SLAC-200
UC-32

STAN-CS-77-594

FILE MIGRATION*

EDWARD P. STRITTER

STANFORD LINEAR ACCELERATOR CENTER

STANFORD UNIVERSITY

Stanford, California 94305

PREPARED FOR THE ENERGY RESEARCH AND DEVELOPMENT ADMINISTRATION

UNDER CONTRACT NO. E(04-3)-515

January 1977

Printed in the United States of America. Available from National Technical Information Service, U S Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161. Price: Printed Copy $5.50; Microfiche $3.00.

*PhD dissertation

ABSTRACT

This thesis considers the problem of automatic file migration. In a computer system, program and data files are stored on secondary memory such as discs. If the capacity of the secondary memory is not sufficient to hold all the files, some sort of back-up memory (e.g. magnetic tape) is used to supplement secondary storage. The transfer of files between secondary and back-up storage is refered to here as "migration." Migration may be done explicitly by each user or automatically by the operating system. This work studies automatic migration strategies.

We seek the migration strategy which maximizes the probability that a file that is accessed will be found in secondary memory and which minimizes the migration activity. The problem of deciding which files should migrate, the file replacement problem, is analogous to the well-studied page replacement problem. It reduces to the problem of predicting the next inter-access interval for a file. An important difference between the file replacement problem and page replacement is that files are not equal in size. In addition, the size and time scales for file accesses are much larger than for page references. Consequently most page replacement results need to be reconsidered.

Trace data of file system activity on a general purpose computer system have been collected for one year. This type of data has not been published previously. The data provide the basis for a quantitative study of migration. Statistical analysis of the trace data reveal characteristics of file system activity. Special attention is paid to possible correlations of files' inter-access intervals to other factors, which may be useful in predicting inter-access intervals.

Various possible file replacement, or migration, strategies are presented. Two non-equivalent criteria, the miss ratio and migration traffic rate, are used for evaluating strategies. LRU (least recently used) and MIN (the theoretically optimum strategy for fixed sized pages) are studied. Other strategies, including LFU (least frequently used), that are suggested by the results of statistical analysis, are examined.

The fact that files are not equal in size can be used to advantage and the previously mentioned replacement strategies are shown to be improved when the size of a file is taken into consideration. Finally some non-realizable strategies using knowledge of files' future accesses are presented for comparison with the practical strategies already mentioned.

A simulation model of file system activity is examined. The model is useful for generating artificial trace data. Simple parameter changes allow the model to simulate other environments.

This work could never have happened without the guidance and encouragement of Forest Baskett.

Thanks also go to my other committee members, Vint Cerf, Tom Bredt and Don Perkel for their valuable suggestions. I wish to acknowledge the Computation Research Group at SLAC for supporting me, and Harriet Canfield for her typing. Finally, the job was made made easier by the excellent computing facility provided at SLAC by the Stanford Center for Information Processing, and by the graphics programs of Bob Beach and Roger Chaffee.

Work supported by the U. S. Energy Research and Development Administration contract E(043)-515 and the Joint Services Electronics Program number N00014-67-A-0112-0044.

TABLE OF CONTENTS

LIST OF FIGURES.....vi

CHAPTER I. Introduction to File Migration Concepts.....1
1.1 Introduction.....1
1.2 The File System Environment.....2
1.3 File Migration Concepts.....3
1.4 File Migration Strategies.....4
1.5 Relation to the Page Replacement Problem.....7
1.6 Thesis Summary.....8

CHAPTER II. Empirical Data on File System Activity.....10
2.1 Introduction.....10
2.2 Description of the Data.....11
2.3 Method of Collection.....11
2.4 Strengths and Shortcomings of the Data.....12
2.5 Characteristics of File System Activity.....15

CHAPTER III. Statistical Analysis of File Activity Data.....22
3.1 Introduction.....22
3.2 Analytic Methods.....23
3.3 The Accessing Process of a File.....25
3.4 Implications of the Accessing Process Model.....29
3.5 Accesses to Libraries.....35
3.6 Distribution Fitting for Inputs to the Model.....35
3.7 Dependency Relationships of Access Time Intervals.....39
3.8 Summary.....43

CHAPTER IV. Simulation Model.....46
4.1 Introduction.....46
4.2 Basic Structure.....46
4.3 Development of the Model.....47
4.4 Validation.....48
4.5 Uses of the Model.....49

CHAPTER V. File Migration.....51
5.1 Introduction.....51
5.2 The Implementation of File Migration.....53
5.3 Evaluation of Migration Strategies.....55
5.4 Stack Algorithms and Stack Processing.....58
5.5 A Primitive Migration Strategy.....63
5.6 The LRU and MIN Migration Strategies.....66
5.7 Alternatives to LRU.....67
5.8 Migration Strategies Based on File Size.....74
5.9 Other Migration Strategies.....78
5.10 Conclusion.....78

CHAPTER VI. Examples.....83
6.1 Improved Performance from New Migration Strategies.....83
6.2 Changes in Storage Hierarchy for Improved Performance or Costs.....88
6.3 The Migration Traffic Evaluation Measure.....94
6.4 Summary.....96

CHAPTER VII. Discussion.....98
7.1 Summary and Conclusion.....98
7.2 Previous Work.....99
7.3 Future Work.....101

APPENDIX A: The Negative Binomial as a Compound Poisson.....103

APPENDIX B: Non-optimality of MIN Replacement.....104

BIBLIOGRAPHY.....105

LIST OF FIGURES

Figure 2.1 Distribution of file sizes.....17
Figure 2.2 Fraction of storage by file size.....19
Figure 2.3 Fraction of files and space by age since last access.....20
Figure 3.1 Number of files accessed by day.....32
Figure 3.2 Number of files on line by day.....34
Figure 3.3 Lifetime distribution of files.....37
Figure 3.4 Age distribution of files.....38
Figure 3.5 Size distribution of files.....40
Figure 3.6 Mean idle time by file size.....41
Figure 3.7 Mean idle time distrubution.....42
Figure 3.8 Mean idle time by file lifetime.....44
Figure 5.1 Hit ratio by age since last access.....52
Figure 5.2 Least recently used miss ratio.....61
Figure 5.3 Least recently used migration traffic.....62
Figure 5.4 Migration threshold replacement.....64
Figure 5.5 Variance of running average estimate.....70
Figure 5.6 Number of accesses in lifetime.....71
Figure 5.7 Running average strategy.....72
Figure 5.8 Least frequently used.....73
Figure 5.9 Least recently used times size.....76
Figure 5.10 Least densely used.....76
Figure 5.11 Replacement by file size.....77
Figure 5.12 Sort on next idle time times size.....79
Figure 5.13 Random replacement.....80
Figure 5.14 Replacement by file lifetime.....81
Figure 6.1 Migration threshold replacement miss ratio.....84
Figure 6.2 Least recently used.....86
Figure 6.3 Least recently used times size.....87
Figure 6.4 Least recently used times size.....90
Figure 6.5 Comparison of miss ratios.....93
Figure 6.6 Least recently used migration traffic.....95

CHAPTER I

INTRODUCTION TO FILE MIGRATION CONCEPTS

In most current large computer systems, program and data files are stored primarily on secondary storage, such as magnetic discs. When the amount of disc space available is not adequate to store all existing files, some files must be moved to a slower and less expensive medium, called back-up store, generally magnetic tape. The process of transferring files between secondary storage and back-up storage is referred to in this paper as "migration". The decision as to which files should migrate may be made in a number of ways. For instance, many computer systems allocate to each user a fixed amount of disc storage and require the user to arrange for migration of his own files when necessary. This thesis explores algorithms for automatic migration, that is, migration controlled by the system and (possibly) hidden from the user. The primary goal of this work is to determine what migration strategies are most effective and efficient. It will be demonstrated that with proper automatic migration in a computer system, the amount of disc space required, and therefore the total system cost, can be substantially reduced.

The work presented here is in two major parts, a model of a file system environment, and a study of migration algorithms. First, extensive trace data on file system activity are presented. These data are used to characterize the activity of a disc file system on a modern computer. In addition, the trace data are used as input to simulations of the various migration algorithms to be studied.

A detailed model of disc file activity has been developed. The model generates artificial trace data that simulate the empirical data mentioned above. Thus, the model provides a useful source of input to migration algorithm simulations. Furthermore, while the empirical trace data characterize only the computer system that was actually measured, the model, by changing appropriate parameter values, can be used to simulate other environments as well. Finally, the model provides insights into the nature of the file accessing process.

The second part of the thesis uses the file system characterization developed in the first part to study migration algorithms.

This chapter presents concepts of file migration and outlines and motivates the rest of the work in non-technical terms.

1.2 The File System Environment

We assume in the following that the computer system under discussion is configured as follows:

- There is a limited amount of fairly expensive "secondary storage" (typically disc storage) on which users may store their program and data files.

- There is an unlimited amount of "back-up storage" which is relatively less expensive, but much slower than secondary storage (this is usually magnetic tape). This back-up storage may consist of a hierarchy of storage devices (such as magnetic strips at one level and removable tape reels at a lower level).

- Either the top level (secondary storage) of this memory hierarchy is the only level directly accessible from a running program (this effectively implies that secondary storage is randomly, as opposed to serially, accessible), or program access to files on lower levels of the hierarachy (back-up storage) is so time consuming as to be highly undesirable.

If the system can be managed so that only those files which will be accessed in the near future are kept in secondary storage, the amount of secondary storage capacity, and thus the total system cost, can be reduced without significantly changing system performance. This is the purpose of implementing a migration algorithm.

The need for some sort of migration system is fairly evident. Computer systems' users often have a number of saved files that are not currently in use but which are still kept in the computer's storage system. Thus, a significant portion of the secondary store is used to hold inactive files that should be stored on back-up storage. We might expect that a file that has been idle for some time is less likely to be accessed in the near future than a file that has been used recently. Both of these fairly evident statements, that there are many idle files on the secondary store and that idle files are likely to remain idle, will be supported mathematically in Chapters II and III. These two conditions supply the intuitive rationale for having an automatic migration scheme.

1.3 File Migration Concepts

Given that the need for migration has been established, there remain the questions of how to implement migration (the migration algorithm) and how to decide which files should migrate (the migration strategy). Most users will not move their files to back-up storage unless encouraged or coerced to do so. Even if this were not the case, it would be attractive to have the operating system take care of migration automatically without requiring user knowledge or intervention.

In the automatic migration system envisioned here, the user need not know on what type of storage files are saved. The system keeps track of the location of all current files and makes the decisions as to which files should be most accessible and which can migrate to less expensive storage. Making the migration scheme invisible to the user frees the user from having to understand the exact configuration of the storage system and the details of using the various devices. It also ensures that any changes in the number and type of storage devices that are available will necessitate changes only in the migration system programs. In fact, though this thesis generally refers to the storage system as though it consisted of only two levels, disc and tape, the

number of levels in the system hierarchy could be changed with very little impact if the hierarchy is accessed only through the file migration programs. This would allow system designers to easily add various new devices that may become available (charge-coupled devices, magnetic bubbles, laser addressed storage, etc.) for back-up storage.

It is desirable, in some instances, in spite of the transparency to the user of the file system hierarchy, to allow a user to direct the migration system to move files. The user should be able to specify, for instance, that a certain file will not be used for a given period of time so that the system can immediately transfer it to the appropriate level in the storage hierarchy. Also, users need the ability to specify that a certain file be stored on some transportable medium, such as magnetic tape or removable disc pack, that can be physically removed from the computer site.

Basically then, what the file migration system does is to automatically move files to back-up store when they are inactive, and automatically restore them to more accessible hierarchy levels when they are re-accessed. The major portion of this thesis deals with how the migration system can decide which files should be migrated.

1.4 File Migration Strategies

Currently running computer systems generally do not provide automatic migration. The problem of secondary storage being unnecessarily filled with inactive files is ignored or avoided. One way to deal with the problem is to provide enough secondary storage to satisfy all users' demands. This solution ensures that no migration needs to be done, but as we will see, is unnecessarily expensive. Very nearly the same level of service (as measured by access time to saved files) can be provided with significantly smaller secondary storage capacity (and therefore lower cost) if appropriate migration is done.

A more common solution in current computer systems is to pre-allocate to each user a fixed amount of secondary storage. The user

then is responsible for deciding which files to keep in secondary storage and which files to manually "migrate" to back-up storage. This strategy may, in fact, be quite efficient since the person with the most knowledge of a file's accessing pattern, the user, makes the decision as to which file to migrate. On the other hand, this scheme is probably too inflexible to be very efficient. A new user may not have developed enough files to fill the allocation so some storage remains unused. More active users may have insufficient storage allocation to cover their needs, so that they increase the processing load on the I/O system by requiring many transfers of files back and forth from back-up storage. This is analogous to the thrashing problem in paging systems [De68]. Another problem with this fixed allocation method is that the amounts of allocation are likely to be determined, not on the basis of level of activity of the respective users, but on some other measure such as rank in the company or the amount of money the user is prepared to spend.

In any case, this fixed allocation scheme is probably the most common method of controlling secondary storage. We shall see in this thesis that very simple migration schemes can provide much more flexible and efficient use of expensive storage. At the very least, a file system should provide easy to use system routines that enable the user to move his own files to and from back-up storage. It might be convenient for the file system to automatically cause a file to migrate whenever the user attempts to save another file that would overrun his secondary storage allocation. If this system were used, the system probably should ask the user which old file should migrate since the user has the best knowledge of which files are likely to be active in the future.

A simple form of automatic migration in use on some computer systems is one where all files that have been idle for a given amount of time are caused to migrate. This and the following migration algorithms are discussed in detail in Chapter V. The time quantum for which a file is allowed to be idle before migration must be adjusted according to the activity in the system and the capacity of secondary storage. If the

quantum is on the order of several weeks or a month, then the percentage of migrated files that are immediately brought back to secondary storage will probably be small. Notice that if many files show cyclic activity, for instance being accessed once a month, then the time quantum should be large enough to contain one cycle, or extra migration activity will be necessary. If most files have cyclic accessing patterns, however, some other form of migration, such as immediate migration and pre-fetching, might be more appropriate. The data used in this study, obtained from an actual computer system and analyzed in Chapter III, does not show any such cyclic behavior.

Choosing a fixed quantum size as a threshold idle time for migration will involve some storage inefficiencies. Storage usage will vary over time, and the total size of all files that have been idle less than the idle quantum will vary as well. If the idle quantum chosen is too small, then the total size of the files in secondary storage will be less than the total available secondary storage, and some secondary storage will be unused. If the quantum is too big, then secondary storage may not be large enough to hold all the files that have not migrated. Clearly, what is needed is continual adjustment of the quantum size so that secondary storage is always just full. In other words, we migrate only enough files to assure that secondary storage can contain all files that have not migrated, and we migrate files that have been idle the longest time. This scheme is called the least recently used replacement (LRU) algorithm and is familiar from studies of page replacement algorithms. Chapter V presents the LRU replacement algorithm in detail.

Throughout the preceding non-technical discussion, we have used the idea that a file that has been accessed in the recent past is more likely to be accessed in the near future than a file that has been idle for some time. This idea is intuitively appealing. That this is, in fact, the case will be shown in Chapter III. LRU replacement then, uses the file's current idle time as a measure to predict the file's future idle time (that is, to predict when the file will be accessed next). While current idle time is an obvious predictive measure, it is not

clear that it is the best possible measure. We seek the measure that will best predict which files will be accessed in the near future. Some other measures that might be useful are the file's accessing rate (a measure of past activity), the file's size, the file's age since creation, etc. In addition, some combination of these or other measures might be the best predictor. The main emphasis of this work is to evaluate these potential predictors.

Because, in general, the computer system cannot know the future accessing pattern of a file, the migration predictors are only heuristics for guessing future activity. Using trace data or artificially generated activity data however, we can study algorithms which take into account the future accessing pattern of a file. These algorithms, though not implementable, give us theoretically near-optimal or optimal behavior against which to compare our practical algorithms. Algorithms with knowledge of the future are also discussed in Chapter V.

1.5 Relation to the Page Replacement Problem

The page replacement problem in a virtual memory system, how to decide which page to swap out when a new page must be brought in, has been studied extensively. A file migration system is similar to a virtual memory system in that we manage a top level of memory, that is not big enough to hold all the necessary information, by transparently (to the user) moving information to and from a back-up memory when needed. This analogy is useful and this work takes advantage of some paging results and methods. There are important differences, however, that must be noted. One obvious difference is that of scale. In the time domain, page replacements may occur many times per second, whereas in this study, we will measure time in days. Page sizes are usually on the order of 512 or 1K words while file sizes range from 1K words to 100K or larger. The crucial difference, however, is that pages are assumed to be of one fixed size but files may vary in size over a large range. Allowing variable sizes violates an assumption that is crucial to some paging results so that these results do not necessarily apply to

the file replacement problem. We find, in general, that we can prove less about the file migration environment and that, consequently, file migration algorithms must depend more on heuristic methods.

1.6 Thesis Summary

Chapters II and III discuss the empirical data on file system activity that were collected and which form the basis for this study. Chapter II describes the trace data and its collection, and makes some observations on the data. Chapter III presents a more detailed statistical analysis of the data. This analysis uncovers some characteristics of file system activity that are crucial for designing file migration algorithms.

In Chapter IV, we discuss a model that has been constructed to generate simulated trace data similar to the empirical data presented in the previous two chapters.

The main results of this work are presented in Chapter V on migration strategies. Readers primarily interested in migration, but not a study of file system activity, can read Chapter V as a self-contained unit. A detailed development of migration schemes through various refinements is given. Possible predictors are presented and evaluated. Also, various theoretically near-optimal (through unimplementable) algorithms are used for comparison to the realizable algorithms.

Chapter VI gives some examples (using representative values) of possible savings resulting from using migration. Multilevel storage hierarchies using newly emerging technologies are briefly discussed. The relation of the present work to computer networks and distributed data bases is mentioned.

A conclusion and discussion of possible directions for future study are given in Chapter VII. Chapter VII also relates this work to other related studies and gives a review of the relevant literature.

Appendices A and B provide further development of some ideas in Chapters III and V. Appendix A shows the mathematical derivation of the negative binomial distribution as a compound Poisson. The non-optimality of the MIN algorithm is demonstrated in Appendix B.

CHAPTER II

EMPIRICAL DATA ON FILE SYSTEM ACTIVITY

The first step toward gaining an understanding of file system migration is to make measurements and gather trace data on file system activity for a real computer system. These data, and their collection, are described in this chapter. The environment for this empirical study was a large computer center used primarily for physics research.* The user population is quite large (700 users) and the job stream is a mixture of short runs for program debugging and the like, and large production jobs for physics data reduction, information retrieval, etc. In this sense, the system under study may be more typical (or at least more diversified) than one which only handles small student programs or an industrial or business situation where most jobs are large, batch mode, production runs. In any case, the results of this paper are applicable to the environment under study. They apply to other environments only to the extent that this environment is typical. The model of file system activity presented in Chapter IV can be scaled by changing various parameters to simulate other file system environments. However, the question of validating the scaled model by comparison to some other real system, though discussed in Chater IV, is not directly addressed in this work.

Disc storage** serves as the secondary storage for this system. Backup storage is magnetic tape. All users store their files on disc. Each user is allowed a fixed amount of disc space for storing files. No migration of files is done by the system, but routines are available for users who wish to transfer files to back-up storage (magnetic tape) themselves.

*The Stanford Linear Accelerator Center. Equipment consists of two IBM/370/168's and an IBM 360/91, loosely coupled with ASP and supporting WYLBUR, an interactive text editing and remote job entry system.

**IBM 2314 and 3330 type devices

2.2 Description of the Data

The data consist of a day to day accessing pattern for all user disc files on the system. The measurement period was just over one year. A similar set of data was collected independently during this time at another installation. It is briefly described in [Re74]. For each disc file present in the system, with the exception of temporary scratch files which were ignored, the following data were recorded:

file identifier
file creation date
file deletion date
file size
day by day accessing pattern

The accessing pattern consists of a bit string, one bit for each day in the measurement period, with bits set for each day that the file was accessed (read or written).

The collection of these data records gives a complete trace (with time unit of one day) of all user disc file activity in the system for one year. From the data, we can derive day to day system activity, age, creation, and deletion patterns of files, size distributions of files, idle time distributions for files, etc.

2.3 Method of Collection

The data collection process was very simple. A system accounting program is run every night which prepares a file listing all user files on the system, their size, and other information. The data collection program was simply attached to this accounting program as an extra job step. The data collection program maintained a trace file which it compared every night to the newly created accounting file. The trace file was then updated, recording file creations and deletions and setting the appropriate bit in the access bit string associated with each file that the accounting file indicated had been read or written during the previous day. No modifications, either to the accounting routines or the file accessing routines, were necessary for this data collection. In fact, the only input used by the collection program, the listing of all files on the system, is public information available to

all users.

Other computer installations maintain a similar listing of all current files. Usually however, the date of last update is recorded for each file. In the present case, the data of last access is recorded. This is crucial in determining the accessing pattern of a file. The data collected here are somewhat unique, therefore, because they cannot be easily generated on other computer systems.

2.4 Strengths and Shortcomings of the Data

As mentioned above, the data was very easy to collect. More importantly, the cost of data collection was negligible and the impact on the system of making these measurements was also negligible. This was not a case, as sometimes happens with system measurement, where the measurement itself causes the system to behave differently from the way it would if no measurement were being done.

The nature of the migration process requires that we look at file system activity on a macroscopic, or relatively high, level. The costs, speeds and capacities of secondary storage devices imply that we must deal with migration in terms of hours or days rather than minutes or seconds. Furthermore, in order to be able to average local variations over time, we needed data for a number of months, not just a few days. Thus, the file trace data for a period of one year, in time units of one day, are quite adequate and appropriate for this study. In fact, if the data had consisted of a trace of each individual file access, the amount of data collected over a full year would probably have been unmanageable.

There are some disadvantages to this set of data. In particular, having the unit of time equal to one day, though advantageous in some ways as discussed above, masks some information that could be useful. In the study of various migration algorithms in Chapter V, we look for various predictors for the time of the next access to a file. One possible predictor is the access rate a file has experienced in the

past. Generally, files are not accessed every day and so we can derive a fairly accurate access rate from the trace data. For very active files, however, the data does not allow us to distinguish between a file that is accessed once a day and a file that is accessed 100 times a day. This lack of resolution in the time scale also impedes the statistical analysis in Chapter III.

