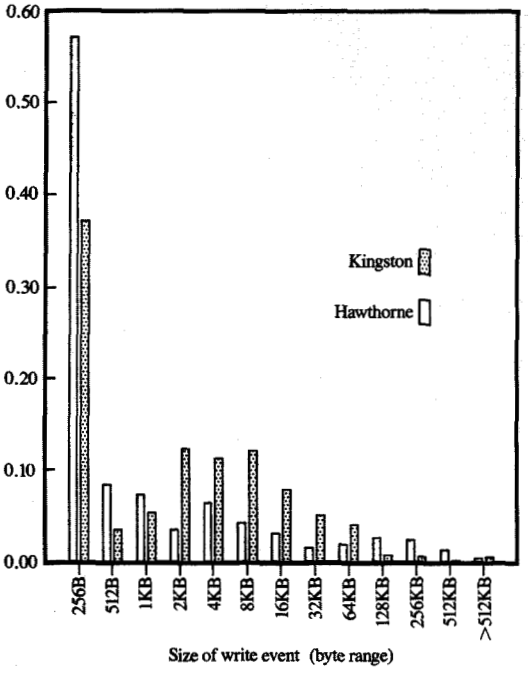

Figure 12

Fraction of total files written by write event size.

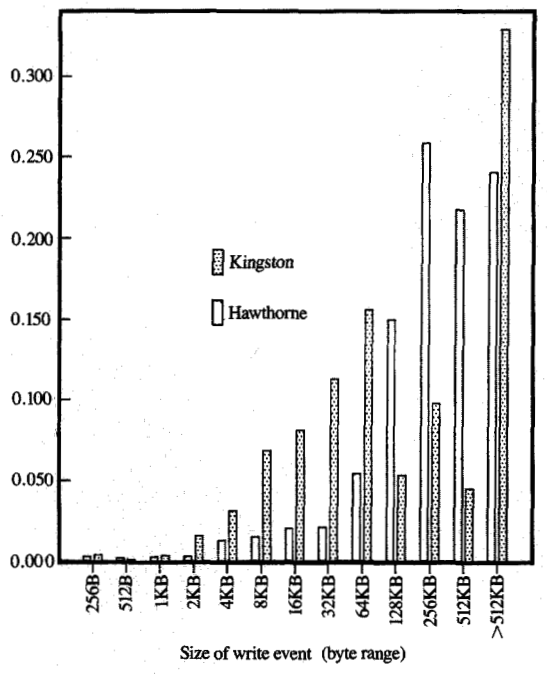

Figure 13

Fraction of total bytes written by write event size.

remain in this mode for extended periods of time. We were interested in the frequency of these commands because, when issued, they read entire file directories.

The STATE command, which is the CMS directory search primitive, dominates at both sites. We incremented this count for each directory searched (i.e., looking for file QUEENS PASCAL globally with ten minidisks ACCESSed would result in an increment of 10 if not found, and, for example, 5 if found on the fifth minidisk searched). The larger number at Hawthorne is due to a larger number of minidisks being ACCESSed on average. The special case of STATE for a writable minidisk is used by programs such as editors and compilers to determine where to write output.

Distribution of times between file system events

It is also of interest for modeling purposes to know the distribution of times between file system events (the interarrival time distribution). This distribution was even more repeatable than the previous distributions. It showed a three-part structure: a sharp peak between 10 and 100 ms, a time of the order of a disk access time, another peak between 1 and 20 s, a time of the order of an active user's response time, and a slow decay thereafter, reflecting the distribution of users becoming active after a period of inactivity (e.g., responding to the sudden arrival of electronic mail). However, since two of these three parts are primarily determined by user interaction, the interarrival times can be divided into two groups. The first is primarily due to the "burst" effect of file accesses after a user enters a command such as a compilation which causes multiple file references. The second group is

Table 4 File system event rate per user per hour.

Type	*Hawthorne*	*Kingston*
READ	149.28	123.19
WRITE UPDATE	8.61	4.85
WRITE REPLACE	1.59	1.63
WRITE NEW	5.71	5.44
RENAME	2.36	2.59
ERASE	7.64	6.57
FULIST/FILELIST	1.87	1.72
STATE	1551.68	802.50
STATE (R/W disks)	8.20	8.64

Table 5 Arrival rate coefficient of variation comparison.

Interval (s)	*Hawthorne*	*Kingston*
0–3600	12.58	13.29
1–100	1.47	1.40
2–100	1.26	1.22
3–100	1.13	1.10
3–60	0.93	0.91
3–80	1.03	1.02

primarily caused by user interactions which create activity after a period of (short or long) quiescence.

