LBL-8278 c.2
UC-32

LONG TERM FILE MIGRATION - PART II: FILE REPLACEMENT ALGORITHMS

Alan Jay Smith

October 1978

Prepared for the U. S. Department of Energy
under Contract W-7405-ENG-48

LBL-8278 c.2

LBL-8278

LONG TERM FILE MIGRATION - PART II: FILE REPLACEMENT ALGORITHMS*

Alan Jay Smith**

University of California, Berkeley
Lawrence Berkeley Laboratory
Berkeley, California 94720

C.R. Categories: 4.35, 4.33, 4.41

Keywords: File Migration, Paging, Replacement Algorithm

ABSTRACT

The steady increase in the power and complexity of modern computer systems has encouraged the implementation of automatic file migration systems which move files dynamically between mass storage devices and disk in response to user reference patterns. Using information describing thirteen months of text editor data set file references, (analyzed in detail in the first part of this paper), we develop and evaluate algorithms for the selection of files to be moved from disk to mass storage. We find that algorithms based on both the file size and the time since the file was last used work well. The best realizable algorithms tested condition on the empirical distribution of the times between file references. Acceptable results are also obtained by selecting for replacement that file whose size times time to last reference is maximal. Comparisons are made with a number of standard algorithms developed for paging, such as Working Set, VMIN, etc. Sufficient information (parameter values, fitted

equations) is provided that our algorithms may be easily implemented on other systems.

---

*Partial support for this research has been provided by the National Science Foundation under grants MCS75-06768 and MCS77-28429, and by the Department of Energy under Contracts W-7405-ENG-48 (to the Lawrence Berkeley Laboratory) and EY-76-C-03-0515 (to the Stanford Linear Accelerator Center).

**Computer Science Division, EECS Department, University of California, Berkeley, California, 94720. The author is also on the staff of the Lawrence Berkeley Laboratory and is a visitor at the Stanford Linear Accelerator Center.

I. INTRODUCTION

Most large computer installations have memory hierarchies ~~similar to that shown in figure 1. The limited storage~~ capacity of the disks generally result in some form of file migration, whereby active files are kept on or moved to the disk and inactive files are moved to or kept on tape or mass storage. This migration may be managed by the user or may be done automatically by the system. The increasing need for such migration has led many large computer installations and manufacturers to supply migration as a feature of the file system, although such features vary widely in their transparency to the user and the degree to which they are automatic. Currently SLAC (Chafee et al., 1977), the Lawrence Berkeley Laboratory (Knight, 1976) and a number of other Department of Energy Laboratories (DOE, 1977) have projects in progress to install such file migration systems. IBM now provides automatic file migration (IBM, 1978, Considine and