Another shortcoming of the trace data is the unit of measurement of size. The accounting data file used to gather the trace data lists file sizes in units of one track (about 7200 characters or 1800 words). Thus, any file whose actual size is less than 1800 words (1 track) is shown as being one track in length. This masks the actual size of many files. This is particularly unfortunate since, as we will show later in this chapter, the size distribution of files is strongly skewed to small files (there are many more small files than large files). One mitigating factor in this situation is the existence in the system of "libraries".* Because some of the secondary storage devices use a track as the smallest unit of transfer, it is impossible to store files that are smaller than one track without wasted space. The system discourages users from storing files smaller than one track and provides the alternative of libraries. A library is a single physical file that is made up of a collection of small logical files. Because libraries are used extensively in this file system, there are fewer files whose size is less than one track, and thus the lack of resolution in determining file size is not as much of a problem as it might have been without libraries.

The existence of libraries means that the distribution of file sizes that we derive from the trace data does not match the size distribution we would find if there were no libraries. As we will see later in this chapter, there is a preponderance of small files on the system. Libraries mask this effect; users actually use many more small files than the trace data show.

*called Partitioned Data Sets by IBM

The nature of the trace data has some other disadvantages. For instance, no indication is made as to whether an access to a file was a read or a write operation. All we can determine about a file is whether it was accessed on a given day, not how often it was accessed and whether or not it was modified. Knowledge of whether a file had been modified might have helped in designing algorithms for choosing files to replace when migration must take place. As in paging systems, a file that has not been altered while in secondary storage need not be rewritten onto back-up storage because its image on back-up storage, provided one is kept, is still valid. Thus, it might be advantageous to choose for replacement those files that haven't been written in preference to those that have been altered. The nature of the trace data does not allow this distinction to be made.

Some indication of which files tend to be accessed together can be derived from the trace data. However, no explicit indication of this is available. Thus, if there is some kind of locality, or clustering of references to files as there is in page references [De66], such information, which might be useful in migration replacement decisions, can be determined only indirectly. More detailed data might also be able to take advantage of naming conventions that reflect groupings of files by project or usage, groupings of files by usage in nodes of the catalog hierarchy, or actual indications by the user of files that will be used together.

The trace data include only disc files. Files stored on tape are not included. Since each user has a fixed allotment of disc space, some users must do their own migration of inactive files to tape. This is not reflected in the trace data.

No system files (compilers and utility programs, etc.) are included in the data. Systems files are assumed to be used often enough that they must be kept in secondary storage and never migrate.

Finally, the data trace file activity only for the computer system

that was measured. What differences there may be with other computer systems' file activity is not known. Comparison with the only other similar data on file systems known to the author [Re74], [Re75] shows that these data may be fairly representative of file activity in a large, general purpose computing system.

2.5 Characteristics of File System Activity

A total of more than 25,000 files were recorded during the measurement period. There were typically 5000 files present in the system at any one time, using about 45,000 tracks. The secondary storage capacity of the system is 67,000 tracks (500 million characters).

The data collected do not allow us to determine who the owner of a given file is so we cannot break down the file population by user. However, there is nearly a one-to-one correspondence between users and user ID's (the identification a user must supply to the computer). Some users have more than one user ID, some users share a common ID. The file naming conventions used in this system prefix each file name with a user ID. We can, therefore, distinguish between files belonging to different user ID's. A total of 707 distinct user ID's were recorded. On a typical weekday about 225 user ID's would have some files accessed. On the average, about 650 separate files are accessed each weekday. The total space occupied by the files accessed in one day averages about 13,000 tracks. These figures are intended to give a general impression of the size of the computing environment under study.

It is instructive to look at some daily statistics of the data and see how they vary over time. For instance, the number of files created each day is a randomly varying number averaging about 68 new files per day. Statistical analysis* shows that the number of new files is a stationary process (its average value does not vary over time).

*The statistical methods used in this study are described in Chapter III.

Similarly, the number of files deleted each day is a stationary process with mean 64.

This would indicate that the number of files present in the system is increasing at a rate of 4 (68-64) per day. Indeed, the plot of files present in the system by day shows clearly an increasing trend with slope of about 4.

The number of files accessed per day is also increasing with time, but at a rate of about 2 files per day. Since this rate is slower than the total growth rate of the system, it is clear that the percentage of idle files in the system is increasing. This is one indication of the need for a migration policy.

The data also reveal some interesting facts about how a file's size is related to its activity and lifetime in the system. Figure 2.1 shows the size distribution of files present in the system at a typical point in time. The mean file size is around 8.4 tracks (as indicated earlier by 5000 files occupying 45,000 tracks). The expected size of a file that is accessed, however, is about 20 tracks. This indicates that larger files are accessed more frequently than smaller files.

Similarly, the average size of all files recorded in the system for the entire measurement period is 7 tracks. This means that more distinct small files have been created than their proportion of the total population would indicate. This implies that the expected lifetime of small files is less than that for large files.

These phenomena, large files being more frequently accessed and small files having shorter lifetimes, are partly explained by the presence of libraries. A library is a single file that contains many small logical files. The data do not distinguish libraries from other files. We would expect that libraries would be larger than average files. Because they contain more than one logical file, libraries should be accessed more often than non-libraries (each access to an individual library member counts as an access to the library). Also, because libraries can contain a varying collection of files, with some

DISTRIBUTION OF FILE SIZES

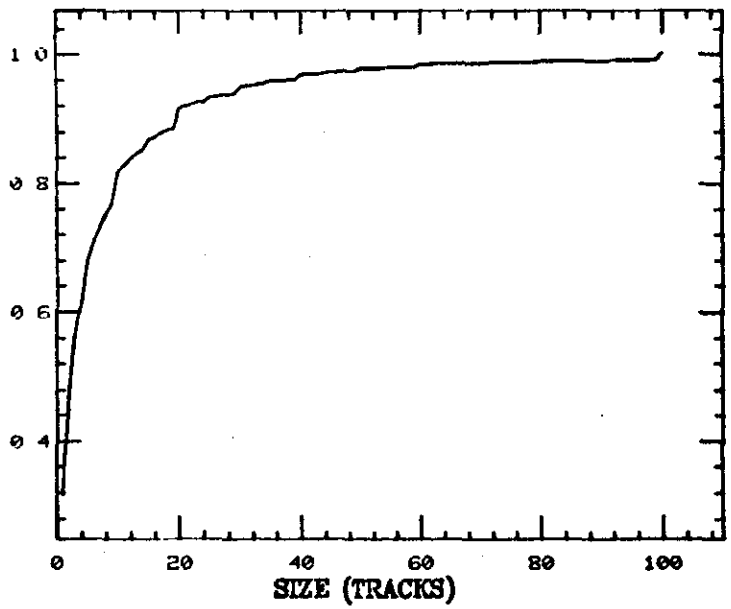

FIG. 2.1

being deleted and new ones created, we expect the lifetime of a library to be longer than average. Thus, libraries help account for the different characteristics of large and small files.

Figure 2.1 shows the size distribution is strongly weighted to small files. Of all files, 32% are one track or less in length. This does not mean however that a migration scheme need only worry about small files, just because they are the most numerous. Figure 2.2 shows the fraction of the total space used that is occupied by various sized files. Point A in this figure indicates that 75% of the disc space in use is covered by only 20% of the files. Large files, while they don't dominate in number, do dominate in total space occupied. This balance between large and small entities may suggest a general rule. It is similar to the rule of thumb commonly used for CPU time in computing systems: 20% of the jobs use 80% of the CPU time.

One indication that a migration policy would save storage is the number of idle files stored in the file system. Figure 2.3a and 2.3b show the fraction of all files by idle time (time since last access) and the fraction of all space in use (allocated to files) by idle time. Of all files, 40% have typically been idle for more than 30 days. Of all file storage space, 20% has been idle for more than 30 days. Migrating these files, on the assumption that if they have been idle as long as 30 days they are not likely to be accessed soon, could save a substantial amount of secondary storage.

Perhaps the simplest migration scheme is to migrate files that have been idle longer than a certain amount of time. Figures 2.3 give an indication of what the appropriate migration threshold might be. A reasonable first estimate might be to stay near the knee of the curve in Figure 2.3b (i.e., about 20 to 25 days). This must be balanced by a knowledge of the amount of migration activity required for each migration threshold. Short thresholds save secondary storage space but cause more input/output traffic for migration. Larger thresholds have the opposite effect. This trade-off is examined in more detail in Chapter V on migration algorithms.

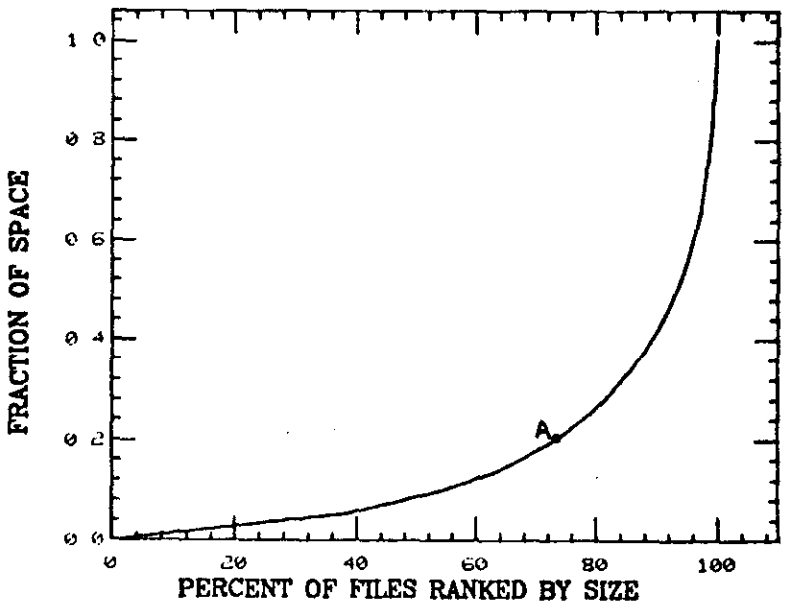

FIG. 2.2

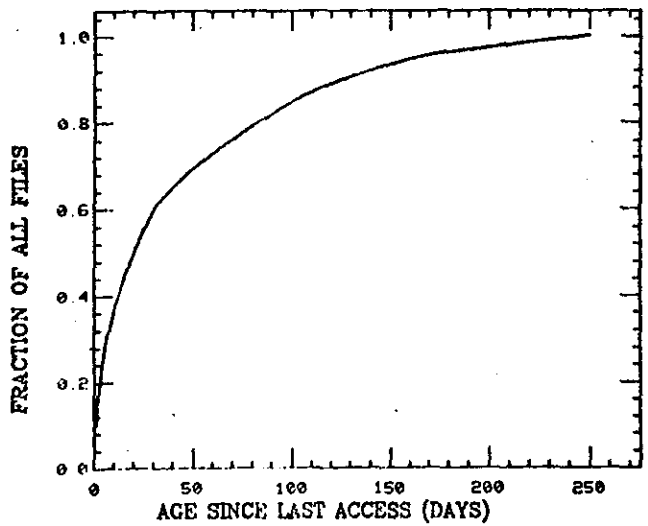

FIG. 2.3A

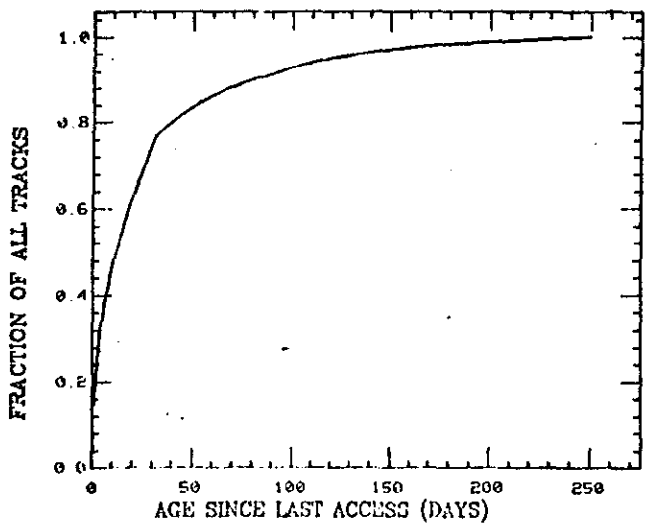

FIG. 2.3B

This section has presented some characteristics of the file system as indicated by the trace data. The intent is to give an overall feeling for the environment under study and to provide some informal indication of why migration is needed. Chapter III gives a more complete statistical analysis of aspects of the data pertinent to modeling the file system and understanding migration strategies.

CHAPTER III

STATISTICAL ANALYSIS OF FILE ACTIVITY DATA

Before studying migration strategies themselves, it is appropriate to do a detailed analysis of the file system trace data. Such a study can be useful in several ways. Basically, we hope to be able to understand the intrinsic structure of the processes underlying file system activity. This knowledge is necessary for building a reasonable model of file activity (Chapter IV). For example, it is necessary before building a model to know the most appropriate probability distributions for describing various events and elements in the system. Among other things, we need the distribution of inter-access intervals (idle times) for files. Distributions for sizes of files and lifetimes of files are also necessary. A basic approach in this work has been to verify the appropriateness of any distribution used in the model, rather than making convenient but untested assumptions (for instance, that a certain distribution is exponential or normal or whatever).

Beyond the particular (marginal) distributions that describe various aspects of the system, it is crucial to understand the dependencies of variables on one another. For instance, we need to know if the lifetime of a file is correlated with its size, if successive inter-access times to a file are correlated, etc.

Knowledge of the underlying structure of file system activity will also aid in choosing and understanding various migration strategies (Chapter V). Here we are looking for predictors of the next idle time (inter-access interval) for a file. Any dependency of idle time on other variables (previous idle time, file size, etc.) may indicate potentially useful predictors.

This chapter presents statistical analysis of the trace data described in Chapter II. Particular attention is placed on the accessing pattern of individual files since this is the basis both for the model and for understanding migration algorithms. Many different mathematical and statistical tools are available for such a study. We

start by describing the tools and procedures used in this particular study.

3.2 Analytic Methods

The analysis presented here draws heavily on the theory of stochastic processes and on the branch of statistics dealing with the analysis of a series of events. The theoretical concepts are reviewed and referenced below. To augment this theory, several computer programs were used. These include a statistical analysis package, a probability distribution curve fitting program, and a program which extracts probability distributions and moments from the trace data. The first step in the analysis was to run the distribution plotting program on the trace data. This custom program merely extracts from the data the distribution and moments of various variables of interest. It also plots the time sequence of variables when appropriate. Informal study of the resulting plots served to provide familiarity with the data. It also helped point out outlying data points and anomalous or unexpected behavior of the data. For example, the fact that weekend activity is markedly different from weekday activity is immediately evident. Unless specifically stated otherwise, this paper deals only with normal weekday activity. Also, the presence of several days in the measurement period when the accounting program, whose output was used for gathering the trace data, was not run, was detected. This allowed the error points to be eliminated or smoothed before more rigorous analysis was performed.

This basic information suggested appropriate probability distributions for some variables. It also indicated the possible structure of some underlying processes, in particular, the series of inter-access intervals for a file. It remained to verify or disprove the hypotheses suggested by this exploratory analysis. The next step in studying the trace data was to use a probability distribution curve fitting program.* This program aids in matching a set of data to

*Originally written by John Zolnowsky at SLAC and modified by the author.

commonly used probability distributions. It first computes the best known estimators of the various parameters of each of nine well known probability density functions. These estimators are either method of moments estimators or maximum likelihood estimators, as appropriate. The program outputs graphs of the empirical data (and empirical hazard function, defined below) plotted with each of the various theoretical probability densities (and theoretical hazard functions), drawn with the appropriate (best estimate) parameter values. The output proved extremely useful in deciding which functions best described various input data, in the absence of specific statistical tests.

It should be noted that graphical analysis such as this should be supplemented with statistical testing whenever possible. In many cases this was done here, as described below. One reason that graphical curve fitting is sometimes inadequate, however, is the practice of fitting probability distribution functions (cumulative probability functions) rather than probability density functions. Cumulative distribution functions tend to be very similar in shape and hard to distinguish. Density functions are somewhat easier to tell apart. Two other factors employed here help increase the accuracy of graphical curve fitting. First plotting monotone decreasing density functions with a log scale on the vertical axis tends to expand the resolution in the midrange of the functions, enabling closer fitting. Secondly, the use of hazard functions ($h(x) = f(x)/(1-F(x))$, also called age-specific failure rate, is very helpful for determining a distribution's behavior in the tail. For each theoretical distribution tested by the fitting program, the theoretical and empirical hazard functions are plotted together. While probability density functions approach zero as the argument approaches infinity, the hazard function may tend to zero, a constant positive value, or infinity, thus providing a better indication of the upper tail behavior of the probability function. (The hazard and density functions are completely equivalent; one can be derived from the other.) Other investigators have used the log survivor function instead of the hazard function. These approaches are equivalent since the hazard function is the derivative of the log survivor. The log survivor function has some of the same smoothing and masking effects with respect to the hazard function as the cumulative function has with respect to the density

function. For a more complete discussion of the hazard function see Cox [Co62] pages 3-7.

The statistical testing done in this study used a publicly available statistical analysis program: SASE IV [Le66a]. This program provides a wide range of tests for analysis of a series of events. Thus, it was ideal for analyzing the inter-access time sequence for files. Some of its routines are useful for other types of data as well. It can be used for detection of trends and for determining serial and cyclic correlations in sets of data. The theoretical background for the program is discussed fully by Cox [Co66]. Use of the program in a number of studies is also well documented [Le66b], [Le71], [Le73]). This paper will not discuss the statistical theory, but will include a discussion of the use and appropriateness of some of the tests provided in SASE IV.

3.3 The Accessing Process of a File

In this section, we study in detail the series of accesses to a typical individual file. We will deal with this series of events as a stochastic point process. Some of the results of this treatment will directly affect our choice of migration strategies in Chapter V. Our approach is to think of the file system as a collection of individually characterized files, rather than as an amorphous set of files. For example, we deal here with the accessing pattern of individual files rather than the probability distribution of the number of files accessed per day, though we will mention the latter as well and discuss its derivation from our more detailed model. This approach is in contrast to other studies of the same type of data [Re75] but is necessary to our understanding of migration strategies and for modeling file system activity in detail. All tests described in this section were performed on the inter-access trace data for a number of randomly selected files, with various sizes and liftimes. Testing a variety of files chosen at random should point out typical file characteristics. Therefore, the following stated results apply to files in general unless specifically noted otherwise.

The intent, then, is to understand the probablistic structure of the sequence of accesses to a typical file. Several models suggest themselves from the theory of stochastic point processes [Pa62]. A stochastic point process, as usually defined, is a series of point events, i.e. events distinguishable only by their time of occurance, that are separated by random time intervals. The sequence of accesses to a file fits this general model and so the theory of stochastic point processes is used here. For our purposes, a stochastic point process can be described by its marginal distribution of interval lengths and by the dependency structure of the intervals. The marginal distribution of intervals is obtained from the empirical interval distribution with the curve fitting progrom described earlier. We first discuss the dependencies of intervals on preceding intervals. The simplest dependency model (no dependency) is the renewal process, described at length by Cox [Co62]. In a renewal process, successive interval lengths are independent of each other. The marginal distribution of interval lengths, therefore, completely describes a renewal process. The simplest model with dependencies is the Markov process. Here the length of the current interval is again a random variable but depends on the length of the immediately previous interval. A generalization gives the nth order Markov chain in which the current interval length depends on the lengths of the previous n intervals. A semi-Markov process is a further generalization. In this model, the system can be in any of a number of states. The current interval length is drawn from the distribution associated with the current state and at the end of each interval, transition to another state occurs with prescribed probability. Other models (branching renewal process, doubly stochastic Poisson process) are discussed by Cox [Co66].

For modeling the inter-access interval process for files, several models are plausible. A renewal process model implies that consecutive idle intervals are independent. This may be true, and the renewal process is attractive because of its simple structure, but it implies a lack of "locality" of reference for files. In studies of accessing patterns to pages of programs, it has been shown that accesses to pages

tend to be clustered in time so that a page may be very active for awhile, then relatively inactive when the references move to a new locality. It is possible that references to files might show the same type of behavior. If this is the case, then a semi-Markov model with two states might be appropriate. While the file is in the "active" state, inter-access intervals are described by some distribution with relatively small mean. There is a certain probability of switching to the "inactive" state where inter-access times have a larger mean value (and possibly a differently shaped distribution). Discussion of the application of a similar two-state semi-Markov process in a different situation (modeling page reference stack distances) is in [Ch76], [Le73]. To choose among the various possible models, we perform statistical tests (using the SASE IV program) on sequences of inter-access intervals for a number of files taken from the trace data. First, the presence of trends in the data must be tested. A stochastic process is said to be stationary if the distribution of interval lengths (or equivalently, the distribution of the number of events in a fixed interval) does not change with time. Stationarity, therefore, is the absence of trends in the data. The theory of stationary stochastic processes is much more developed than the theory of non-stationary processes. The existence of trends, or non-stationarity, in the data, invalidates many of the statistical tests that can be applied. The SASE IV program has tests for uniform trends and indications of cyclic trends as well.

Tests for trends have been applied to access interval sequences of files in the trace data. Though there are exceptions, the typical file does not show any linear trend in its inter-access intervals. In cases where trends are identified, they are almost always caused by the presence of one long idle period before the deletion of the file. In other words, those files that do show a trend tend to be accessed frequently for a period after they are created and then fall idle for some time before they are deleted.

In the absence of trends, other tests may be applied. The next question to be answered is whether there are serial correlations in the

data. If the intervals are serially correlated, then the relatively simple renewal process model is not applicable and other, more complex models must be investigated.

The analysis program tests for serial correlation or independence of intervals by computing the correlation coefficients of adjacent intervals (and those separated by a fixed number of intervals) and testing their closeness to zero (the value of correlation coefficients for independent intervals). Other tests based on the spectral density function of intervals were also used. These tests are only strictly applicable to intervals whose distribution is normal, not the case here as we shall see, but the results are generally so strong that the conclusions drawn seem to be warranted.

The tests show that, again in the vast majority of cases, inter-access intervals are not correlated. The indication of independence of intervals is usually quite strong, though in some instances some serial dependence is shown. This is an important result with implications in building a model of file activity and for understanding migration strategies. It means that we can characterize the accessing process of a typical file as a renewal process, without any further consideration of possible dependency structure. We discuss some implications of this result in the next section.