With any interarrival time boundary used to arbitrarily separate these two groups, there was some overlap. Some user interactions, especially those performed with function keys, could occur very rapidly, and many "bursts" could have their component file references separated by intervals of CPU consumption (e.g., the compilation of a large program) or the dispatching policy of the operating system. However, by using a boundary of 1–3 s we were able to get a good fit using an exponential distribution. The mean interarrival times for this distribution, which depend on the number of users and their activities, were different for the two systems. The mean arrival rates per user were also different. This was somewhat expected, because of the difference in number of users and the type of activities in which they were engaged.

A necessary (but not sufficient) condition of the exponential distribution is that the coefficient of variation is 1. That is, the mean and standard deviations for this distribution are the same. As further corroboration of the exponential fit, we calculated the coefficient of variation for interarrival times up to an hour. We also calculated it for smaller intervals that tended to exclude the bursts of file system commands caused by a user command and for very long interarrival times which were often caused by users who remained logged on (at our request) while performing other work such as attending meetings. The results are shown in **Table 5**. We were surprised by the similarity of the distributions for both systems. The interarrival times, which were between 3 and 80 s, had a coefficient of variation that was 1.03 in the Hawthorne case and 1.02 in the Kingston case. We believe this time range is made up largely of user-entered commands and excludes most of the file event *bursts* and long interruptions.

Conclusions

Both the Hawthorne and Kingston systems were similar to systems studied previously, especially various UNIX systems, in the following respects:

- Static file size distribution was similar to that found in [2] for a TOPS-10 system.
- There was considerably more read activity than write activity; this was very close to the result found for UNIX by Floyd [4].
- File accesses in both CMS and UNIX were bursty.
- Most files were read sequentially and in their entirety; this is similar to (but exceeds) conditions found in two UNIX systems [4, 5].
- Most files were written sequentially, as in UNIX [5].
- Most read and write accesses were for small amounts of data, but the accesses to large files accounted for most of the bytes transferred. This strongly resembled the findings of the UNIX studies of Floyd [4] and Ousterhout et al. [5].
- New files tended to have short lifetimes similar to (but not quite as short as) those found by Ousterhout et al. [5].

This similarity, despite significant differences in the file systems and the manner in which files are used, suggests that these general patterns are likely to be seen in other systems as well.

Comparing Hawthorne to Kingston, we found that the two systems were similar (different) in the following respects:

- Both systems exhibited temporal locality of file reference for both reading and writing. This, in conjunction with the high read/write ratios, suggested a benefit from using a cache to improve I/O response time. This in fact turned out to be the case, as subsequent work confirmed [7].
- The static distributions of both files and space were similar (although there were more very small files at Hawthorne).
- On both systems, over 90% of the read events were less than 16 KB (although there were many more small read events at Kingston).
- On both systems, fewer than a third of the bytes read were from read events that were less than 16 KB.
- Small write events (≤256 bytes) dominated on both systems (although this was much more true of Hawthorne).
- On both systems, write events to large files accounted for most of the bytes accessed (although very large files dominated even more at Kingston).
- Both systems had high (but different) read/write ratios.
- The rate of WRITE REPLACE, WRITE NEW, RENAME, ERASE, directory listing—FULIST and FILELIST—and R/W STATEs were similar for both systems (the rates of READs, WRITE UPDATEs, and STATEs were significantly higher at Hawthorne).
- The interarrival distribution of user commands appeared to be exponential on both systems (although the rates differed significantly).

We found some correlation between the static file size distribution (fraction of all files by file size) and the distribution of the fraction of files read by read event size. The comparison of static space utilization by file size with the fraction of bytes read by read event size was mixed, giving a negative correlation coefficient at Hawthorne and a positive one at Kingston.

We concluded that it would be possible, although certainly not trivial, to build a model of CMS file system activity. This model could be table-driven so that, given the proper metrics, any VM/CMS system could be modeled. However, detailed inspection of the traces showed that the distributions were strongly affected by the popular tools and applications being used. As these tools and applications evolved and changed, the distributions would also be likely to change. Therefore, any model would probably not be robust and would require frequent (and tedious) validation using traces.

Finally, a note of caution. This study was done in the mid-1980s, when most of the population at both Hawthorne and Kingston still used mainframe timesharing services. We expect that, to the extent that the work performed is similar, file reference patterns on workstations are similar. However, as technology changes encourage different modes of work, it is likely that file reference patterns will also change. For example, although there was some use of graphics on VM during this period, it was not as prevalent as it has become on workstations. Extensive use of graphics and image data is likely to increase the amount of data transferred per user and also to affect the dynamic reference distribution. In fact, a recent UNIX study indicates a significant increase in file activity in a distributed workstation environment [9].