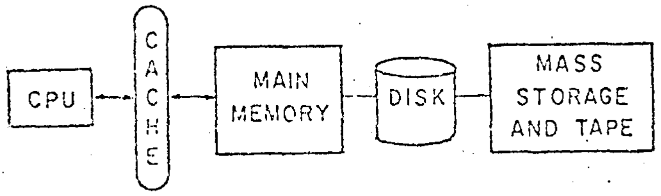


Figure 1

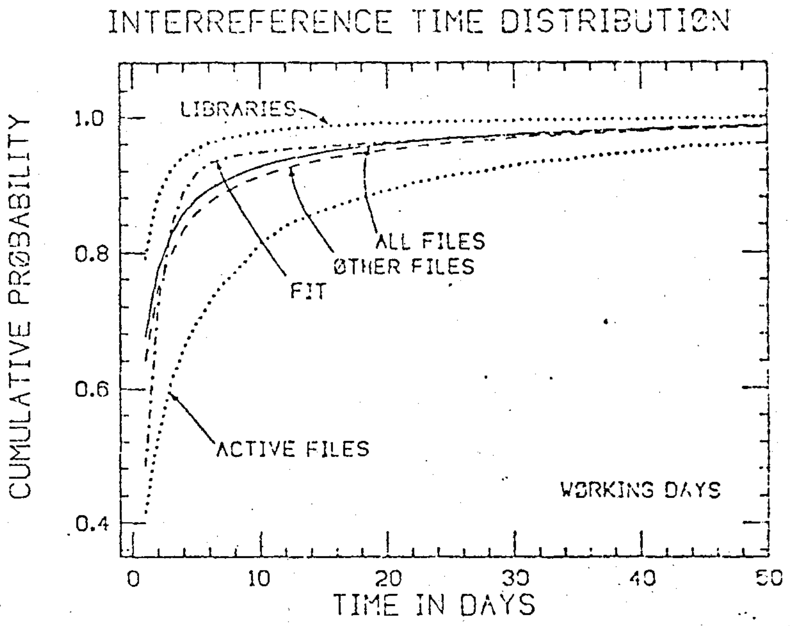


Figure 3

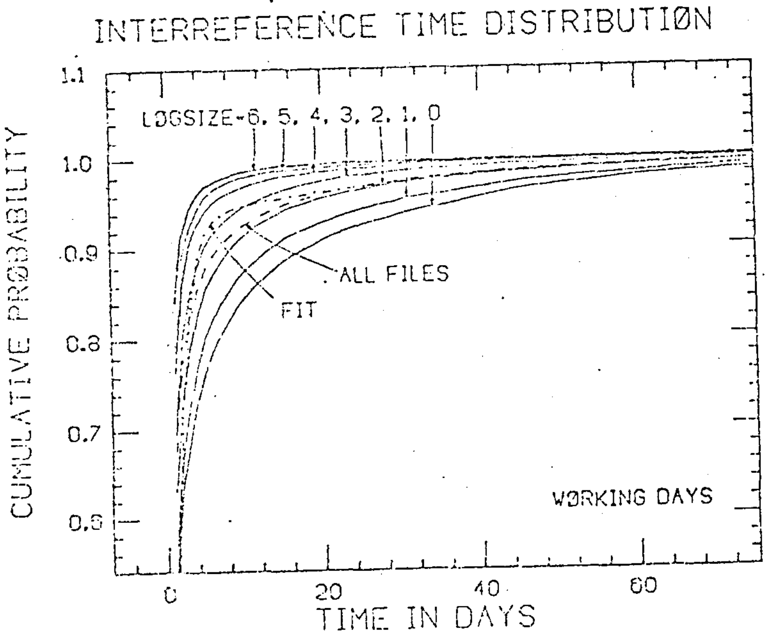


Figure 2

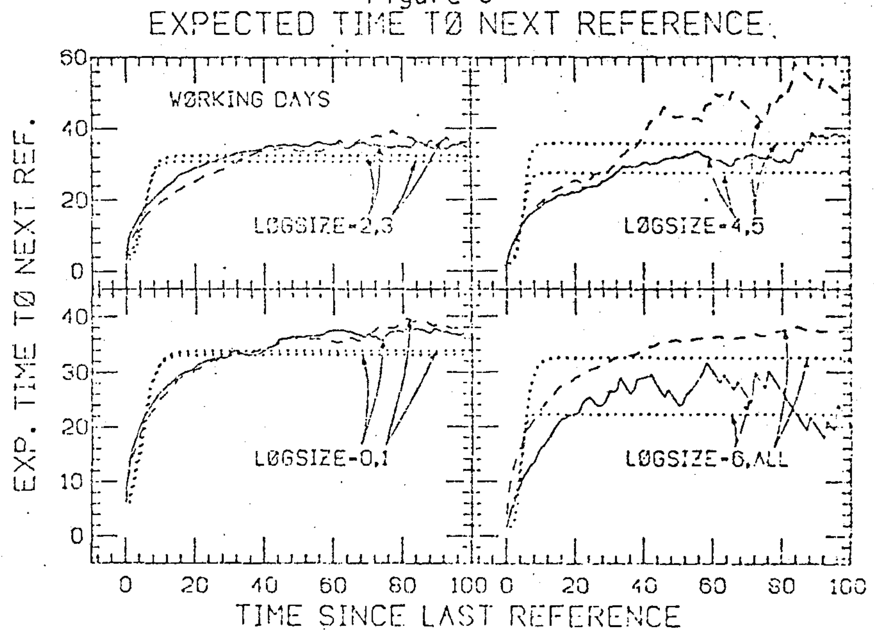


Figure 4

Myers, 1977). Various independent software vendors (e.g. UCC, 1977) also have available file migration packages.

The spreading popularity of file migration has not been accompanied by the research results needed to select good or even satisfactory algorithms for the fetch, placement and replacement of files. The problems are: when to migrate a file from mass storage to disk (fetch algorithm), where on disk to place it (placement algorithm) and when to remove that file to mass storage when the disk space is again needed (replacement algorithm). In this paper, we develop and evaluate, based on real file system access data, a number of file replacement algorithms. Our data does not permit us to consider the fetch problem in a useful way and we choose at this time not to consider the placement algorithm problem.

There has been considerable study of the replacement algorithm problem in the context of main memory paging. We refer the reader to Smith (1978a) for an extensive bibliography on the subject. With regard to files, however, the only useful study of file migration, because it uses real data, is one by Stritter (1977); we use the same data as he collected, but our analysis goes considerably further. To our knowledge, the only other set of data in existence suitable for similar research is that desribed by Revelle (1975), and he has not considered file migration algorithms. Zeheb and Boies (1978) describe the algorithm that they implement, but their system is specialized and no comparative evaluations or data are given.