It remains to determine the distribution of interval lengths (idle times) to completely characterize the accessing process. The general shape of this distribution resembles an exponential distribution. The curve fitting program indicates that exponential, Pareto, Weibull, or hyperexponential distributions may be appropriate. The most attractive alternative, from an analysis point of view, is that the intervals be exponentially distributed. Several tests are provided in the statistical analysis program for determining if a series of intervals are exponentially distributed. Again, of course, the result does not hold for all files. However, the large majority of files do pass the test indicating that their inter-access intervals can be modeled by an exponential distribution. A renewal process whose intervals are

exponentially distributed is called a Poisson process (the number of events in a given time interval is Poisson distributed).

We thus have the following model for accesses to a typical file. The rate of accessing is constant (stationary process), the lengths of successive idle intervals are independent (renewal process), and the distribution of inter-access intervals is exponential (Poisson process). Although this characterization is not true for all files, it is very often applicable. Restricting our discussion to a theoretical population of files with Poisson accessing processes is a useful and not unwarranted simplification.

We have not mentioned the accessing rate of a typical file. Although the structure of files' accessing processes are the same, the accessing rate (or mean inter-access interval) varies from file to file. Thus, files' accessing patterns, though described by Poisson processes, are not identical. How the accessing rate varies with file size is discussed in section 3.6.

3.4 Implications of the Accessing Process Model

The characterization of the access process of a file as a Poisson process will influence how we choose migration algorithms in Chapter V. It especially affects how we make use of the concept of locality that is so useful in the study of page replacement algorithms. In this section, we consider implications of the Poisson process on some higher level aspects of the file system, in particular, the accessing pattern of libraries and the accessing pattern of the file system as a whole.

From a systems design point of view, a characterization of the accessing process of the whole file system is much more useful than the processes for individual files. Measures such as accessing rate and number of accesses per day are of interest. Since the file system is the collection of all the individual files, its accessing process is the superposition of the individual accessing processes. The process

consists of the combined output of the component processes. Some mathematical results are available for superposed renewal processes.

When the component processes are Poisson, their superposition is also a Poisson process. So if we model each file's accessing as a Poisson process, the accessing process of the entire system will also be Poisson. Even when the component processes are not Poisson, the superposition of a large number of independent (but not necessarily identical) processes tends to be Poisson. This result is from Khintchine [Kh60]. He proves that in the limit, the superposition of a number of independent renewal processes is a Poisson process. The result is true even in the case that the component renewal processes have different interval distributions. Khintchine's theorem helps explain why the Poisson process seems to describe very well a number of processes that occur in real life. Khintchine points out, for example, that telephone calls arriving at a telephone switching machine are the combined output of a large number of independent processes, i.e., the calls made by individuals, and so can be expected to be Poisson. In our case, we might expect that the pooled accesses to all files can be described by the Poisson process. We investigate here whether this is true.

First we point out that the assumptions needed for Khintchine's result may not strictly hold. The theorem deals with a fixed population of component processes, while the file system is a dynamic population. The number of files in the system varies from day to day. This variation is a small percentage of the total population (.1% per day), but could perhaps affect the validity of this result. Another more subtle problem is the independence of component processes required in Khintchine's theorem. To some extent, this assumption does not hold. Users sometimes access (and create and delete) a set of related files together so that their access processes are dependent and possibly synchronized. The total file population, however, is spread over more than 700 users, mostly using their files independently, so the effect of this violation of assumptions is probably negligible.

Unfortunately, we cannot accurately measure the accessing process of the entire file system from the data collected. Because of this, it is impossible to directly verify the hypothesis that accesses to the file system as a whole are Poisson by applying tests for Poisson processes. The reason that we can't measure the accessing process of the whole system is outlined below, and is a result of the trace data being measured on a time unit of one day. This means that for each file, we have an indication of whether or not it was accessed on a given day, not how many times it was accessed on that day. This implies that for the entire file system, we cannot get a count of the number of file accesses each day but only a count of the number of distinct files that were accessed each day. The empirical distribution of this data, the count of separate files accessed per day, has been gathered and plotted (Figure 3.1). This distribution is distinctly non-Poisson. This is shown most directly by comparing the sample mean and sample variance. The sample mean is 678 and the sample variance is 14,400, whereas for a Poisson process, the mean is equal to the variance. The appropriate discrete probability function, when the variance is greater than the mean, is the negative binomial distribution,

$$f_n(x) = \binom{x-a-1}{x} p^x (1-p)^a$$

The negative binomial distribution fitted to the data by the method of moments is also shown in Figure 3.1 (solid curve).

The distinction between the number of distinct files accessed per day and the total number of accesses per day is very important. The fact that the distribution of files accessed per day is a negative binomial is not inconsistent with the prediction that the number of accesses per day is Poisson. The following paragraphs outline a construction of the negative binomial distribution from the assumption that individual file accesses are Poisson.

We assume that the accessing pattern to a single file is Poisson. This has been demonstrated above and is valid on a macroscopic level when the file is viewed over a period of days. Our grain of measurement

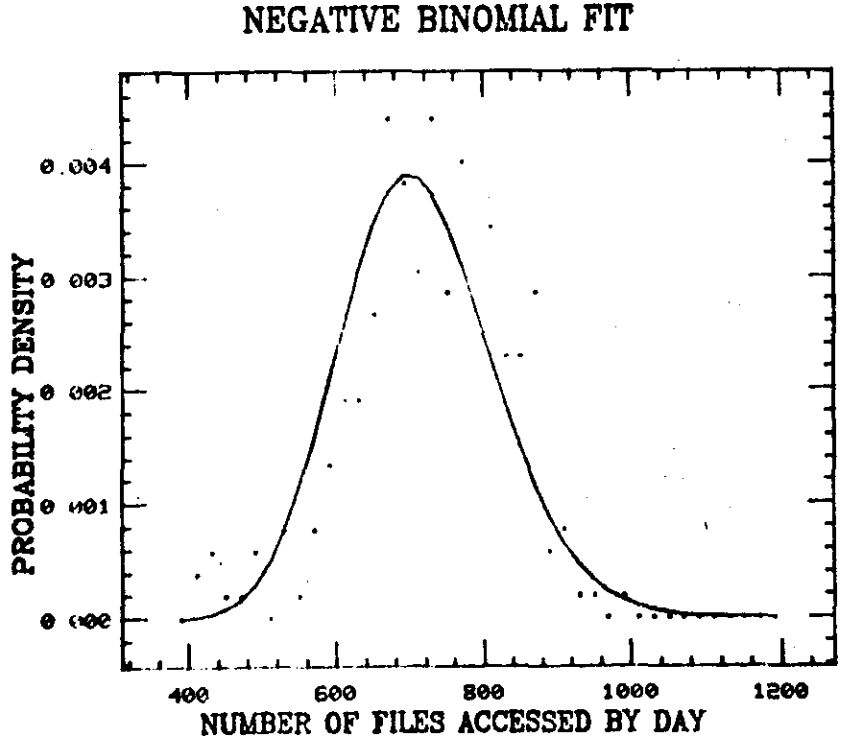

FIG. 3.1

of time (one day) affects what a file's accesses look like in the trace data at a more microscopic level. For each day, the trace data only show whether or not a file was accessed. Since the Poisson process model assures that the probability of access of a file in one day is independent of accesses on other days, we can think of the access to a file in a day as a Bernoulli trial (takes either the value 0, no access, or 1, some access). Bernoulli trials imply that, at this level where time is measured in discrete units of one day, the inter-access intervals of a file are geometric distributed and its access process is binomial. (The discussion earlier in this chapter dealt with time over a larger range where, though measured in days, it could be thought of as continuous. In continuous time, of course, the geometric intervals become exponential and the binomial accesses become Poisson.)

Within each day then, we have a series of N Bernoulli trials where N is the number of files in the system. The parameters of these various Bernoulli trials are not equal, being the accessing probabilities of the individual files. Feller [Fe50], page 263, proves that the number of sucesses (here interpreted as file accesses) in a sequence of Bernoulli trials with variable probabilities (also called Poisson trials) is described, in the limit of a large number of trials, by the Poisson distribution.

The final step in this construction is to note that the number, N, of files in the population varies from day to day according to an Erlang distribution (Figure 3.2, the Erlang being a special case of gamma distribution). Since the rate parameter of the Poisson distribution of files accessed in a day is proportional to the population, N, which is gamma (Erlang) distributed, the distribution of files accessed per day is a so-called compound Poisson distribution (Poisson randomized with gamma). A simple proof, given in Appendix A, shows that the compound Poisson randomized with Erlang rate parameter is the negative binomial.

Thus, we see that the model of individual files is consistent with the experimental determination of the distribution of files accessed per day being negative binomial. Furthermore, a physical interpretation can

ERLANG FIT

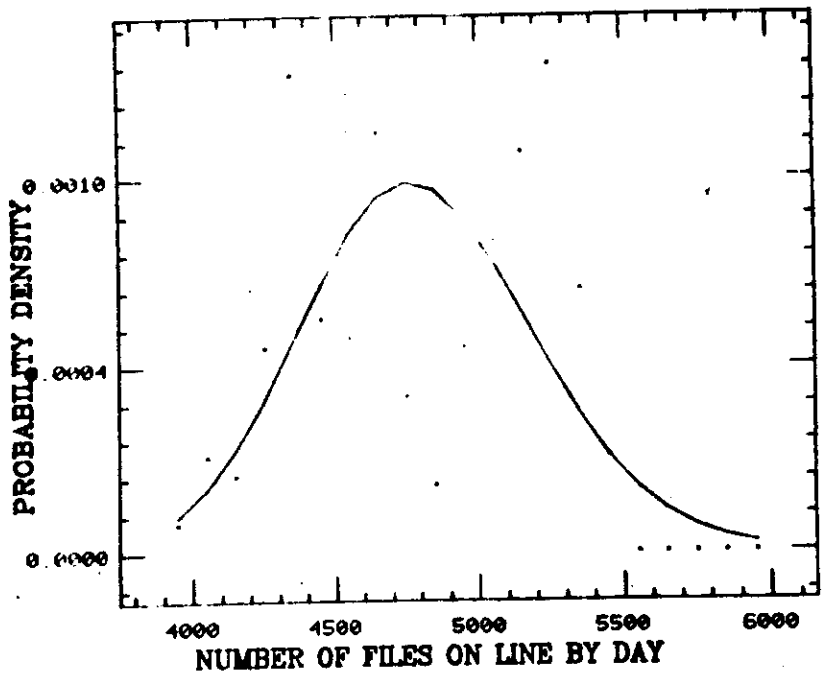

FIG. 3.2

be given to the negative binomial distribution.

3.5 Accesses to Libraries

Modeling the typical file's accessing process with a Poisson process also has implications to our understanding of libraries. A library is a collection of small files that is dealt with in the file system's directory structure and storage allocation as a single contiguous unit. From the trace data, we cannot distinguish accesses to individual member files of a library. Extending our model of individual file accesses as Poisson processes, we can postulate that member files in libraries are also accessed according to Poisson processes. As before, the accessing to the whole library, as the superposition of Poisson processes describing the accessing to library members, is also a Poisson process. This is borne out by the trace data in most cases. The statistical analysis program generally shows that access intervals to libraries are trendless, independent and exponentially distributed, indicating that the underlying process is Poisson. In those libraries where this is not true, the time grain of one day may be having an effect again. For actively used libraries where the mean idle time is near one day, the trace data may show an insufficiency of small intervals (since the smallest interval the data can record is one day) causing rejection of the Poisson hypothesis in statistical tests [Co66]. Also, in libraries where the component files are often used together, the independence assumption for superposition of accessing processes may be violated, causing a library's overall accessing to be non-Poisson.

3.6 Distribution Fitting for Inputs to the Model

As discussed in Chapter IV, a model has been constructed to generate artificial trace data which simulate the actual trace data. The model's inputs are a number of probability distributions derived from various empirical distributions describing the real data. This section discusses the distribution fitting process and illustrates the results. Although the purpose of this distribution matching was to provide input for the model, the distributions characterize the file

system activity and so their study increases our understanding of this activity. The distribution fitting program plots the input data against various standard probability distribution functions. The parameters are matched by standard methods, usually method of moments, or maximum likelihood estimation. Both the density functions and the hazard functions are plotted.

The model assumes that lifetimes of files are independent. Artificial lifetimes can, therefore, be generated by sampling the probability distribution of lifetimes of files. Figures 3.3a and 3.3b show the empirical distribution of lifetimes plotted with the corresponding fitted exponential distribution. This is the distribution used in the model to generate lifetimes of files. The mean is 257 days.

A related model input is the ages since creation (again in days) of files in the system at the start of the measurement period. Again we find that this distribution can be modeled by the exponential (Figures 3.4a, 3.4b). This distribution is needed by the model to create a starting population for simulation.

Clearly, there is a relationship between the age distribution and lifetime distribution. By sampling the age of a file at a given time, we obtain what is called the backward recurrence time of the file's lifetime. This concept was developed for renewal theory [Co62]. A distribution, f(x), is related to the distribution, r(x), of its backward recurrence times by $r(x) = (1-F(x))/m$ where F(x) is the cumulative distribution of f and m is the mean of f. For the special case here of exponentially distributed lifetimes

$$r(x) = 1-(1-e)^{\lambda x}/(1/\lambda) = \lambda * \exp(-\lambda x) = f(x)$$

So we expect the age and lifetime distributions to be identical exponentials. In fact, the mean age is 276 days and the mean lifetime is 257 days. Some of this discrepancy may be explained by the non-infinite measurement period limiting the measurement of some long lifetimes.

EXPONENTIAL FIT

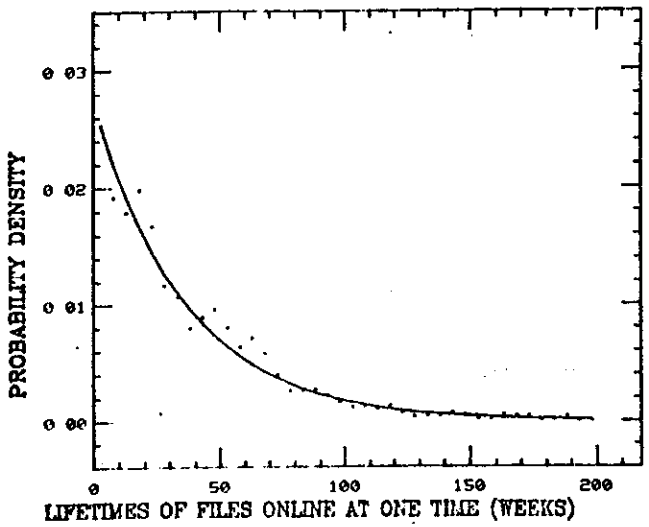

FIG. 3.3A

EXPONENTIAL FIT

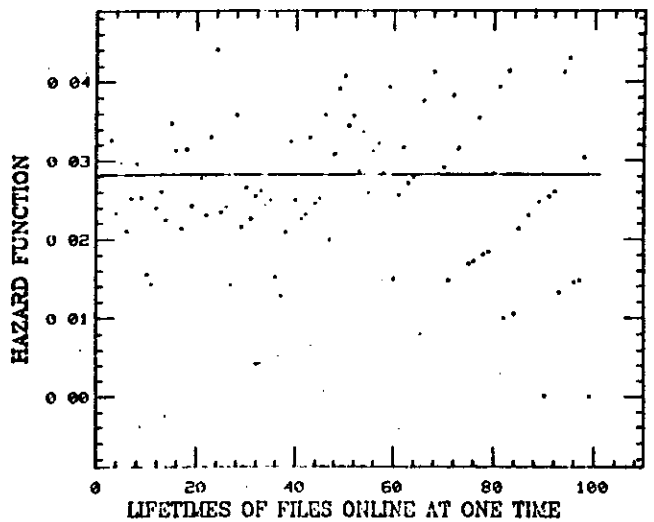

FIG. 3.3B

EXPONENTIAL FIT

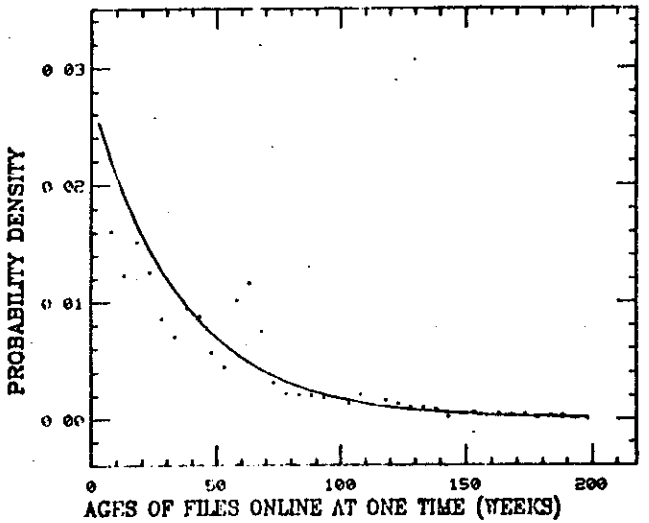

FIG. 3.4A

EXPONENTIAL FIT

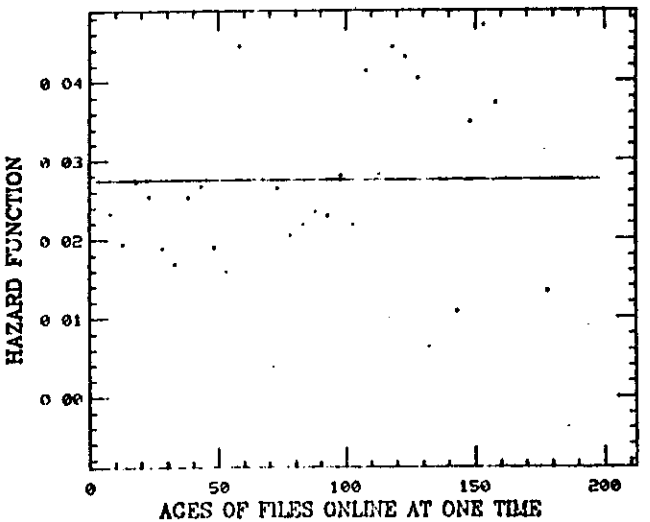

FIG. 3.4B

The size distribution of files in the system is shown in Figure 3.5a,b,c. This distribution is very strongly skewed to small file sizes. The exponential distribution fits poorly and we use, instead, the Pareto distribution,

$$f_p(x) = cx^{-c-1}$$

The size distributions of files created and files deleted are practically identical to the size distribution of files in the system. This means that the size mix of the file population is stable, not changing with time.

The size distribution of files accessed also has Pareto shape but is substantially less skewed toward small files. The mean size of files in the system is 8.4 tracks, but the mean size of files accessed is 19.9 tracks. This means that large files are accessed more frequently than small files. Figure 3.6 shows how the average inter-access time for files varies with file size. The distribution of inter-access intervals for specific files has been discussed earlier as exponential. A plot of mean inter-access intervals for files of a given size is shown in Figure 3.7. Either a Pareto or a Weibull distribution

$$f_w(x) = cx^{c-1} / b^c * e^{-(x/b)^c}$$

is appropriate here, and for other file sizes as well.

3.7 Dependency Relationships of Access-time Intervals

In Chapter V, various migration strategies are discussed. The migration strategies may be thought of as using different predictors of the next idle interval of a file. For instance, LRU (least recently used) uses the most recent idle time as a predictor of the next idle interval. LFU (least frequently used) uses the average idle interval (or accessing rate) to predict the next idle interval. One purpose of our analysis of the trace data should clearly be to discover any dependency relationships that a file's inter-access intervals might