References

1. S. J. Mullender and A. S. Tanenbaum, "Immediate Files," *Software Pract. & Exper.* **14**, No. 4, 365–368 (April 1984).
2. M. Satyanarayanan, "A Study of File Sizes and Functional Lifetimes," *Proceedings of the Eighth ACM Symposium on Operating Systems Principles*, Pacific Grove, CA, 1981, pp. 96–108.
3. M. J. Bach and R. Gomes, "Measuring File System Activity in the UNIX System," European UNIX Users' Group, London, Spring 1988, pp. 43–52.
4. R. A. Floyd, "Short Term File Reference Patterns in a UNIX Environment," *Technical Report 177*, University of Rochester, New York, March 1986.
5. J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson, "A Trace-Driven Analysis of the UNIX 4.2 BSD File System," *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, Orcas Island, WA, December 1985, pp. 35–50.
6. W. Pope, "Dynamic Access of CMS Files," *Research Report RC-10483*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, September 1984.
7. G. P. Bozman, "VM/XA SP2 Minidisk Cache," *IBM Syst. J.* **28**, No. 1, 165–174 (1989).
8. J. Ousterhout and F. Douglis, "Beating the I/O Bottleneck: A Case for Log-Structured File Systems," *Oper. Syst. Rev.* **23**, No. 1, 11–28 (1989).
9. M. Baker, J. Hartman, M. Kupfer, K. Shirriff, and J. Ousterhout, "Measurements of a Distributed File System," *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, Pacific Grove, CA, October 1991, pp. 198–212.

Received March 27, 1990; accepted for publication January 21, 1992

Gerald P. Bozman *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Mr. Bozman is a programmer with a special interest in operating systems. Recently he has worked on System Managed Storage for MVS/DFP™ and the Parallel Processing Compute Server, a prototype System/390™ multicomputer.

Hossein H. Ghannad *IBM Networking Systems, Via Paolo Di Dono 44, 00143, Rome, Italy.* Mr. Ghannad is an advisory performance analyst in the NetView®/Distribution Manager design department of the Rome Networking Systems Laboratory. He joined the IBM Communication Systems Division in 1981 as an associate programmer. Mr. Ghannad has performed numerous modeling activities, performance evaluations, and measurements for the DPPX/8100, VM/SP HPO, and VM/XA™ operating systems. His performance-evaluation activities on VM products from 1983 to 1987 included workload characterization of the VM/CMS file system and modeling and analysis of the intersystem serialization mechanism. The work described in this paper was done primarily while he was on assignment at the IBM Thomas J. Watson Research Center from 1985 to 1986. Mr. Ghannad joined the VTAM™ organization at Research Triangle Park as an advisory performance analyst in 1987. He received an IBM Excellence Award in 1989 for identifying inefficient searches in VTAM and proposing solutions to improve their performance. He also developed an OS/2®-based tuning tool prototype for VTAM while at Research Triangle Park. At present, Mr. Ghannad is on assignment to the IBM Networking Systems Laboratory in Rome. He has a B.A. in mathematics from the University of Wisconsin at Oshkosh, an M.A. in mathematics from Morgan State University in Baltimore, and an M.S. in computer science from the University of Virginia, and is a graduate of the IBM Systems Research Institute. Mr. Ghannad has taught several courses, both before and after joining IBM. He developed the Computer Performance Evaluation course and has taught it annually at several IBM Mid-Hudson Valley locations and at Research Triangle Park. He has also published several papers on computer performance modeling.

E. D. Weinberger *RTC, 17th Floor, Hong Kong Bank, 140 Broadway, New York, New York 10005.* After receiving his B.S. in mathematics from MIT in 1973, Dr. Weinberger was a programmer in the aerospace industry until he began graduate training in mathematics in 1981 at New York University. He continued working in the computer industry while in graduate school, first at a small software firm developing a PC-based

system to draft legal documents, and then participating in the research that led to this paper. Subsequently, he became interested in the way biological evolution implements parallel solutions to the problem of finding optimally fit organisms. This topic was the subject of his 1987 dissertation and postdoctoral research at the University of Pennsylvania and then at the Max-Planck Society in Göttingen, Germany. Between postdoctoral university assignments, Dr. Weinberger developed the data-compression algorithm now used by Prodigy Services, as well as a new file-compression technique. He is now a consultant to the financial industry in New York City.