A number of other researchers have considered the file migration problem, either in general, descriptive terms or by using unvalidated mathematical models. In the first category, we note papers by Boyd (1978) and Eastlake (1977). Mathematical models and mathematical optimization results are presented by Morgan (1974), Morgan and Levin (1977), Lum et al. (1975) and Segall (1976). Some of these authors provide more thorough considerations of some of the issues than we have space to do here, and the interested reader may wish to refer to some of those papers for background.

As noted earlier, this paper is concerned with the development and comparative evaluation of algorithms for the migration of files from disk back to mass storage. Our emphasis thoughout is on the use of measured file system data to both construct algorithms and then to evaluate them. In the companion paper to this (Smith, 1978b), we discuss the origin and features of this data in detail and analyze it extensively. A brief summary of this information is provided in the next two sections (II and III), ~~but the reader is urged~~ to refer to the paper referenced. We then consider principles and criteria for replacement algorithms and that discussion is followed by a section in which file replacement algorithms are explained and/or developed. Experimental comparisons and parameter values appear in section VI. Some consideration of various implementation issues appears in the last section.

II. DATA DESCRIPTION

Our data consists of a record of each Wylbur (Fajman and Borgelt, 1973) text editor data set that existed over a 384 day period (1974-5) at the Stanford Linear Accelerator Center. This data was collected by E. P. Stritter and a preliminary analysis of it appears in his dissertation (Stritter, 1977). For each file and each day, one bit is available indicating whether or not that file was referenced that day. Also available is the file name, the account id of the file's owner, the date on which the file was created, the date (if any) on which the file was scratched and the file size (in IBM 2314 disk tracks; 1 track is approximately 7000 bytes). No information is available as to whether the file was read or written, or how many times per day it was used.

We were able to classify files both by size and "class". Files were placed into seven size groups, based on the base 2 logarithm of their size in tracks. The file size groups (numbered 0...6; called "logsize" or "Lsize" or "L") were: 1 track, 2-3 tracks, 4-7 tracks, 8-15 tracks, 16-31 tracks, 32-63 tracks and >=64 tracks. There were three file classes: files with the name "Active" were created by the system to save Wylbur text editor data sets in use at the time of a system crash or an automatic logout. Files whose names began with "Lib" were classified as partitioned data sets and were placed in a class called "libraries". All other files were placed in a class called "other files". Some of our replacement algorithms use both the size and class groupings.

The 384 day period during which the system was observed consisted of working days, holidays, weekends and 15 days during which either no data was collected or the system was down. Since file migration would likely occur on working days only, we have restricted our work in this paper to using the data for working days only. Specifically, we have mapped all file events (create, scratch, reference) onto the next working day after the given holiday, weekend or unmeasured day. Thus for the purposes of migration, files created or accessed on a Saturday would be considered to have been acted upon on the following Monday. The intent is to represent a system in which the file replacement program would run every working day at midnight, using "time since last refetence" based only on working days.

III. FILE REFERENCE PATTERNS

As noted earlier, file reference patterns were extensively analyzed in part I of this paper (Smith, 1978b). Here we summarize the essential findings and present some results not shown earlier.

It was observed that most files were used on a small number of days - half of all files were accessed on two or fewer days during the measurement period. The mean number of days files were referenced, however, was 10.6; thus the distribution of day-references/file is highly skewed. We were therefore able to statistically analyze the reference process to only a fraction of all files (about 20%); the remainder had been referenced so few times that no statistical analysis was possible. Of those files tested, about 1/3 were found to show a declining rate of reference with age. Considering only those files with no trend, about 5% showed statistically significant serial correlation of the interrefernce intervals (positive in almost all cases) and about 40% were found to have interreference time distributions that were more skewed that the geometric distribution. Significant differences in various parameters (interreference time distributions, number of references, etc.) were found between files of different sizes and classes.

An attempt was made to fit the interreference distributions with a two part hypergeometric distribution.

Let g(i,L,C) be the empirical interreference time distribution for files of the given size class (L) and class (C), and let gf(i,L,C) be the fitted distribution. Then gf(i,L,C) is of the form:

$$gf(i,L,C) = a\ b\ (1-b)**(i-1) + (1-a)\ c\ (1-c)**(i-1) \qquad (1)$$

The method of moments was used to select the values of a, b and c $(0<=a<=1,\ 0<b<1,\ 0<c<1)$ for each size/class combination and the results are shown in table I. Although the chi-square goodness of fit test showed that in most cases the fit was unsatisfactory, we do make use of the fitted distribution later in this paper and find it useful.

Figures 2 and 3 show the empirical interreference time distributions for references to files and fitted distributions to all files. These figures give the results for working days (as does table I) as compared to table VIII and the figures in (Smith,1978b) which shows similar information for all days.