EXPONENTIAL FIT

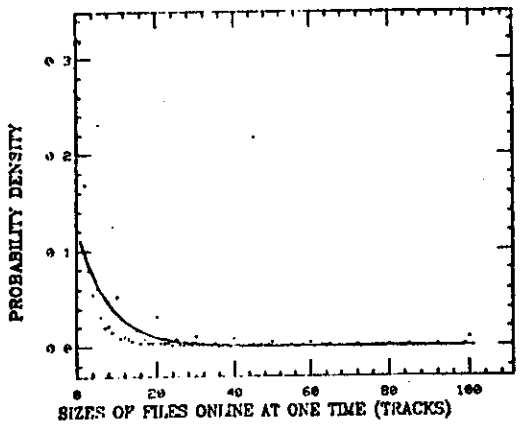

FIG. 3.5A

PARETO FIT

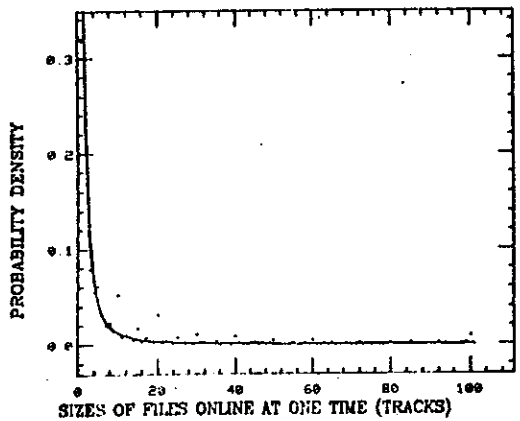

FIG. 3.5B

PARETO FIT

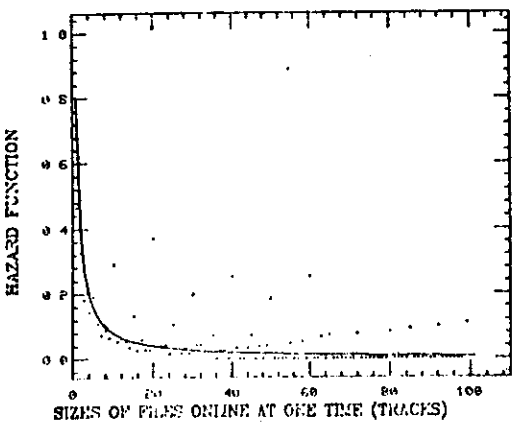

FIG. 3.5C

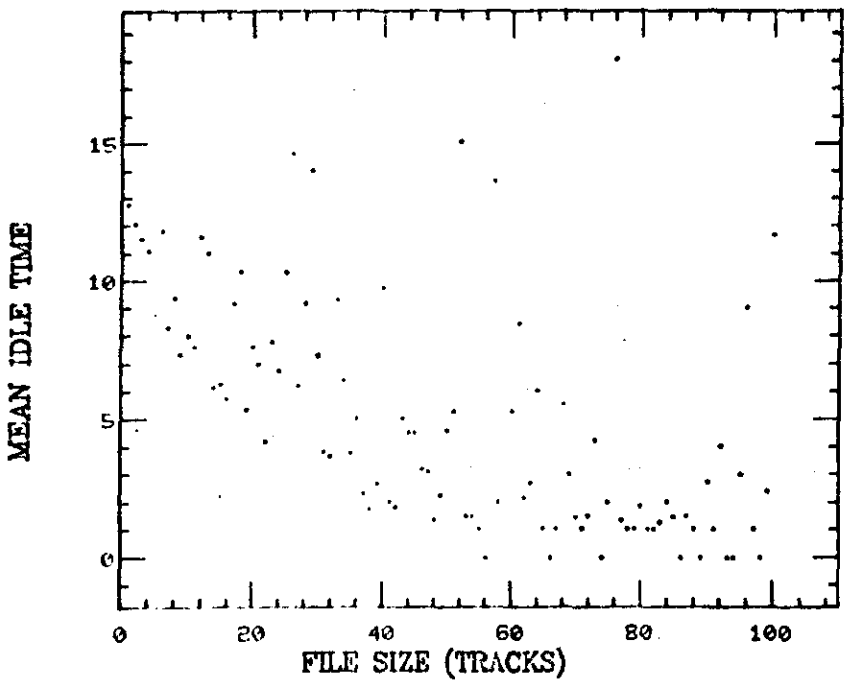

FIG. 3.6

WEIBUL FIT

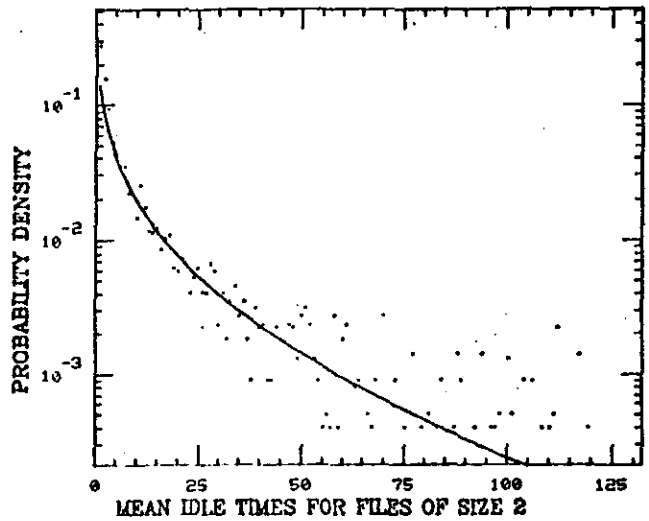

FIG. 3.7A

PARETO FIT

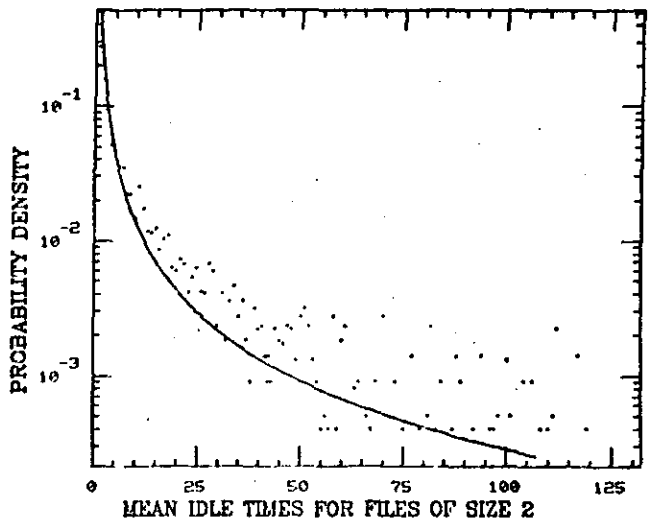

FIG. 3.7B

show. If inter-access intervals are correlated with any file characteristic that is easily measurable, we may be able to use that measure to help predict idle intervals.

The most obvious correlation would be between successive interaccess intervals. It has been demonstrated that the accessing process is a renewal process; therefore, that previous idle times are not correlated with subsequent inter-access intervals.

The inter-access interval distribution of a file is influenced by other factors. The previous section showed (Figures 3.6, 3.7) how the size of a file is correlated with its accessing rate. Another possibility is that a file's accessing rate depends on its lifetime in the system (or its current age in the system). Figure 3.8 plots this relationship. With the exception of files with very short lifetime, and several outlying points where the number of files in the system of that size is very small, there appears to be very little correlation between lifetime and accessing rate (inter-access time). Chapter V discusses the design of migration strategies using the dependence of inter-access intervals on file size and their independence on file age, and previous inter-access intervals.

3.8 Summary

This chapter describes statistical analysis of the empirical file trace data. First a model of accesses to a typical individual file is developed. The Poisson process is shown to be appropriate. This conclusion is consistent with the observation that the number of files accessed per day is described by the negative binomial distribution. Accesses to libraries are discussed and again the Poisson process is the model used.

Results from curve fitting experiments are given. The age and lifetime distributions of files are described by the exponential distribution. File sizes vary according to a Pareto distribution. Finally, the dependence of file accessing rate on other file

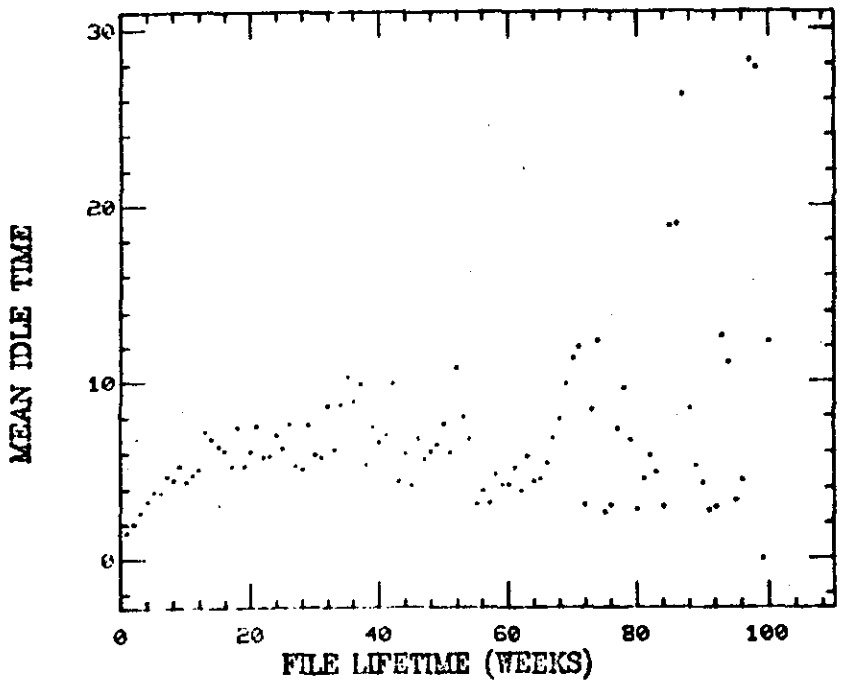

FIG. 3.8

characteristics is investigated. The Poisson accessing model implies that there is no serial correlation of inter-access periods. The expected value of the inter-access time is shown to be independent of the file's lifetime in the system but correlated to the file's size.

CHAPTER IV

SIMULATION MODEL

The trace data described in Chapter II provide the basis for simulating file system activity used in the study of migration strategies presented in Chapter V. The data could also be useful in many other studies of file systems, file structures, and storage hierarchies. File system data can be expensive and time consuming to collect, however. An alternative is to build a computer program model of file system activity which will generate artificial file system trace data. This chapter describes such a simulation model, its structure, its validation, and possible uses.

4.2 Basic Structure

A file system model can have varying complexity depending on the level of detail required in the artificial trace data to be generated. For instance, some studies would only require a simulation which gave various system-wide activity figures, such as the number of files accessed by day, number of files deleted by day, etc. In this case the model might consist merely of a set routines that generate values for files accessed per day, etc. from the corresponding fitted distributions. For this work on file migration however, much more detail is needed. The migration algorithms to be studied deal with individual files, making migration decisions based on the characteristics and past accessing history of each specific file. A macroscopic model which only deals with number of files accessed, etc. cannot supply the necessary information. A detailed micromodel is required which simulates the activity of individual files.

Another reason for using a detailed model is that it requires a much better understanding of the characteristics of individual files, of the dependencies of these characteristics on other file characteristics, and of the relationships between files. This increased level of knowledge and detail makes the model useful for many types of experiments in which the model's structure or its inputs are modified in

order to simulate different environments. Examples of possible experiments are listed in a later section of this chapter.

The model built for this study is detailed to the level of individual files. The file system is viewed as a collection of independent files. Each file is individually characterized by its size, lifetime, and its accessing trace. A population of files is assumed to exist at the start of the simulation and for each simulated day a number of new files are created. No assumptions are made about the number or distribution of files or tracks accessed per day, of files or tracks in the system at a given time (except at the start of the simulation) or of files deleted per day. These global values are generated as the simulation proceeds, resulting from the simulated accesses to individual files and the simulated deletions of files as their "lifetimes" run out.

The inputs to the model are

-size distribution of files
-inter-access interval distribution of files
-lifetime distribution of files
-distribution of new files per day.

Given these distributions the program simulates file system activity and generates artificial file accessing trace data.

4.3 Development of the Model

Development of the simulation model was done in two steps. The first step was to construct a valid model using as inputs the empirical distributions derived from the actual trace data. This permitted building the basic model structure and discovering the important dependencies, with known, valid input distributions (in the form of tables of values). The second step was to replace the empirical input distributions with analytic formulas for standard probability distributions fitted to the empirical data.

Major iterations in building the model were necessitated by the discovery of three important dependencies. First it was found that the lifetime of a file depends on the file's size. The model was adjusted so that smaller files have shorter lifetimes than large files. A more

important dependency is the relationship between file size and the accessing rate of files. The distribution of sizes of files in the system is different from the distribution of sizes of files that are accessed, large files are accessed relatively more often than small files. Furthermore, the accessing of files of a given size is randomized so that all files of that size are not accessed at the same rate. The procedure finally used in the simulation model is to pick, given a particular file, the mean accessing rate of files of this size. The mean accessing rate for the particular file in question is then generated randomly from an exponential distribution with mean equal to the mean accessing rate for this file size. This assures that files of a given size do not all have the same mean accessing rate. The inter-access intervals for the file are then generated randomly from the appropriate distribution (exponential) whose mean is the derived accessing rate for that particular file.

Another factor that significantly affects a file's characteristics is its type. The only type distinction of interest here is whether or not a file is a library (collection of smaller files). The existence of libraries is quite evident in the trace data described in Chapter II. Libraries are larger than most files, they are preallocated by their owner so that their sizes tend to be multiples of five tracks, they are accessed much more often than ordinary files and they are rarely created or deleted. Accordingly the model includes a small subset of files that simulate libraries. Their lifetimes are infinite, their sizes are multiples of five, and their accessing intervals are exponentially distributed with small mean.

4.4 Validation

Validation of a simulation model of this complexity is a difficult and largely subjective process. No attempt has been made to mathematically demonstrate the applicability of the model, although the model is based on demonstrated mathematical properties (distributions and inter- relationships) of the real file system data.

Each iteration of the model was validated using two computer programs. The most useful was a program mentioned in Chapter II which extracts daily activity plots and distributions of pertinent file characteristics from trace data input. The output of this program when run on the trace data generated by the model was compared to corresponding output from the empirical trace data. The program was designed to demonstrate any differences in the traces. For instance trends (increasing or decreasing) in such quantities as files in the system, files accessed, etc. or imbalances (too many large files, small files accessed too often, etc.) were easy to detect. The second method of validation was to run the various migration strategy simulations discussed in Chapter V on the artificial trace and compare the performance to that of the real trace data. The performance of migration strategies depends very closely on the accessing patterns of individual files, so matching the migration performance of the artificial trace data to that of the real data inspires high confidence in the validity of the model.

4.5 Uses of the Model

The artificial trace data generated by the model could be used in place of empirical trace data whenever the latter is not available. Studies of migration strategies (as in Chapter V), evaluation of proposed storage hierarchy changes, projections of future file storage requirements are examples of uses of this type of trace data.

There is another very useful application of the simulation model in which the real trace is not useful. Because the model simulates the structure of the file system activity, not just its gross properties, it can be used to simulate related file systems with slightly different structure. This could be done by changing various input distributions to the model or dependency assumptions used by the model to reflect the hypothetical file system environment. For instance, the model could simulate environments with half (or twice) as many users, or environments in which there are no libraries or where there is serial dependence structure to the accessing patterns of files. The resulting

trace data could be analysed to show the activity properties of the hypothetical file system environments. An interesting experiment would be to study the sensitivity of the behavior the various migration strategies studied in Chapter V to changes in file system activity.

CHAPTER V

FILE MIGRATION

This chapter applies the understanding of file system activity developed in previous chapters to the problem of file migration. Secondary storage, the highest non-executable level in the storage hierarchy of a computer system, is treated as a scarce resource. It is desired to manage this resource by transferring files to and from the lower levels of the hierarchy (back-up storage), so as to maximize the probability that the next file to be accessed is stored in secondary storage. The transfer of files between secondary storage and back-up storage is referred to as migration. The implementation of the management of secondary storage, that is, of migration, involves physical transfer of the files and proper maintenance of directory structures by operating system routines. This implementation is the migration algorithm. The process of deciding which files should be kept in secondary storage and which files should migrate is called the migration strategy. This chapter discusses the migration algorithm and introduces and evaluates various migration strategies.

The motivation for file migration can be either to decrease computing cost or increase computer performance. Proper management of secondary storage can reduce system costs by requiring less secondary storage capacity to achieve the same level of performance. Alternatively, effective migration can increase system performance by decreasing the amount of input from back-up storage, at the same level of secondary storage capacity.

Figures 5.1a and 5.1b provide a clear indication that file migration can be useful. Figure 5.1a is the same as Figure 2.3b. It shows the fraction of total file space that is occupied by files that have been idle less than any given length of time. Figure 5.1b gives, for each idle time, the "hit ratio" or the fraction of file accesses for which the file has been idle for less than the corresponding idle time. From Figure 5.1a, for instance, we see that 20% of the file space used is occupied by files that have not been accessed for 30 days. Figure

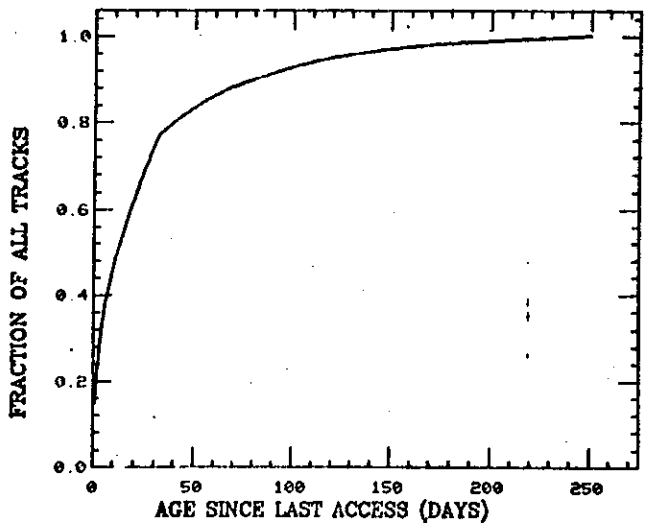

FIG. 5.1A

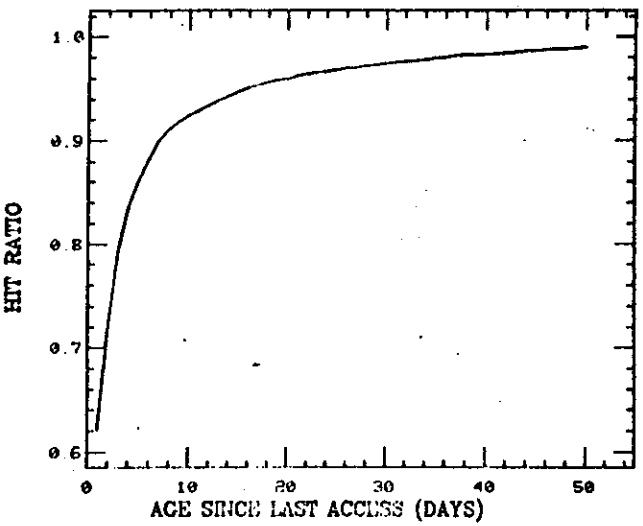

FIG. 5.1B

5.1b shows that 98% of all accesses are to files that have been idle for 30 or fewer days. This means that a simple migration scheme could save 20% of the secondary storage cost with a very small (2%) performance penalty, by causing files that stay idle for more than 30 days to migrate to back-up storage. If a bigger performance penalty is acceptable, say 10%, then files would migrate when they were idle for more than five days and 45% of the secondary storage capacity could be saved. This simplified argument hides the fact that a 90% hit ratio eans that 10% of all file accesses cause input from back-up storage. Depending on the access time and transfer rate of the back-up storage medium, this may cause an intolerable wait time or load on the input/output system.

5.2 The Implementation of File Migration

The migration algorithm, or in other words, the practical details of implementing file migration, is discussed only briefly here. The migration programs must cooperate with the file system and input/output programs, and so depend on their specific structure. Several principles are clear, however.

The migration algorithm implements automatic migration of files between secondary and back-up storage. It relieves the user of knowledge of where his files are kept and assumes responsibility for managing all levels of the storage hierarchy, not just secondary storage. This implies that while the directory structure in many current file systems only describes the set of files in secondary storage, a directory structure with automatic migration must include pointers to all files in the storage hierarchy. In other words, when a file in secondary storage is selected for migration, its directory entry is not erased. The directory entry must be kept but modified so that the address pointer in the entry points to the new location on back-up store. Similarly, when a file migrates from back-up store to secondary storage, its directory entry will already exist and will need only to have the address field updated.

The migration algorithm also implements the process of deciding which files should migrate. Selection of a suitable decision making process, or migration strategy, is discussed in the next sections. These migration strategies make decisions based on some information about files' past histories or physical characteristics. For instance, LRU replacement uses the current idle period of each file as a parameter in its decision. Other strategies to be discussed use the average idle interval of a file, the file size, etc. The file directory entries must be changed to add fields to hold the information needed by the migration strategy. Further, these fields must be updated whenever the corresponding value changes (usually when a file is accessed).

If secondary storage is kept full, then each time a file migrates from back-up store to secondary store, one or more files in secondary store must migrate to back-up store to free space for the new file in secondary store. To avoid having to search through the directory entries of all files in secondary storage to determine which should migrate out, some type of ordered list of the files should be maintained. For simplicity this discussion assumes only two levels in the storage hierarchy. If there are more levels, an ordered list of files is needed for all but the lowest level. For LRU replacement, for instance, the files are ordered by current idle time and the file with the longest current idle time is selected for migration. Other replacement strategies use other measures on which to order the files, but for each strategy to be discussed there is a unique ordering of the files.

A sorted list of the files is not completely necessary however since only the first element of such a list (the file to be migrated) needs to be readily accessible. A partial ordering such as is provided by a "heap" is all that is needed. Heaps, which are discussed in Knuth [Kn73], provide a very efficient mechanism for maintaining partially ordered lists. When a file is accessed or its place in the ordering changes for any other reason, its directory entry must be removed from the heap and re-inserted in sort order. For LRU replacement, the accessed file is inserted at the bottom of the heap. For other

replacement strategies, insertion may be required other than at the head of the list, and some searching through the heap may be necessary. All the migration strategies to be considered in this work have the characteristic that the sort measure of a file (i.e., current idle time, or average idle time, etc.) can only change when a file is accessed. This important property implies that when a file is accessed, other directory entries in the heap do not change their relative sort order. Only the directory entry for the accessed file need be removed and re-inserted.

This added structure of linking the directory entries on a sorted list can be avoided if the secondary storage is not kept completely full. If at the beginning of each day, sufficient free space is allocated in secondary storage to hold all newly created files and all files that migrate from back-up storage during the next day, then the migration of files to back-up storage can take place late at night during the system's minimum load time. Many systems run accounting routines during this time which scan all the directory entries in the file system. If a complete scan is being made anyway, the most likely candidates for migration can be identified at this time and all the migration to back-up storage that is required to free space on secondary storage can take place during low usage time.

The migration algorithm need have very little impact on the file system code. The directory entries must be changed, as discussed, to point to files on back-up storage as well as secondary storage, and to hold additional information needed for the replacement strategy. Migration into secondary storage can take place on demand. If sufficient free space is allocated in secondary storage, then migration to back-up store can be delayed until low usage time when its impact on the system will be small.

5.3 Evaluation of Migration Strategies

The remainder of this chapter deals with a number of possible migration strategies. This section describes the simulation and evaluation of these strategies. Each strategy to be studied is built into a migration simulation program. This program simulates the running

of the migration algorithm discussed in the previous section, using the migration strategy under study as the decision making process. Each simulation is run using the empirical trace data discussed in Chapters II and III as input. Since the input to the simulation is real trace data, as opposed to artifically generated data, the results of the simulations are identical to the results that would be obtained by running the corresponding migration strategy on a real computer. The outputs of the simulations, which provide an indication of the effectiveness of the particular migration strategy, are discussed below.

The output from such a simulation program should give a measure of the performance of the migration strategy in question that can be compared with the corresponding measure for other migration strategies. Two performance measures are used here for evaluation of migration strategies. The first is the "miss ratio". This number gives, for a given secondary storage capacity, the fraction of file accesses that do not find the accessed file in secondary storage, under the migration strategy in question. That is,

$$\text{miss ratio} = \text{accesses to back-up storage} / \text{total accesses}$$

where

$$\text{total accesses} = \text{accesses to files in secondary storage} + \text{accesses to files in back-up storage}$$

also

$$0 \leq \text{miss ratio} \leq 1$$

Clearly, the migration strategy that gives the lowest miss ratio, for a given secondary storage capacity, is preferred. The closer the miss ratio is to zero, the fewer the number of file that accesses fail to find the required file in secondary storage. Each such failure, or "miss", causes a file to migrate to secondary storage from back-up storage, causing additional load to the input/output system and greatly

increasing the effective access time to the file. The miss ratio thus gives a measure of how effectively the particular migration strategy manages secondary storage in the sense of keeping the most used files in secondary storage and reducing the number of file transfers from back-up storage.

The second evaluation measure is migration traffic. The migration traffic gives the average volume of input/output transfers per time unit (tracks per day) needed for migration with a given secondary storage size, under the specified migration strategy. In a page replacement environment where pages are all the same size, the miss ratio and I/O traffic are equivalent measures (traffic is proportional to the miss ratio). This is because each miss, or page fault, requires the transfer of one page to main memory, so the same amount of I/O traffic is required for each page fault. In the file migration environment, this is not true. Since files are not all the same size, the migrations of different files may cause different amounts of input/output traffic. A migration strategy with a low miss ratio may be a worse strategy with respect to migration traffic than another strategy with a higher miss ratio. As a trivial example, suppose that one strategy results in only one miss, to a file of three tracks. Another strategy causes two misses, each to files of one track. The former strategy has a lower miss ratio but the latter causes less migration traffic.

The two performance measures for file migration strategies, miss ratio and migration traffic, are thus not equivalent. Both will be used in this chapter to evaluate the migration strategies presented. Which measure is more appropriate will depend on the implementation used for migration. In most situations the migration traffic load is easily handled by the data path between back-up and secondary storage. The miss ratio evaluation measure is more important in these cases, since it directly affects the average access time to files. If the data path for migration is saturated because it has very small capacity or very high use from other sources, then it may be more crucial to minimize migration traffic than the miss ratio. The difference is explored

further in Chapter VI. This discussion of evaluation of migration strategies has purposely avoided reference to actual costs and specific memory hierarchies and devices. The miss ratio and migration traffic measures do not depend on these installation-specific measures. The actual costs of a migration system can be derived from the miss ratio and migration traffic measures. See for example papers by Lum et al. [Lu74] and Kimbleton [Ki72] where the miss ratio and migration traffic are assumed to be known, and specific costs are derived. Chapter VI of this thesis also gives some examples.

5.4 Stack Algorithms and Stack Processing

Both the miss ratio and the migration traffic measures described above are functions of the secondary storage capacity. For comparison of two migration strategies, it is extremely useful to have performance measure curves, that is, to know the value of the performance measure for every possible secondary storage capacity. It is possible, using a technique first described by Mattson et al. [Mi70], to derive the entire performance measure curve in one pass through the input trace data. Mattson et al. call this method stack processing and describe the set of replacement algorithms, called stack algorithms, for which stack processing is valid.

A simulation program that uses stack processing maintains for each instant of simulated time an ordered list (called the "stack") of all elements (pages or files) that have been accessed. The list is ordered on a sort measure which is unique to the replacement strategy being used. For instance, for LRU replacement, the elements of the list are ordered by their current idle times. Whenever the sort measure of an element changes (in the cases considered here, this only occurs when the element is accessed), the list must be resorted so that the ordering is maintained. The sort measure used is precisely the measure that the corresponding replacement strategy would use in deciding which file should migrate when secondary storage becomes full. Associated with each element on the sorted list is a "depth". The value of the depth of an element is the sum of the sizes of all list elements ahead of and

including the current element. Because of the nature of the ordering of the list, the depth of an element has an important physical interpretation. The list is sorted so that all the elements ahead of a given element currently have a higher priority of being in secondary storage. Thus, if the element in question were stored in secondary storage at a given time, then all the elements ahead of that element would also be contained in secondary storage. The depth of a list element thus gives the minimum secondary storage capacity, such that the element would be in secondary storage, under the migration strategy in use, at this point in simulated time.

A "stack algorithm" is a replacement strategy which, at each moment in time, defines a total ordering of the elements that have been accessed. This property is necessary (and sufficient) for the proper operation of stack processing. All of the migration strategies considered in this work are stack algorithms. A stack algorithm has the property that the miss ratio curve is monotone decreasing as the size of secondary storage increases. This means that increasing the size of secondary storage cannot cause worse performance (more misses). The same property, monotonicity, holds for migration traffic curves of stack algorithms. Mattson et al. [Ma70] give an example of a replacement strategy, which is not a stack algorithm, for which this may not be true.

Two important changes were made in the stack processing algorithm for paging to apply it to file migration. First, in paging, the stack depth of a page is simply the page size (which is fixed) times the number of pages with higher priority (i.e., ahead of it on the sorted list). Therefore, all that is needed to compute a page's depth is its ordinal position in the list. Files, by contrast, are variable in size and the simulation must maintain for each file, not merely its ordinal position, but the sum of sizes of all files ahead of it in the sorted list. A second change was necessitated by the nature of the input trace data, but has the side effect of substantial savings in computer time for the simulations. The input data indicates whether or not each file was accessed on a given day. There is no indication of the order in

which files were accessed within each day. Therefore, the accesses to files for each day cannot be distinguished and can be treated as if they all occurred simultaneously. The simulation program, therefore, only reorders the sort list and re-computes depths after all accesses for a given day have been processed.

The stack processing simulation program maintains an ordered list of all files that have been accessed and associates a depth with each file in the list. Whenever the input trace data indicate that a simulated access is being made to a file, the current depth of that file is recorded. Thus, as the simulation proceeds, a histogram of depths is developed, where for each possible depth the number of accesses to that depth is recorded. The miss ratio curve is generated very simply from the depth histogram. When the histogram is normalized, it becomes the depth density function, giving for each depth the probability of access to that depth. The corresponding probability survivor function gives for each depth the probability of access to depths greater than the given depth. This function, interpreted correctly, is the miss ratio function. When the abscissa is thought of as representing secondary storage capacity, then the curve gives for each secondary storage capacity the probability that an access is to a file not stored in that secondary storage capacity. This is exactly the miss ratio as described previously.

Two sample miss ratio curves are given in Figure 5.2a, one dotted, one solid. Clearly, the strategy which generated the dotted curve is preferred since its miss ratio is lower for every secondary storage capacity. The most interesting area of comparison is at low miss ratios (close to zero) since this is where a migration system would probably be tuned to run. To show this region of the curves better, we perform a simple transformation. The transformation is to show the miss ratio on a logarithmic scale in order to expand the interesting region near zero. This is shown in Figure 5.2b. This is the type of curve we will use for evaluating migration strategies with respect to miss ratio.

The other performance measure we will use is the migration traffic

LEAST RECENTLY USED

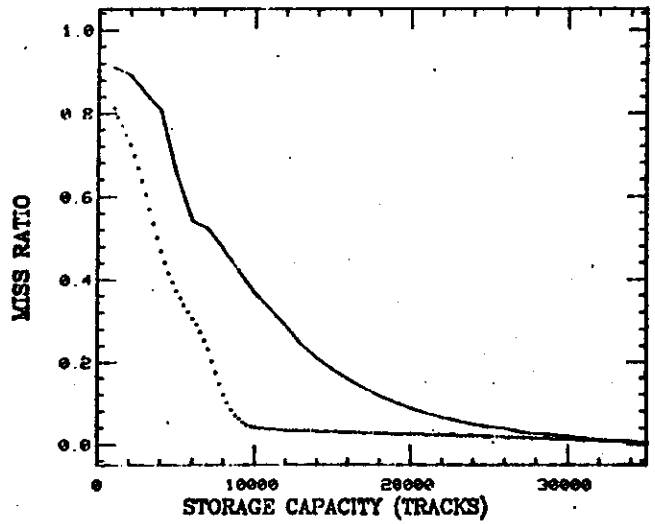

FIG. 5.2A

LEAST RECENTLY USED

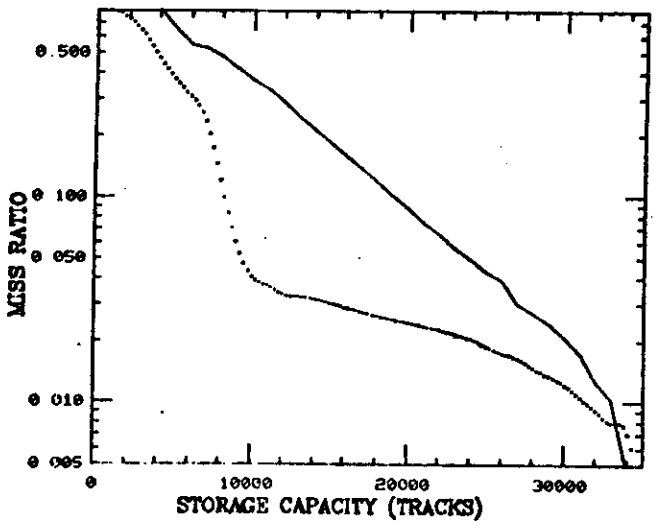

FIG. 5.2B

LEAST RECENTLY USED

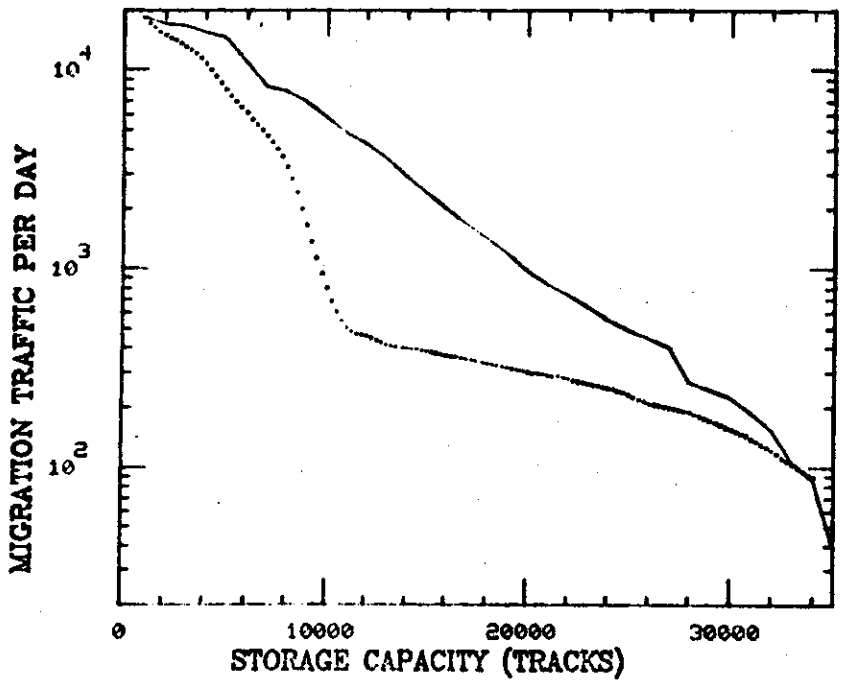

FIG. 5.3

rate curve. An example of this type of curve is in Figure 5.3. The horizontal axis is the same as for miss ratio curves; capacity of secondary storage. The vertical axis measures migration traffic, in average number of tracks migrated per day, for the corresponding secondary storage capacity.

As mentioned above, for the paging environment, the miss ratio curve and the traffic curve are identical to within a scaling factor. In the file migration environment, where the migrating elements are not all equal in size, this equivalence no longer holds. The traffic rate curve, though never reported before, is a natural extension of the miss ratio curve and can be generated by the stack processing algorithm with a simple change.

In determining the miss ratio, the program essentially records for each file access the fact that, for each secondary storage capacity less than the current depth of the file, a miss has occurred. The traffic rate modification records for each file access the information that, for each secondary storage capacity less than the current depth of the file, migration traffic proportional to the size of the accessed file must take place. The resulting traffic rate curve is not equivalent to the miss ratio curve and provides a second evaluation criterion for migration strategies.

5.5 A Primitive Migration Strategy

We now begin to discuss migration strategies. The first is a migration strategy that is used on some current machines. It is mentioned previously in Chapter II and Chapter V as an example. This strategy simply forces files to migrate when they have not been used for some pre-determined length of time. We will refer to that length of time as the migration threshold. Clearly, if the threshold is increased, fewer files migrate and the chance of accessing a file not in secondary storage (i.e. the miss ratio) decreases and less migration traffic results. On the other hand, if the threshold is small less secondary storage capacity is required. Figures 5.4a, 5.4b and 5.4c

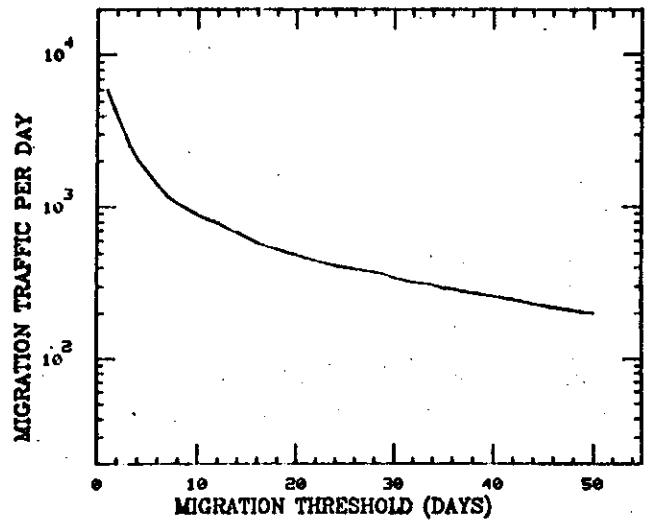

FIG. 5.4A

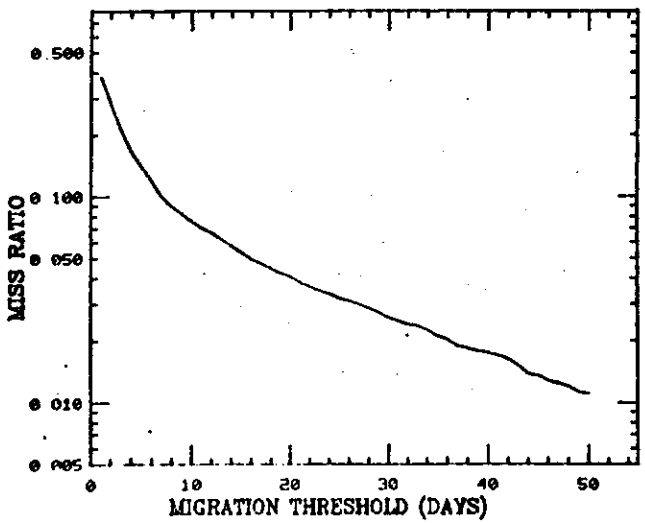

FIG. 5.4B

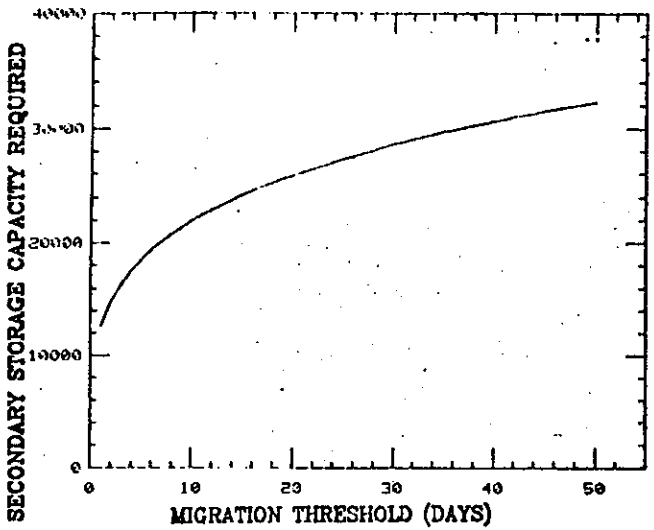

FIG. 5.4C

MIGRATION THRESHOLD REPLACEMENT

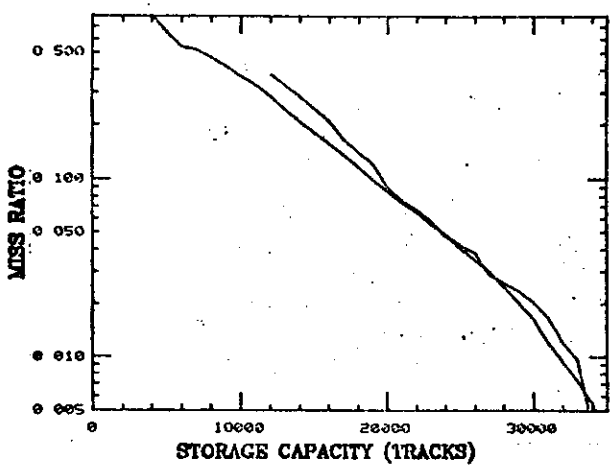

FIG. 5.4D

show these relationships. A trade-off must be made between secondary storage capacity and migration traffic. The trade-off decision depends on the actual costs involved. Chapter VI gives several examples with representative costs. The miss ratio curve for this strategy can be generated, without using the stack processing algorithm, by combining Figures 5.4b and 5.4c and removing the common parameter, "migration threshold". Figure 5.4d gives the result.

The choice of migration threshold determines, as shown in Figure 5.4b, the amount of secondary storage required. Using a fixed migration threshold, however, causes inefficient use of the secondary storage. This is because day to day variations in usage will cause the total space occupied by files whose idle time is less than the threshod to vary. Thus, secondary storage will not always be kept full. This inefficiency can be avoided by varying the migration threshold so that secondary storage is always kept full. The migration strategy then becomes:

- files migrate in order of their current idle times, largest idle time first

- a migration need take place only when secondary storage cannot hold all files not yet migrated.

This is precisely the least recently used (LRU) replacement strategy. The next section develops the LRU strategy and generates its miss ratio by stack processing.

5.6 The LRU and MIN Replacement Strategies

Figure 5.2 shows the miss ratio and traffic curves for LRU (least recently used) replacement. The solid line is the performance of LRU, the dotted line is for the MIN strategy. The MIN curve is shown in all plots, both miss ratio and migration traffic, and serves as a basis for comparison. The MIN replacement strategy, also called OPT, was first described by Belady [Be66]. Belady shows that for replacing fixed size elements (pages) no strategy can do better than the one (MIN) which first replaces those elements which will be accessed furthest in the

future. In the context of stack processing described previously in this chapter, MIN sorts the elements on the length of time to next access. This strategy minimizes migration activity by assuring that those files which will be accessed in the near future are available in secondary storage.

Because it requires knowledge of the future, in the form of the next access time for each file, MIN cannot be implemented on a running system. It is used in paging studies as a reference for practical replacement strategies. MIN is used here for the same purpose.

The MIN strategy is, however, not the optimal strategy in the file migration environment. The proof of optimality of MIN requires that the elements all be the same size. Files are unequal in size and so MIN is not optimal for files. A simple counter-example is given in Appendix B. The optimal replacement algorithm for variable sized elements is not known. A study by Casey and Osman [Ca74] discusses this problem in another context. (Other work, such as Fabry and Prieve's VMIN [Fa76], deals with fixed size pages in a variable sized store, a different problem.) The MIN strategy performs quite well however and is used here for reference. In some instances, other strategies do slightly better than MIN for some storage capacities.

5.7 Alternatives to LRU

The performance of the LRU replacement strategy was shown in Figure 5.2. We shall see that this is a reasonable strategy though there are strategies which perform better. In page replacement situations, LRU is usually the best choice. This is because of a property of program performance called "locality". Locality refers to the fact that at any given time in the execution of a typical program, there is a set of pages that are being frequently accessed. This set of pages is the current locality. The pages in the locality can be identified as those that have been used in the recent past. They are the most likely pages to be accessed in the near future, though the locality changes from time to time and some pages become less frequently used and others form the

new locality and become active.

With respect to the accessing pattern of an individual page, the locality property means that there is serial dependence between successive inter-access times. Short inter-access times indicate the page's presence in the current locality and indicate that short inter-access times are expected in the near future. This is precisely why LRU is effective in the paging environment, the most recent idle time (the LRU measure) is a good predictor of the next idle time. Chapter III discussed tests for serial dependence of inter-access intervals of files. For the typical file, there is no serial dependence of inter-access intervals. Files are generally accessed at the same rate throughout their lifetime. The accessing rate, however, does vary from file to file and LRU works in the file migration environment because the latest idle time is an estimator of the mean idle time, or equivalently, of the accessing rate. This section investigates migration strategies based on other estimators of the mean idle time; a running average of the file's idle times and the true average of the file's previous idle times.

For the running average used here, the exponentially smoothed running average, the idle interval is updated at each access, using the new idle interval sample as follows:

$$RA = k*RA' + (1-k)*X$$

where RA is the new running average RA' is the old running average X is the most recent idle interval and k is the averaging factor ($0 \leq k \leq 1$).

The exponentially smoothed running average is an unbiased estimator for the mean idle interval ($E(RA) = E(X)$). Also, in the limit with infinitely many samples, the variance of this running average is less than the variance of a single sample

$$Variance(RA) = (1-k)/(1+k)*Variance(X) < Variance(X)$$

so this running average is a better estimator.

In practical situations, there are not infinitely many samples to average and this greatly effects the power of this running average estimator. If only one sample idle interval is available for a file (or if k=0), then the running average and LRU estimators are identical. As the number of samples increases, the running average improves as an estimator, as plotted in Figure 5.5. The variance of the running average estimator is shown plotted against a number of samples for various values of k. The optimal choice of k, the averaging factor, depends on the number of sample intervals expected. Unfortunately, as Figure 5.6 shows, there are usually very few sample idle intervals for each file. Both the axes in Figure 5.6 have logarithmic scales. This relationship is often found in real life and is known as Zipf's law[Zi62]. The vast majority of files have only one or two accesses to them in their lifetimes. This means that the running average idle time cannot be expected to be much better than the previous idle time as an estimate of the mean idle time. Figure 5.7 shows the performance of the migration strategy that uses the running average idle time as the measure for sorting files in migration order. It is nearly indistinguishable from the performance of LRU replacement. For reference, in Figure 5.7 and following figures, the performance of the LRU strategy is shown as a dotted curve.

The running average has the advantage over the true average of requiring only one extra piece of information to be stored with each file. The true average, while requiring two pieces of information, the current average and current number of samples, is a slightly better estimator. The power of the true average estimator is shown in Figure 5.5 as a dotted curve. Again, because of the low number of accesses to most files (Figure 5.6), we expect that the true average estimator cannot perform much better than the LRU estimator. In fact, it is slightly worse, as Figure 5.8 shows. The reason for this poorer performance is not clear. A possible explanation is that the running average and LRU strategies, by weighting the most recent intervals more

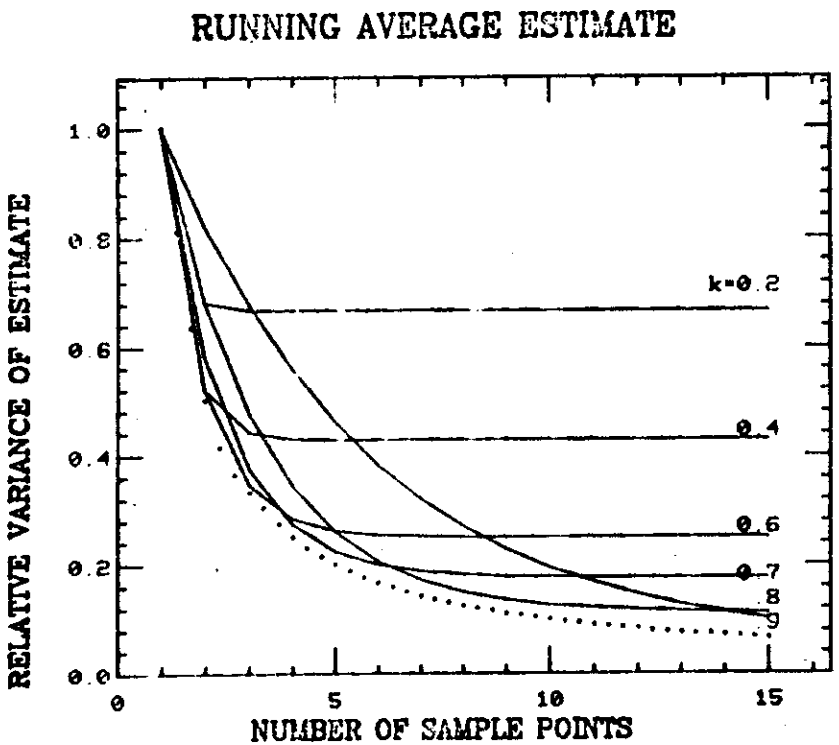

FIG. 5.5

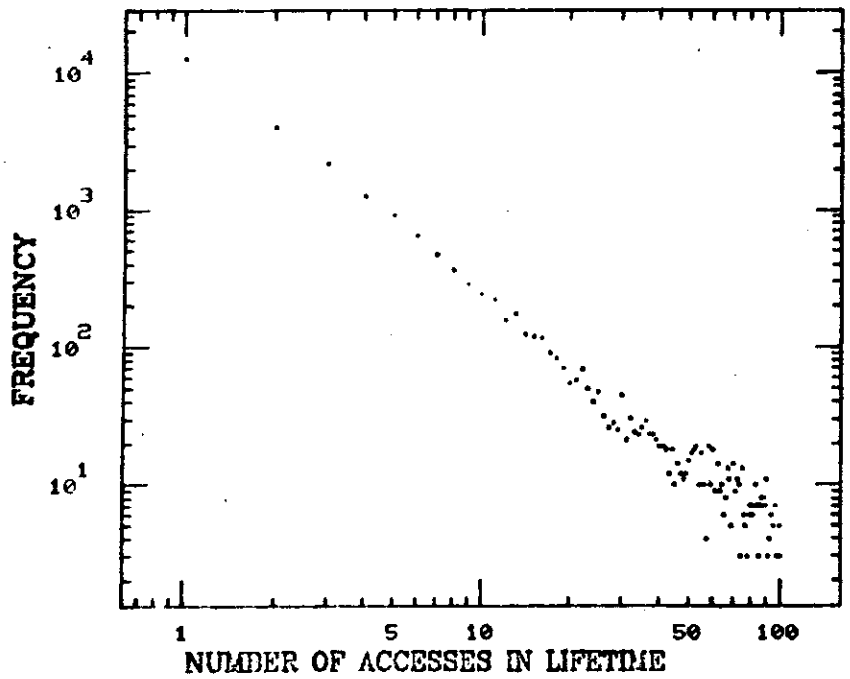

FIG. 5.6

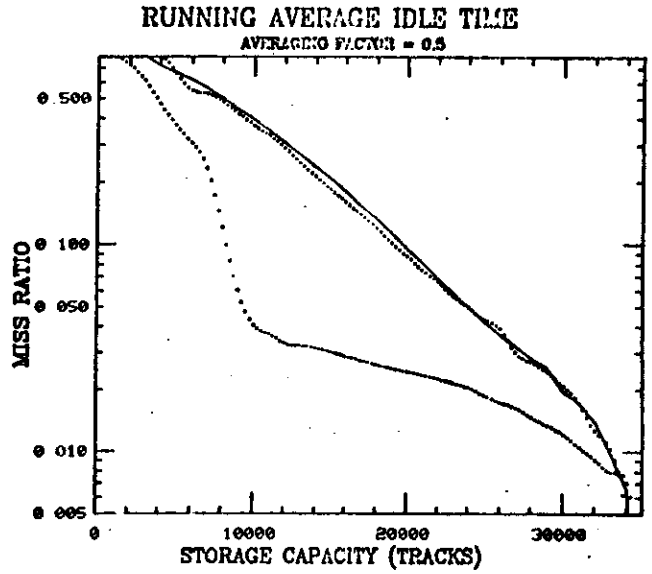

FIG. 5.7A

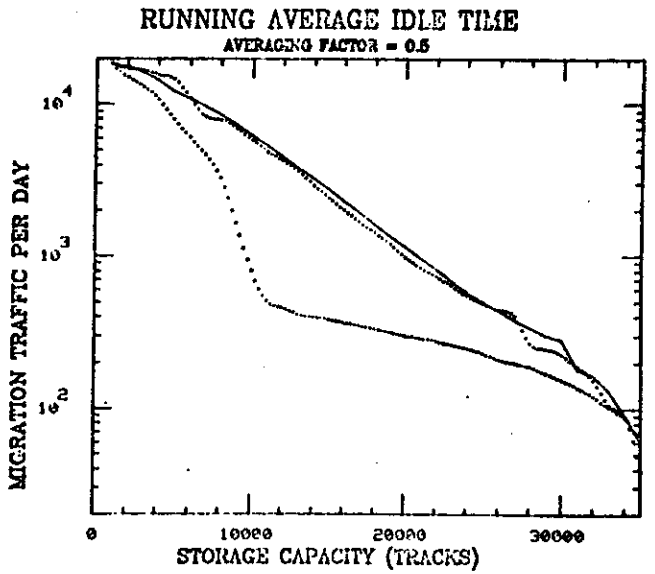

FIG. 5.7B

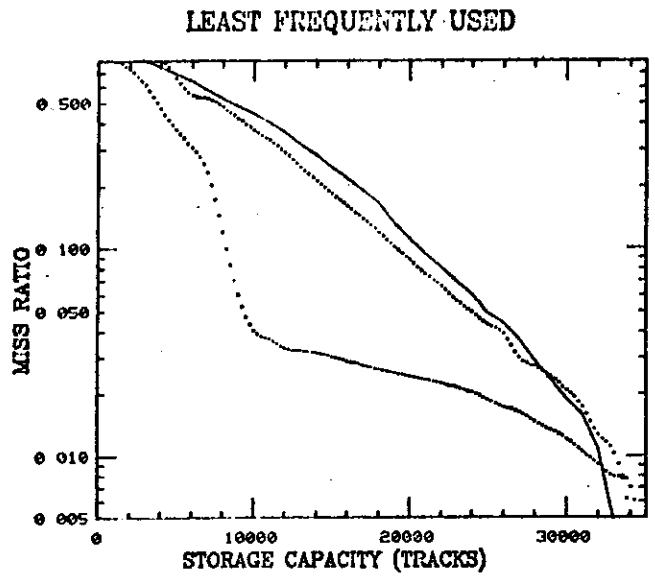

FIG. 5.8A

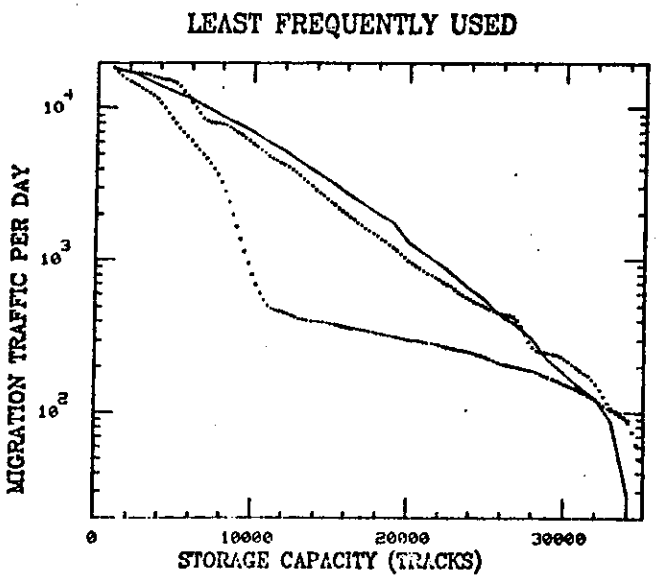

FIG. 5.8B

heavily, do a much better job of managing those atypical files which do have some serial dependence of idle intervals and for which, therefore, the most recent idle intervals are better estimators than the average. This strategy, in analogy with least recently used, is called least frequently used because the file with the highest average idle time, and thus the lowest average accessing rate (i.e., the least frequently used file), is the first to migrate. Least frequently used is abbreviated LFU.

5.8 Migration Strategies Based on File Size

The fact that files are not equal in size can be exploited in designing migration strategies. A situation where this property can be used to improve performance is visualized as follows. Consider three files, one large file and two small ones, such that the total size of the small files is equal to the size of the large file. Suppose, further, that the migration priority, as defined by the migration strategy in use, is slightly higher for the small files (they will migrate before the big file). If the secondary storage capacity is such that the small files are kept on back-up store and the large file is on secondary storage, then if all three files are accessed, one "hit" and two "misses" will result. An alternative migration strat- egy, using as priority measure the priority measure of the above strategy multiplied by the file size, would have placed the small files in secondary storage and the large file on back-up storage. The same series of file accesses would then result in two "hits" and one "miss", for a better miss ratio.

What is needed is a migration strategy whose measure makes use of the size of the file as well as some estimate of the next idle time. A natural way to combine the file size and an estimate of the files' next idle interval is to use their product. The result is an estimate of the space-time product of the file's occupancy until next access. Small files with short estimated future idle period will be the least likely to migrate because their estimated cost in secondary storage measured in space-time until next access is small. Another way to think of the product of size and next idle time is as an estimate of the accessing

"density" of the file. Small, active files are accessed more densely than others.

This type of measure has been used in related problems. The problem of dynamic file migration with unknown future accessing probabilities has not been studied before. Other research however has been published on the file allocation problem for a fixed population of files with known access probabilities (see Chapter VII for a more complete discussion). In particular Morgan [Mo74], Ramamoorthy and Chandy [Ra70], Arora and Gallo [Ar73], and Kimbleton [Ki72] all use an accessing density measure to solve particular formulations of the file allocation problem. Gilmore and Gomory [Gi66] show that a value density measure is a good heuristic for finding solutions to the knapsack problem. The knapsack problem is related to the file migration problem (fitting unequal sized elements with unequal values into a fixed space so as to maximize the total value) though the element values do not change over time. The migration strategies considered in this section are extensions of the least recently used and least frequently used strategies developed in the previous section. In each case, the new strategy sorts files on a new measure which is equal to the old measure (most recent idle time or average previous idle time) times the file size. Figures 5.9 and 5.10 show the resulting performances. As expected, the miss ratio curves are better. The migration traffic curves, however, are not as good because, although the miss ratio's are lower, the new strategies cause larger files to migrate so that those misses that do occur cause more migration traffic. Thus the selection of migration strategy between the strategies in the previous section and those same strategies extended to take file sizes into account, depends on which evaluation criterion, miss ratio or migration traffic, is considered more important. The choice of evaluation criteria is discussed in Chapter V, Section 3, and in Chapter VI.

The migration strategy which sorts files by size alone is shown in Figure 5.11a. This strategy migrates large files in preference to small files without regard for past accessing history. As a result, its performance is poor. In fact, this strategy works in direct opposition

LEAST RECENTLY USED TIMES SIZE

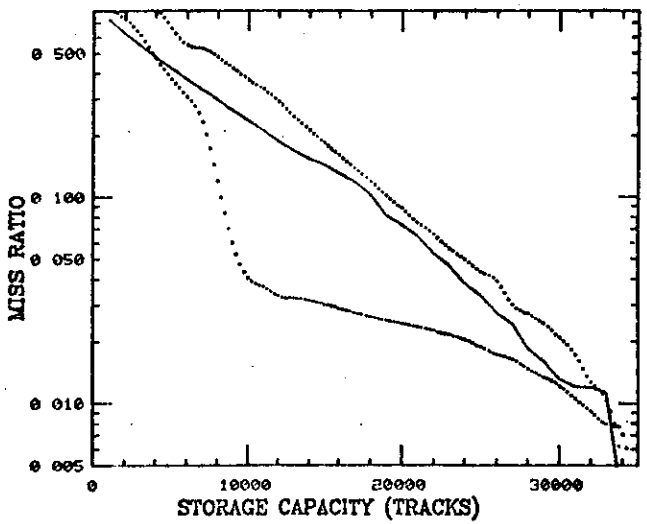

FIG. 5.9A

LEAST RECENTLY USED TIMES SIZE

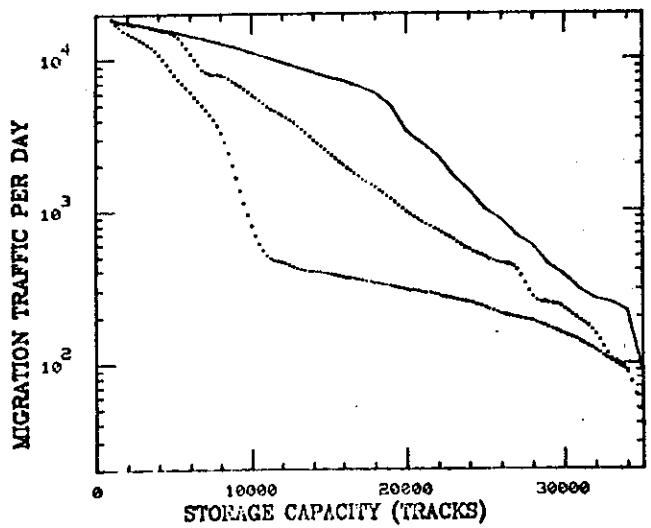

FIG. 5.9B

LEAST DENSELY USED

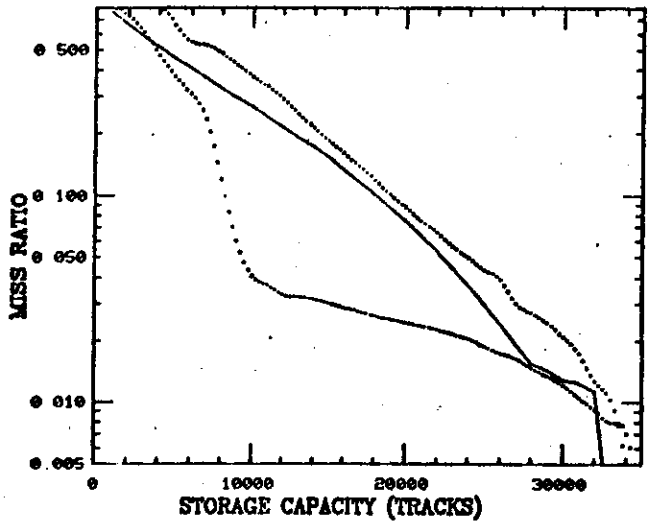

FIG. 5.10A

LEAST DENSELY USED

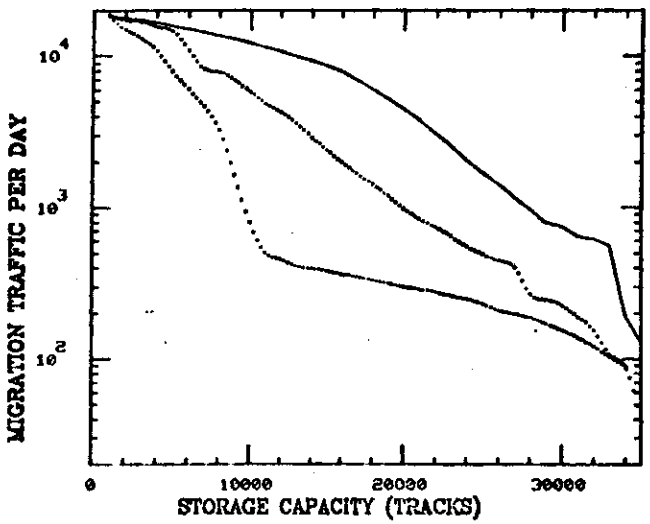

FIG. 5.10B

REPLACE LARGEST FILES FIRST

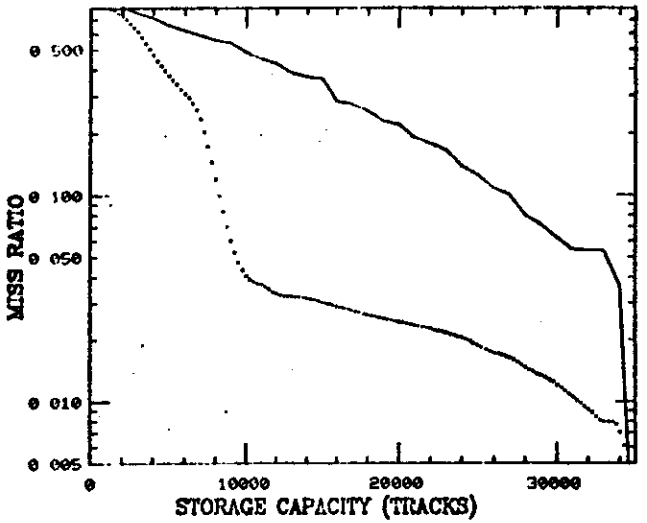

FIG. 5.11A

REPLACE SMALLEST FILES FIRST

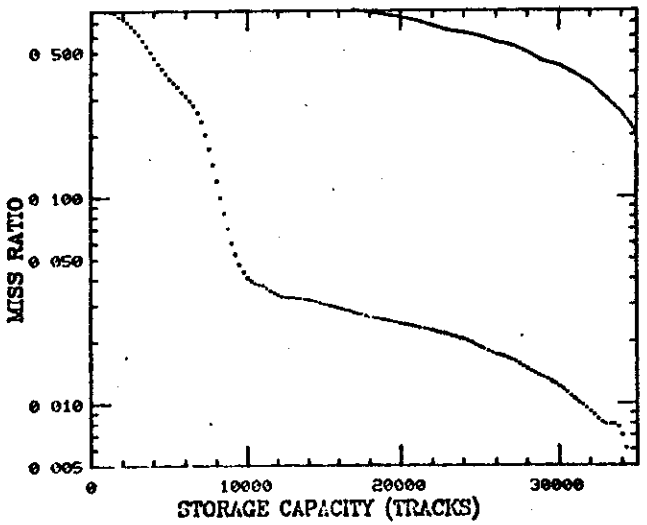

FIG. 5.11B

to the fact, shown in Chapter III, that large files are accessed more frequently than small files. This would indicate that a migration strategy should favor large files and cause small files to migrate first. The latter strategy (Figure 5.11b) also performs poorly because it again ignores the past accessing history of the files.

The extended strategy corresponding to MIN, that is the strategy that sorts files on the product of next idle interval and file size, is shown in Figure 5.12. This strategy performs substantially worse than MIN. MIN is near optimal and the strategy shown reverses many of the correct decisions made in MIN by favoring small files.

5.9 Other Migration Strategies

Figure 5.13 shows the performance of random replacement as a migration strategy. This is useful as a worst case comparison. Figure 5.14 is shown here to corroborate statements made in Chapter III. It was shown there that there is essentially no correlation between a file's lifetime or age since creation and its accessing rate. This implies that a migration strategy which orders files by age (time in the system) would perform like the random replacement strategy. This is shown to be the case in Figures 5.14.

5.10 Conclusion

This chapter on file migration has presented the major results of this work. File migration is discussed in general and its implementation is outlined. Then the specific question of the migration strategy to be used is discussed. Two measures, the miss ratio and the migration traffic rate, are suggested for evaluating migration strategies. Each strategy is simulated running on empirical trace data of real file system activity.

LRU replacement is shown to be a useful strategy. Related strategies, the running average idle interval and least frequently used, do not provide any improvement. Modifying these strategies to favor

SORT ON NEXT IDLE TIME TIMES SIZE

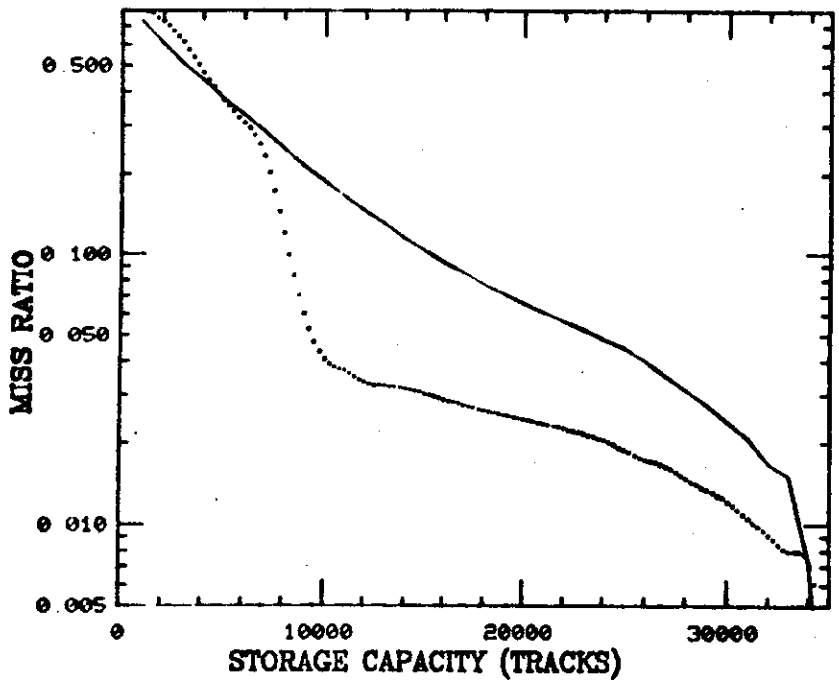

FIG. 5.12

RANDOM REPLACEMENT

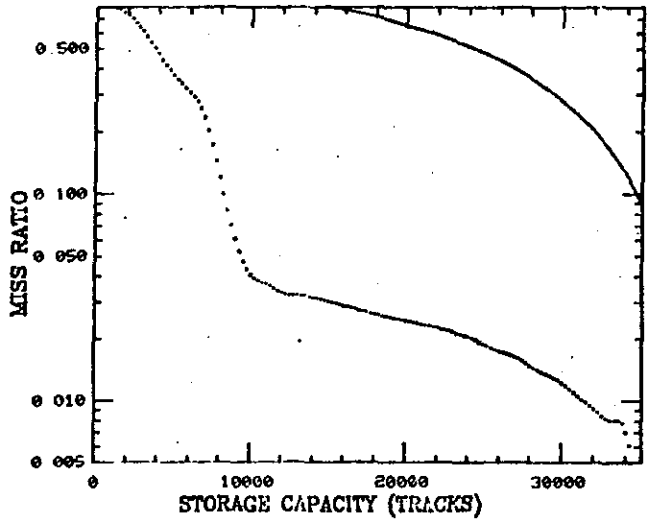

FIG. 5.13A

RANDOM REPLACEMENT

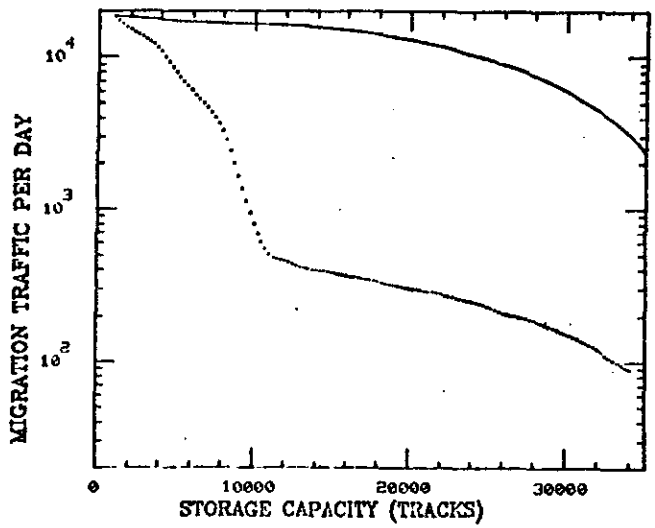

FIG. 5.13B

REPLACE YOUNGEST FILES FIRST

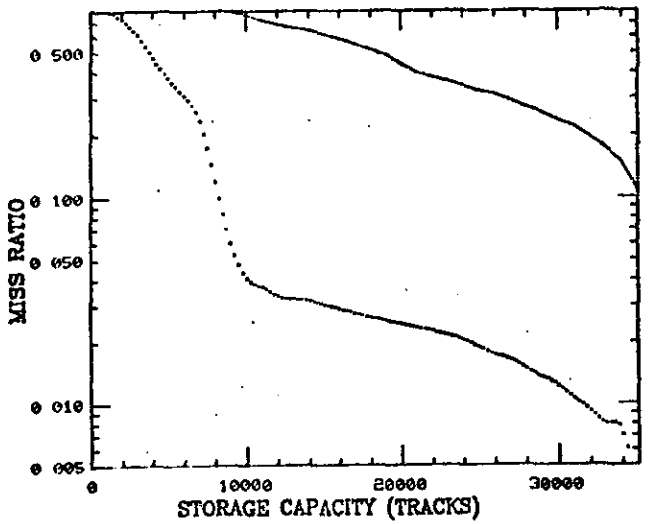

FIG. 5.14A

REPLACE OLDEST FILES FIRST

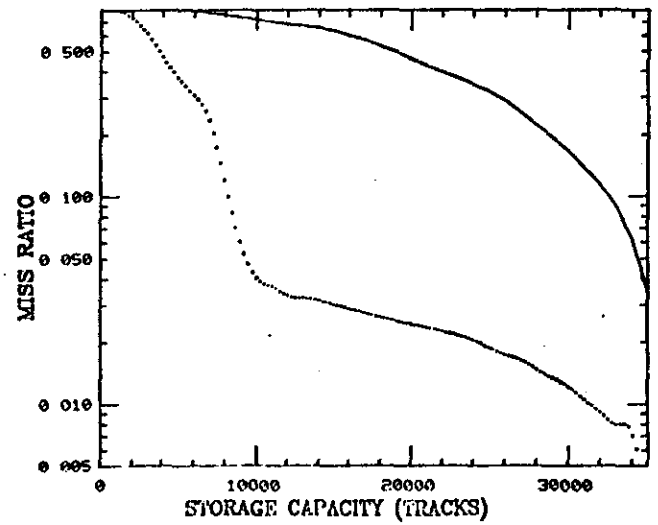

FIG. 5.14B

small files gives good improvement with respect to the miss ratio measure, but not with respect to the traffic rate measure. Chapter VI discusses examples of situations where one or the other measure is appropriate. Finally, the performance of the MIN strategy and random replacement strategy is given as near-best and near-worst cases for comparison.

CHAPTER VI

EXAMPLES

This chapter presents some examples of file migration systems. The purpose here is to put into context the results of previous chapters and to give a feeling for file migration. These examples are intended to demonstrate the cost and performance improvements that are possible, and to put into meaningful terms the two evaluation measures for migration strategies presented in Chapter V.

It is important to be careful in applying results based on empirical measurements of one computer system to radically different systems. For this reason, the hypothetical systems presented here are closely related to the one used in gathering the data on which this study is based. Other file systems with radically different scale or purpose, for example a small minicomputer file system with one disc or a very large data base system with hundreds of discs, can, of course, be analyzed using the methods of this thesis. The results of this work may not apply directly to such systems however.

6.1 Improved Performance from New Migration Strategies

Some file systems use a primitive form of LRU replacement for automatic migration of files. This strategy assigns a migration threshold and a file migrates when it has been idle for a longer period of time than the threshold value. This migration strategy has been discussed in Section 5.5. From Figures 6.1a,b (the same as Figures 5.4b,c), we can determine the miss ratio and secondary storage needs that result from a choice of migration threshold. Some migration thresholds and corresponding miss ratios and secondary storage capacity requirements, are:

Migration Threshold	Miss Ratio	Storage Required
7 days	.10	20000 tracks
14 days	.057	23750
21 days	.038	26250
30 days	.026	28599

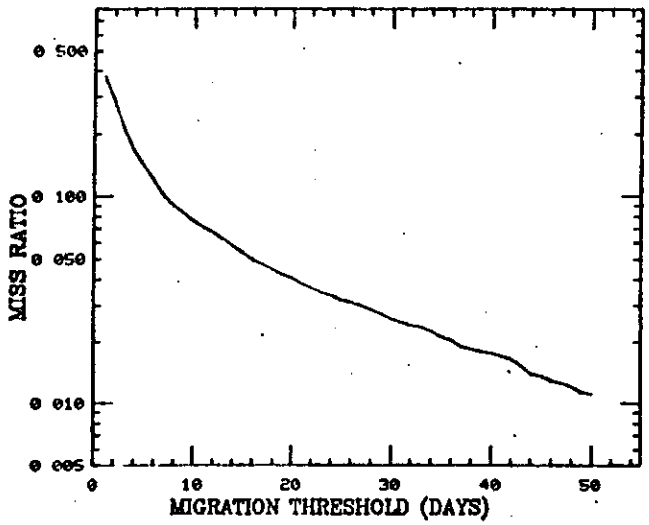

FIG. 6.1A

MIGRATION THRESHOLD REPLACEMENT

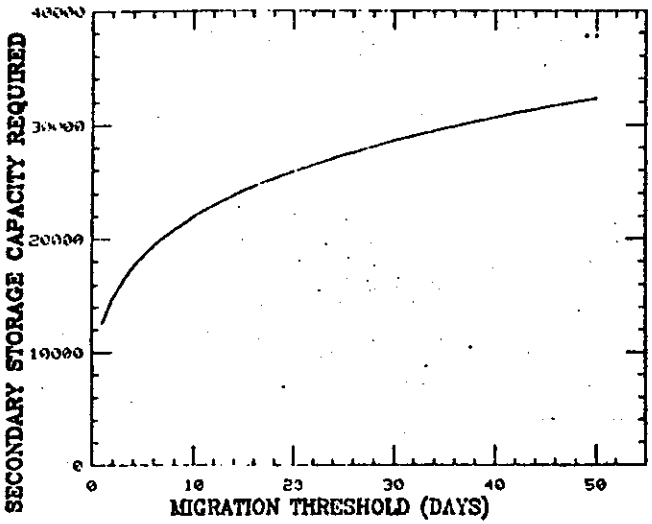

FIG. 6.1B

The discussion in Chapter V of this migration strategy pointed out that using a fixed migration threshold is wasteful of secondary storage. Because of day to day fluctuations in the accessing of files, the size of the population of files accessed within the migration threshold period (i.e., the files not currently migrated) will vary. This will cause some secondary storage to be unused some of the time. Using the LRU replacement strategy overcomes this inefficiency. LRU migrates only enough files so that secondary storage is just full. This is equivalent to using flexible migration threshold. The miss ratio for LRU is given in Figure 5.2b (reproduced here as Figure 6.2). From this figure, we can derive miss ratios for various secondary storage capacities. The LRU miss ratios and miss ratio improvements over those for the migration threshold strategy are:

Storage LRU Miss Ratio

Capacity	Miss Ratio	Improvement
20000 tracks	.085	.015
23750	.049	.008
26250	.036	.002
28500	.024	.002

The performance increase shown in these tables is achieved at no extra cost and with no change in storage hierarchy configuration. The only change required is to the migration strategy software.

It was shown in Section 5.8 that a simple modification of the LRU migration strategy, to take into account the sizes of files, provides better migration performance with respect to the miss ratio evaluation criterion. The modified strategy orders files for migration on the product of the current idle time and the file size, thus favoring (in the sense of not migrating) small files as well as recently accessed files. The miss ratio curves for LRU and modified LRU (referred to as "LRU times size") are shown in Figure 6.3. A migration system using the LRU strategy can get a substantial performance improvement by changing to the modified LRU strategy. The following table lists, for various secondary storage capacities, the miss ratios achieved by LRU and modified LRU.

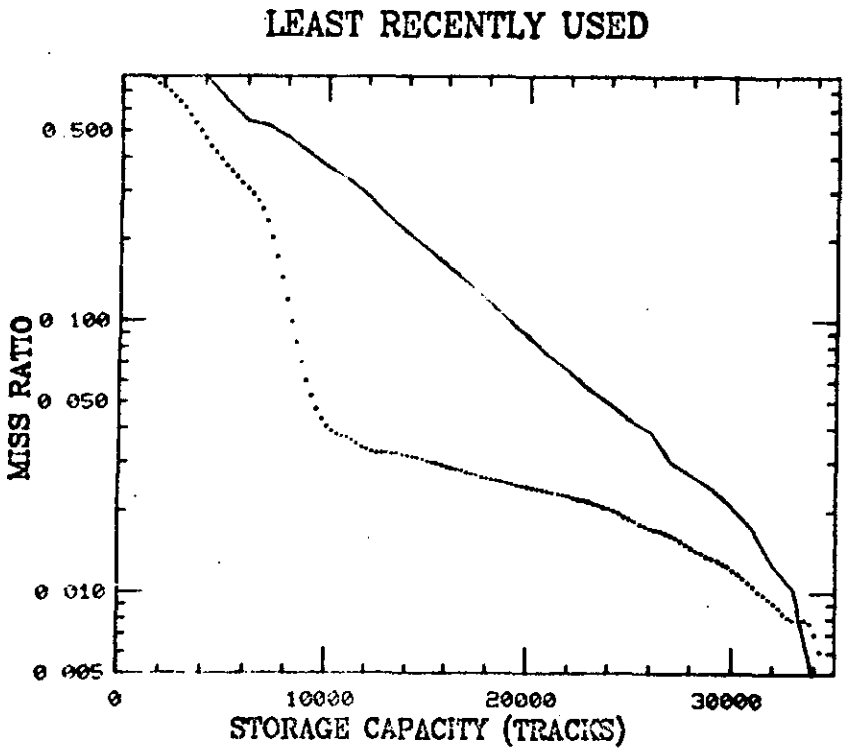
LEAST RECENTLY USED
MISS RATIO
0 500
0 100
0 050
0 010
0 005
0
10000
20000
30000
STORAGE CAPACITY (TRACKS)

FIG. 6.2

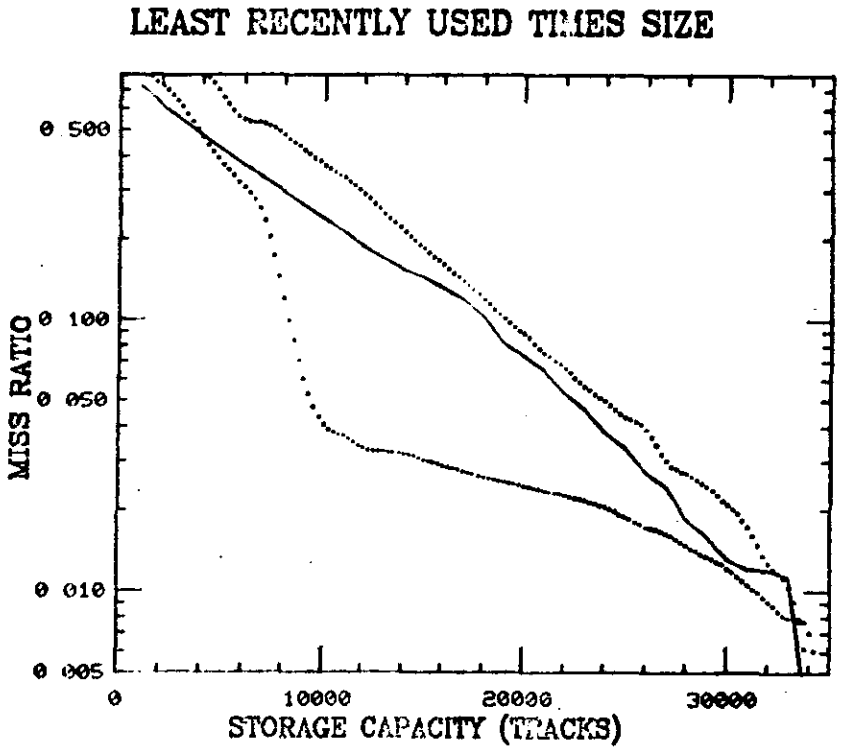
LEAST RECENTLY USED TIMES SIZE
MISS RATIO
0 500
0 100
0 050
0 010
0 005
0
10000
20000
30000
STORAGE CAPACITY (TRACKS)

FIG. 6.3

Secondary Modified Miss

Storage Capacity	LRU Miss Ratio	LRU Miss Ratio	Ratio Improvement
10000 tracks	.367	.234	.133
20000	.087	.074	.013
30000	.021	.013	.008

The effect of various miss ratio values on system performance is discussed in the next section. It is clear though that by changing migration strategies, again at no cost in increased hardware, non-negligible performance improvement can be made.

6.2 Changes in Storage Hierarchy for Improved Performance or Costs

Unless there is a surplus of secondary storage capacity, any reduction in the secondary storage capacity will cause an increase in the miss ratio. This is because when less secondary storage is available, more files must be kept on back-up storage and there is a higher probability that a file being accessed is found only on back-up storage. The effect of increasing the miss ratio will depend on the type of back-up storage being used and on the cost of delaying running programs while they wait for file migration. The cost of delaying a running program is system dependent and very hard to quantify. No attempt will be made to do so here. The effect of the type of back-up storage and, in particular, the access time to back-up storage is discussed later in this section.

If an increased miss ratio can be tolerated in a particular system, then secondary storage capacity can be reduced and hardware costs decreased accordingly. The secondary storage requirements for various miss ratio values can be determined from Figure 6.3. The following table summarizes the relationship.

Secondary Storage Capacity Required

Miss Ratio	Modified LRU	LRU
.01	33000 tracks	33000
.025	26500	28500
.05	22500	24000
.1	18000	19000
.25	9000	12500

As an example, a system that requires 99% of all file accesses to be to secondary storage (miss ratio = .01) needs 33000 tracks, but if a miss ratio of .1 (one-tenth of all accesses cause migration from back-up store) is tolerable, then only 18000 tracks are required. The savings in cost of secondary storage is 45%. Such a reduction in secondary storage capacity, besides increasing the miss ratio, also increases the amount of migration traffic. It is important to assure that the data path between secondary and back-up stores can handle the increased load. Figure 6.4 shows the migration traffic curves for LRU and modified LRU. In the example above, reducing secondary storage capacity from 33000 to 18000 tracks increases the migration traffic from 250 to 6000 tracks per day. If this traffic occured mostly in the 6 hour peak work period then the rate would be 1000 tracks per hour or one track every 3.6 seconds. This amount of traffic would be insignificant for most data channels.

The most obvious effect of increased miss ratio in a file system is the resultant increase in average access time. When all files are kept in secondary storage and the miss ratio is zero, the average access time to a file is just the average access time of the secondary storage device. If the miss ratio is not zero, then some file accesses are handled by the back-up storage. Since the access times to typical back-up storage devices are much longer than for secondary storage, an increase in miss ratio can have a large effect on the average access time to a file. The relationship is

$$T = (1-MR)*Ts + MR*(Tb + Tt)$$

where

T = effective average access time
Ts = average access time to secondary storage
Tb = average access time to back-up storage
Tt = average time to transfer a file from backup to secondary storage
MR = miss ratio.

The hypothetical system to be examined here uses disc for secondary storage and magnetic tape for back-up storage. The following values will be used in these examples:

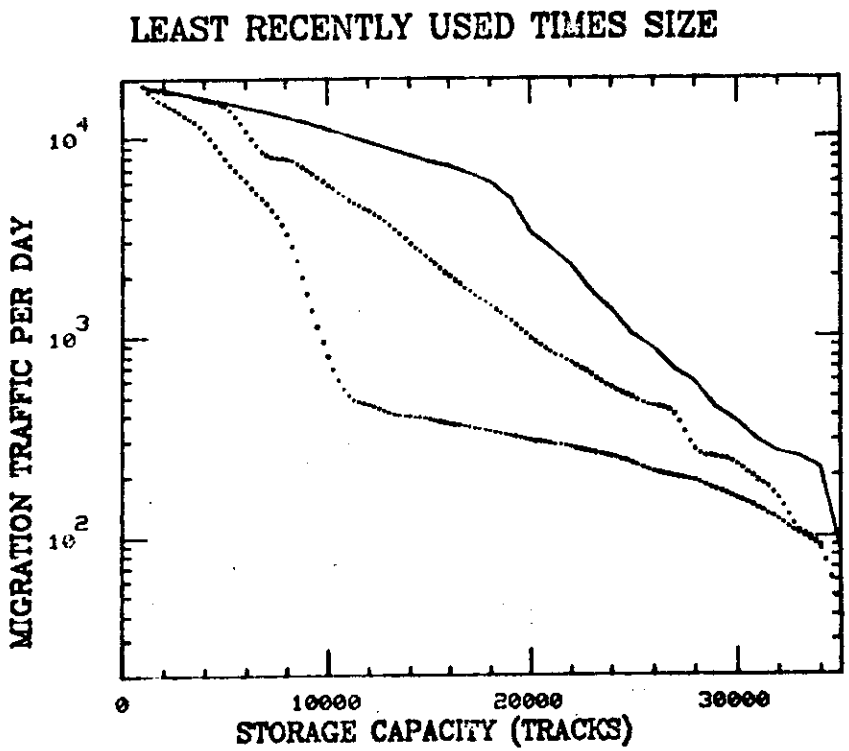

FIG. 6.4

Secondary store average access (typical disc storage device)	50 msec
Back-up storage average access (typical tape access time including mount by human operator)	200 seconds
Transfer time to secondary storage (average 20 track file at 800,000 characters/second)	200 msec

The effective average access time varies linearly with the miss ratio from 50 msec, for miss ratio of zero, to 200 seconds for miss ratio of one.

Effective average access times for various miss ratio values are:

Miss ratio	0	.0001	.001	.01	.1
Average access time	50msec	70msec	250msec	2.05sec	20sec
Increase	-	1.4 times	5	41	401

Clearly, the much slower access to back-up storage has a strong effect on the average access time. The distribution of access times, however, is strongly bimodal. Most accesses take only the 50 milliseconds required to access the disc, while a small percentage take much longer (> 200 seconds). Because of this, the average value for access time can be somewhat misleading and so must be considered with the corresponding miss ratio kept in mind.

Access to back-up storage in magnetic tape form is slowed primarily by the time it takes a human operator to find and mount the tape. Several new systems are available [Jo75], [Ha75], however, which eliminate the need for human intervention. These devices, called mass storage systems, store files on magnetic tape strips which can be physically accessed by the machine and automatically brought to and loaded on reading stations. Average access time is on the order of 15 seconds (compared to three minutes for tape). Pricing for these systems is such that, once the minimum storage capacity is purchased, costs are

approximately the same as for magnetic tape of equivalent capacity. The minimum capacity available is large (16 billion bytes; or 2 million tracks) so for file systems like the one under study, purchase of a magnetic strip file may not be justified for file migration alone. The data in this study, however, do not include files currently saved on tape. Replacing this population of tapes with a mass storage system could well be economical and have the added advantage of providing fast back-up storage for migration of disc files.

The following table shows effective access times and increases for various miss ratio values assuming the use of a 15 second access time mass storage system.

Miss ratio	0	.0001	.001	.01	.1
Average access time	50msec	51msec	65msec	201ms	1.56sec
Increase	-	1.03 times	1.3	4	31

The faster mass storage system lessens the penalty caused by increasing the miss ratio. If, for example, an effective average access time of two seconds (miss ratio = .01) was considered adequate in a magnetic tape system, then secondary storage capacity of 27500 tracks is required. With a mass storage system, the same average access time is achieved with miss ratio of 0.14. Only 15000 tracks of secondary storage are required to obtain a miss ratio of 0.14, a savings of 45%. Similary, if effective access time of 250 msec is required, mass storage allows a 16% saving in secondary storage capacity.

With a mass storage device for back-up storage, the file system can run at various miss ratio levels, depending on system performance requirements. Figure 6.5 shows that the optimal migration strategy for a given range of miss ratios may not be the optimum for a different range. For storage capacities from 22000 to 33000 tracks, the "least densely used" (modified "least frequently used", see Sections 5.7 and 5.8) performs better than "LRU times size" (modified LRU). In the range of secondary storage capacities of 10000 to 18000, the opposite is true.

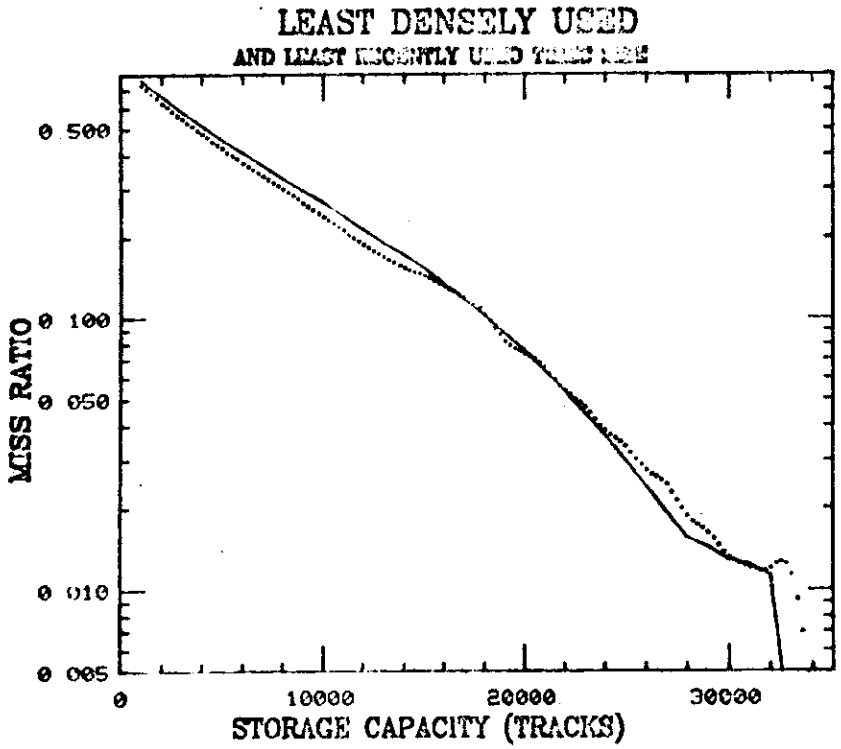

FIG. 6.5

6.3 The Migration Traffic Evaluation Measure

All the preceding examples have used the miss ratio, rather than migration traffic load, to compare performance of various configurations and strategies. This is appropriate because the miss ratio directly relates to highly visible performance characteristics, like average file access time. Furthermore, when a file system is configured to operate with a low miss ratio, there is very little migration activity and the migration traffic will not be significant. The migration traffic level for various secondary storage capacities should be kept in mind to assure that the data channel between back-up and secondary store is not overloaded. In a typical large scale computer system, the data channels have no problem handling the migration traffic of one thousand or more tracks per day (see Figure 6.6). There may exist systems, however, where the data path between secondary and back-up storage is very slow, or very busy, so that the volume of migration traffic becomes a major consideration. This situation will influence the choice of migration strategy.

A possible example is a distributed computing system with local, inexpensive processing nodes connected to a large central data base through a slow and crowded communications network. Consider a computer node of such a network with medium scale computing power and large demand for access to the central file system, and assume that the speed of the communication link, and its use by other traffic, requires that file migration traffic be limited to 5000 tracks per day. It is desired to minimize the amount of local secondary storage needed. Figure 6.6 shows the relevant plots. Previous examples used the miss ratio as an evaluation measure. Here it is appropriate to use migration traffic volume. As explained in Section 5.8, migration strategies modified to favor small size files perform poorly with respect to migration traffic, because the traffic they cause involves larger files. Figure 6.6 shows, for example, that LRU replacement performs much better than does "LRU times size".

To keep traffic below the 5000 tracks per day limit, LRU requires

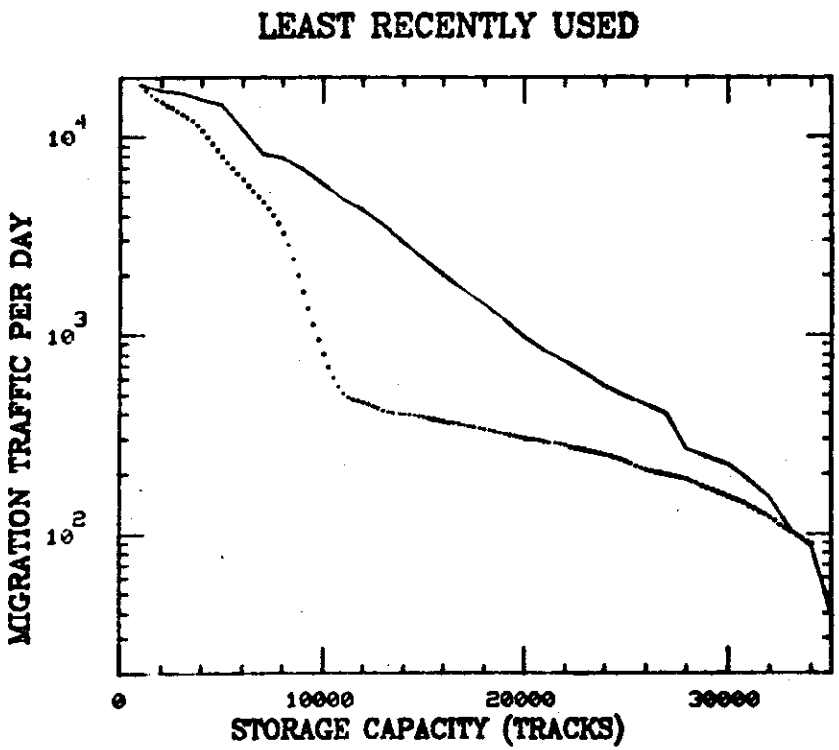

FIG. 6.6

11000 tracks of secondary storage. The "LRU times size" strategy requires 19000 tracks.

Another aspect of this example is the miss ratio. Using LRU replacement with 11000 tracks of secondary storage, the miss ratio is .32. If this is not considered adequate, changing to the "LRU times size" strategy may still not be the best choice because "LRU times size" requires 19000 tracks of secondary storage to meet the migration traffic limitation. Using LRU with more than 11000 tracks, but still less than the 19000 tracks required by "LRU times size" miss ratio as low as .11 can be achieved (Figure 6.3).

6.4 Summary

The examples in the preceding sections are intended to illustrate some of the decisions involved in file migration system design. The results of a study such as the one described in this thesis can be used to make those design decisions in an intelligent manner. The two major characteristics of file systems that have been discussed are the choice of migration strategy and the structure of the storage hierarchy.

The miss ratio and migration traffic plots give a good indication of which migration strategies should be considered. Typically, the migration traffic load will be small enough, compared to the data path capacity, that only the miss ratio need be studied closely. The modification to standard replacement strategies introduced here, in which small files are less likely to migrate than large files, often significantly improves the miss ratio performance of a migration strategy.

The file storage hierarchy has not been covered as thoroughly here, for several reasons. The primary emphasis of this work is on migration strategies. There has been a large amount of research on memory hierarchy structure and sizing (this related work is reviewed in Chapter VII), but little work on replacement strategies. Another reason to emphasize migration strategies instead of the storage hierarchy is that

changes in strategy involve software and not hardware changes and, therefore, usually are less costly and involved. If changes can be made in the storage system, however, performance may be greatly improved. The mass storage systems used in examples in this chapter eliminate the need for human intervention in loading volumes of back-up storage. The resulting improvement in access time can have a large effect on file system performance. Future technological advances that further reduce the need for mechanical motion of the recording medium will help still more. The results and methods of this thesis allow a detailed performance evaluation of any proposed storage system modifications before the changes are actually made.

CHAPTER VII

DISCUSSION

7.1 Summary and Conclusion

It has been the goal of this work to develop useful migration strategies based on empirically observed characteristics of file system activity. The initial chapters of this thesis discuss file migration concepts and the analysis performed to extract the relevant properties of file system activity from the trace data. Chapter V applies this understanding of file systems to the development and evaluation of migration strategies.

It is hoped that the following contributions have been made. File system activity data of the type presented here has rarely been gathered. This is the first time that a detailed analysis of such file system data has been published. The file allocation problem has been discussed and solved in other research only in the restricted cases of fixed sized files, a priori accessing probabilities and static (non-changing) allocation. In a real computer environment, none of these assumptions hold. This work presents and evaluates storage allocation heuristics (migration strategies) for the dynamic allocation of files with unequal sizes and unknown future accessing probabilities.

The performance of dynamic storage allocation algorithms has traditionally been measured (in paging systems) by the miss ratio. A new non-equivalent evaluation criterion, the migration traffic, is used in this thesis. Also, a new class of replacement algorithms based on file size (or, more correctly, on the predicted accessing density of a file) are shown to provide better file system performance in many cases.

Finally, the problem of file migration in real systems, its implementation, its user interface, its migration strategies, and its performance and cost improvements are discussed in detail here, for the first time.

7.2 Previous Work

Much previous work has been done in areas related to file migration, though very little is available on file migration itself. Some of this related work is reviewed briefly here. Several approaches have been taken in this previous research. Many researchers discuss optimizing the storage allocation of a fixed population of equally sized files with known probability of access. The derived allocation in these cases is always a fixed, static allocation, files not migrating as conditions change over time. Ramamoorthy and Chandy [Ra70] discuss this allocation problem. Yue and Wong [Yu73] offer proofs of the optimality of allocating blocks with the highest access probability to the highest levels of the storage hierarchy, for several different hierarchy structures.

This approach is extended by other researchers to cover unequally sized files. Arora and Gallo [Ar73] discuss this problem, as does Morgan [Mo74]. Both of these papers use the accessing probability "density" (i.e., access probability divided by file size) for ordering files for allocation. This is the basis for the modification to migration strategies that takes file size into account, introduced here in Chapter V. The Arora and Morgan papers assume known, fixed access probabilities, however. Lum et. al [Lu74] deal with a similar problem in which a file is considered to be in one of two states, active or inactive, with known probabilities. The optimal hierarchy level for allocation of the file in each state is derived given storage and transfer costs. Their solution is thus a limited form of dynamic allocation.

The work on static allocation is extended by Buzen [Bu71] and Chen [Ch73a], [Ch73b] who deal with the same type of problem, using queueing theory to study the effect of storage contention on optimal storage allocation. A different approach to storage allocation is taken in the work of Bovet and Estrin [Bo70a], [Bo70b]. They model the interactions between various parts of a running program with a graph structure. Decisions as to which parts should be kept in executable memory can be

made using this a priori program structure to predict future accessing patterns. No work has been done on the interactions between files.

Another field of research deals with proper sizing of various levels of the storage hierarchy, usually assuming that the miss ratio curve is known. These studies assign costs to the different storage levels and also to file transfer times, and seek to minimize total system cost or maximize performance within a given cost limitation. Papers by Ramamoorthy and Chandy, Lum et. al., and Chen, mentioned above, discuss hierarchy sizing.

A large body of work from the paging environment deals with dynamic allocation of pages in different memory levels, but always with the restriction that the pages are all the same fixed size. Mattson et.al. [Mi70] discuss many replacement strategies and their evaluation. The stack processing method they describe is used in this thesis. Belady [Be66] developed the MIN algorithm which gives the best possible replacement performance for fixed size pages, but uses knowledge of future accessing. Kimbleton [Ki72] takes another approach to dynamic allocation of pages, similar to the approach to file allocation in Lum et al., based on (known) inter-reference times for pages.

An interesting related problem is how to allocate storage within a given device when access times to different areas of that device are different. A paper by Denning [De67] alludes to this problem. Frank [Fr69], Ramamoorthy and Blevins [Ra71], Yue and Wong [Yu73] and Grossman and Silverman [Gr73] discuss this type of allocation for disc devices. Mitra [Mi74] studies the same problem in relation to magnetic bubble memories.

Revelle [Re74], [Re75] has published some analysis of a set of file activity data similar to the data used in this thesis.

The work in this thesis relaxes most of the restrictions in the previous work on file allocation. The goal is still to find the optimal allocation across a hierarchy of storage levels. The population of

elements to be allocated (files) is variable (files are created and deleted) over time. The files are not assumed to be all the same size. And, most importantly, neither the miss ratio curve nor the accessing probabilities of individual files are assumed to be known. Furthermore, the allocation derived is dynamic, changing as accessing to various files changes.

The allocation strategies developed here are only heuristics, however. In fact, the optimal allocation strategy, even given the future accessing of files, is not known.

7.3 Future Work

There are many areas for continuing and extending this research. In the area of migration strategies, a major unsolved problem is the optimum replacement strategy when future accessing is known. In the paging environment, the MIN algorithm [Be66] has been shown to be optimal. In the file migration environment, where files are of unequal sizes, no equivalent strategy is known. Casey and Osman [Ca74] discuss the same problem from a different viewpoint.

Another aspect of file migration is the structure of the storage hierarchy. The miss ratio and migration traffic curves show migration performance for any size hierarchy levels, but only for a linear hierarchy. If the hierarchy is a tree rather than a simple linear structure, the performance and allocation strategies are not obvious. More work is needed in this area. Another situation where the storage hierarchy structure is not straightforward is the computer network where files may be distributed throughout a complicated network structure.

Storage hierarchy levels are generally thought of as being delimited by physical device boundaries. Many devices, however, have different access times to different storage locations on the same device. Thus, allocation decisions within the limits of one device, based on predicted future accessing, may help increase performance. Other work has been done on this subject [Yu73], [De67], [Fr69], [Ra71],

[Gr73] but usually only in relation to disc devices.

An assumption throughout the work presented in this thesis is that files always migrate as a whole and cannot be split and stored partially on one level, partially on another. A more flexible file structure in which files are divided into equal sized blocks that can be stored non-contiguously might be advantageous. Such a scheme would allow only the most active blocks of a file to be stored on secondary storage so that large files with rarely used sections would not cause unnecessary migration traffic. A further advantage of dividing files into fixed sized blocks is the elimination of problems of finding variable sized blocks of free space and of storage fragmentation and garbage collection. This type of file structure has not been studied from the point of view of file migration.

The model of file system activity of Chapter IV makes many areas of further study possible. For instance, the sensitivity of migration strategy performance to changes in the file environment can be studied. New devices, file structures, and storage hierarchies can be evaluated. New measurements could be made to improve the model, in particular, to improve the time resolution of the trace data to smaller than one day, and the size resolution to less than one track.

Some computer systems structure running programs as a collection of variable sized segments ("segmentation"). The results of this thesis may provide some insight into the scheduling and memory allocation problems of segmented systems.

APPENDIX A: The Negative Binomial as a Compound Poisson

We seek a closed form for the compound distribution of Poisson occurences with variable rate, t, described by the Erlang distribution. (A similar proof is given in [Re75]).

$$\text{Prob}(X=x|T=t) = t^x / x!\ e^{-t} \qquad (1)$$

where

$$\text{Prob}(T=t) = \int_0^\infty t^{a-1} * e^{-t/b} / (b^a * (a-1)!)$$

Removing the condition in (1) gives

$$\text{Prob}(X=x) = \int_0^\infty \frac{t^x e^{-t}}{x!} \quad \frac{t^{a-1} e^{-t/b}}{b^a (a-1)!}\ dt$$

Using

$$\int_0^\infty x^{a-1}\ e^{-bx}\ dx = \frac{(a-1)!}{b^a}$$

gives

$$\text{Prob}(X=x) = \frac{(b/(b+a))^{x+a}}{x!\,b^a(a-1)!} * (x+a-1)!$$

$$= \binom{x+a-1}{x} \left(\frac{b}{b+1}\right)^x \left(1-\frac{b}{b+1}\right)^a$$

which is the probability function for the negative binomial with parameters a and b/(b+1).

APPENDIX B: Non-optimality of MIN Replacement

Belady [Be66] proves the optimality of the MIN replacement algorithm for fixed size pages. Other papers [Be74], [Le74], [Ma70] discuss the computation of miss ratios for MIN. When the elements (pages or files) to be replaced are variable in size the MIN algorithm is not optimal. The optimal replacement algorithm for variable sized elements, with known future accessing, is an open problem. In the following counter-example the population of files is {A, B, C}, with (relative) sizes 1, 2, and 2 respectively, and the accessing sequence is A, B, C, B, A. The first table shows the contents of memory (the "stack") at the time of each access for the MIN algorithm which orders the elements in the stack by time to next access.

time	1	2	3	4	5
access	A	B	C	B	A
memory	A	B	C	B	A
contents	.	B	C	B	.
	.	A	B	.	.
	.	.	B	.	.
	.	.	A	A	.
access depth	-	-	-	4	5

The average access depth (to files already in the stack) is $(4 + 5)/2 = 4.5$.

If at time = 3, file B is placed below file A in the stack, violating the MIN algorithm, the following table results

time	1	2	3	4	5
access	A	B	C	B	A
memory	A	B	C	B	A
contents	.	B	C	B	.
	.	A	A	A	.
	.	.	B	.	.
	.	.	B	.	.
access depth	-	-	-	5	3

Here the average access depth is $(5 + 3)/2 = 4$, an improvement over the MIN algorithm. With respect to miss ratio, if the top level storage capacity is three (dotted line in the tables) then the MIN algorithm results in two misses, the modified algorithm has only one miss.

BIBLIOGRAPHY

[Ar73] Arora, S.R. and Gallo, A., "Optimal Static Loading and Sizing of Multilevel Memory Systems," Journal of the ACM, April 1973.

[Be66] Belady, L.A., "A Study of Replacement Algorithms for Virtual Storage Computers," IBM Systems Journal, p. 78, June 1966.

[Be74] Belady, L.A. and Palermo, F.P., "On-line Measurement of Paging Behavior by the Multivalued MIN Algorithm," IBM Journal of Research and Development, p. 2, January 1974.

[Bo70a] Bovet, D. and Estrin, G., "A Dynamic Memory Allocation Algorithm," IEEE Trans. on Computers, p. 403, May 1970.

[Bo70b] Bovet, D. and Estrin, G., "On Static Memory Allocation in Computer Systems," IEEE Trans. on Computers, p. 492, June 1970.

[Bu71] Buzen, J., Queuing Network Models of Multiprogramming (thesis), Harvard University, May 1971.

[Ca74] Casey, R.G. and Osman, I., "Generalized Page Replacement Algorithms in a Relational Data Base," ACM SIGMOD Workshop on Data Description, p. 101, May 1974.

[Ch73a] Chen, P., Optimal File Allocation (thesis), Harvard University, August 1973.

[Ch73b] Chen, P., "Optimal File Allocation in Multilevel Storage Systems," National Computer Conf. 1973, p. 277.

[Ch76] Chu, W. W., and Opderbeck, H., "Analysis of the PFF Replacement Algorithm via a Semi-Mackov Model," Comm. of ACM, p. 298, May 1976.

[Co62] Cox, D.R., Renewal Theory, Methuen, 1962.

[Co66] Cox, D. R. and Lewis, P.A.W., The Statistical Analysis of Series of Events, Methuen, 1966.

[De66] Denning, P.J., Resource Allocation in Multiprocess Computer Systems(thesis), MIT, MAC-TR-50 (AD675554), May 1966

[De67] Denning, P. J., "Effect of Scheduling on File Memory Operations," Spring Joint Computer Conf. 1967, p. 9.

[De68] Denning, P. J., "Thrashing: its Causes and Prevention," Fall Joint Computer Conf. 1968, p. 915.

[Fa76] Fabry, R. S. and Prieve, B.G., "VMIN-An Optimal Variable-Space Page Replacement Algorithm," Comm. of ACM, p. 295, May 1976.

[Fe50] Feller, W., An Introduction to Probability Theory and Its Applications, Wiley, 1950.

[Fr69] Frank, H., "Analysis and Optimization of Disk Storage Devices for Time Sharing Systems," Journal ACM, p. 602, October 1969.

[Gi66] Gilmore, P.C. and Gomory, R.E., "The Theory and Computation of Knapsack Functions," Operations Research, p. 1045, Dec. 1966.

[Gr73] Grossman, D.D. and Silverman, H.F., "Placement of Records on a Secondary Storage Device to Minimize Access Time," Journal of the ACM, p. 429, July 1973.

[Ha75] Harris, J. P., Rohde, R.D., and Arter, N.K., "The IBM 3850 Mass Storage System: Design Aspects," Proc. of IEEE, p. 1171, August 1975.

[Jo75] Johnson, C.T., "The IBM 3850: A Mass Storage System with Disk Characteristics," Proc. of IEEE, p. 1166, August 1975.

[Kh60] Khintchine, A., Mathematical Models in the Theory of Queueing, Griffin, 1960, pp. 50-56.

[Ki72] Kimbleton, P., "Core Complement Policies for Memory Allocation and Analysis," Fall Joint Computer Conf. 1972, p. 1163.

[Kn73] Knuth, D., The Art of Computer Programming, Volume 3: Sorting and Searching, Addison-Wesley, 1973, pp 145-150.

[Le66a] Lewis, P., "A Computer Program for the Statistical Analysis of a Series of Events," IBM Systems Journal, p. 202, October 1966.

[Le66b] Lewis, P. and Cox, D., "A Statistical Analysis of Telephone Circuit Error Data," IEEE Trans. on Communication Technology, p. 382, August 1966.

[Le71] Lewis, P. and Yue, P., "Statistical Analysis of Program Reference Patterns in a Paging Environment," IEEE Proc. Conf. on Computers, Boston, 1971.

[Le73] Lewis, P. and Shedler, G., "Empirically Derived Micromodels for Sequences of Page Exceptions," IBM Journal of Research and Development, p. 86, March 1973.

[Le74] Lewis, C.H. and Nelson, R.A., "Some One Pass Algorithms for the Generation of OPT Distance Strings," IBM RC-4758, March 1974.

[Lu74] Lum, V.Y., Senko, M.E., Wang, C.P., Ling, H., "A Cost Oriented Algorithm for Data Set Allocation in Storage Hierarchies," Internal IBM Report, 1974.

[Ma70] Mattson, R., Gescei, J., Slutz, D., Traiger, I., "Evaluation Techniques for Storage Hierarchies," IBM Systems Journal, p. 78, March 1970.

[Mi74] Mitra, D., "Some Aspects of Hierarchical Memory Systems," Journal of the ACM, p. 54, January 1974.

[Mo74] Morgan, H.L., "Optimal Space Allocation on Disk Storage Devices," Comm. of ACM, p. 139, March 1974.

[Pa62] Parzen, E., Stochastic Processes, Holden Day, 1962.

[Ra70] Ramamoorthy, C.V. and Chandy, K.M., "Optimization of Memory Hierarchies in Multiprogramming Systems," Journal of the ACM, p. 426, July 1970.

[Ra71] Ramamoorthy, C.V. and Blevins, P.R., "Arranging Frequency Dependent Data on Sequential Memories," Spring Joint Computer Conf. 1971, p. 545.

[Re74] Revelle, R., "Characteristics of a Large Scale On Line Data Base," IBM RJ-1416, July 1974.

[Re75] Revelle, R., "An Empirical Study of File Reference Patterns," IBM RJ-1557, April 1975.

[Yu73] Yue, P.C. and Wong, C.K., "On the Optimality of the Probability Ranking Scheme," Journal of ACM, p. 624, October 1973.

[Zi49] Zipf, G.K., Human Behavior and the Principal of Least Effort, Addison-Wesley, 1949.