IV. PRINCIPLES, CRITERIA AND COMPUTATIONS FOR FILE REPLACMENT ALGORITHMS

In order to compare file replacement algorithms and select one as being better than another, it is necessary to have some criteria for evaluation. Our criteria is defined as follows: let M(A,P) be the fraction of all file "references" which require a fetch from mass store (miss ratio). (The term "reference" here is used to refer to the first time a file is used on a given day; it is assumed that files stay resident for the entire remainder of the day and thus subsequent reads, writes or opens are ignored in our computations.) S(A,P) is

Table I

Working Days

Size (tracks)	Class	Fitted Parameter Values a	b	c	Niref
1	Libraries	.924	.612	.035	189
2-3		.952	.523	.029	887
4-7		.924	.644	.047	4302
8-15		.974	.612	.039	14947
16-31		.981	.719	.051	21099
32-63		.965	.851	.126	7820
>=64		.989	.802	.048	1792
All		.977	.677	.043	51036
1	Other Files	.835	.458	.032	23745
2-3		.880	.418	.029	25500
4-7		.916	.466	.031	24893
8-15		.941	.491	.032	22824
16-31		.954	.543	.034	15223
32-63		.980	.597	.027	10576
>=64		.982	.749	.045	9814
All		.914	.490	.031	132575
1	Active	.818	.280	.026	3654
2-3		.793	.362	.032	1017
4-7		.929	.265	.020	516
8-15		.958	.379	.017	216
16-31		.875	.873	.045	139
32-63		-	-	-	56
>=64		-	-	-	3
All		.843	.288	.026	5611
1	All Files	.836	.417	.030	27598
2-3		.879	.418	.030	27404
4-7		.922	.468	.031	29711
8-15		.956	.527	.033	37987
16-31		.972	.627	.037	36461
32-63		.987	.655	.028	18452
>=64		.983	.756	.045	11609
All		.930	.518	.031	189222

Table II

Stochopt File Retention Period

File Size	Miss Ratio 1%	5%	10%
1	226	226	226
2	235	235	23
3	235	49	16
4	246	28	12
5	246	17	9
6	246	17	6
7	172	13	6
8	78	12	6
9	78	11	6
10	46	10	6
11	40	9	5
12	40	8	4
13	38	7	4
14	35	7	4
15	24	6	4
16	35	7	4
20	16	6	4
25	13	5	3
30	12	4	3
35	10	4	2
40	10	4	2
45	9	3	2
50	7	3	2
60	7	3	2
70	6	3	2
80	6	2	2
90	6	2	2

Table III

Parameter Values

Algorithm	Miss Ratio 1%	5%	10%
Working Set	57	15	7
STWS	540	225	140
Etnr(Lsize&Class)	570	200	110
Etnr(Lsize)	520	200	120
Etnrf(Lsize&Class)	620	210	105
Etnrf(Lsize)	700	210	100
STP**1.4	1800	460	260

Table IV

Tracks Transfered per Day /
Fraction of Volume of Online Files

Algorithm	Miss Ratio 1%	5%	10%
STWS	540/.020	3500/.140	5900/.252
Etnr(Lsize&Class)	600/.033	1600/.073	3500/.200
Etnrf(Lsize&Class)	540/.019	1400/.060	3400/.155
STP**1.4	300/.015	2200/.090	5000/.220
Stochopt	370/.013	1440/.061	2100/.101

the mean number of disk tracks occupied by on-line files. "A" denotes the file replacement algorithm in use and "P" the specific parameter value in use for that algorithm. The pair (S(A,P),M(A,P)) constitutes an operating point and may be plotted on an x-y plot as shown in figures 6-15.

An operating point (X0,Y0) is considered to be better than an operating point (X1,Y1) iff (a) X0<=X1 and Y0<Y1 or (b) X0<X1 and Y0<=Y1. It should be clear that it is possible for two operating points to not be ordered as better or worse (e.g. X0<X1, Y0>Y1). An algorithm A is considered to be uniformly better than an algorithm B if for all operating points (XA,YA) for A and (XB,YB) for B, XA=XB ===> YA<YB. (The better algorithm has an (S,M) curve which is to the left and below the worse algorithm). Algorithms which are not uniformly better or worse may only be compared at specific operating points or ranges of operating points.

There are two important observations to be made about our criteria for comparison. First, we have treated all file misses as equally important. In particular, a missing large file is considered to be no more costly than a missing small file. Since the largest file observed (252 tracks) can be transmitted in less than 1.5 seconds and since the mean access time for most forms of mass storage is upwards of 10 seconds (more likely several minutes), ignoring file size seems to be appropriate. (Stritter (1977) does show some results based on file size, however). Second, we have not selected a specific operating point. In contrast, a very popular criterion for comparing paging algorithms is to compare minimal space-time products (see e.g. Chu and Opderbeck, 1976). We do not believe that this criterion is appropriate even for paging algorithms, since the minimal space-time product may not indicate the correct operating point in a multiprogramming environment. This criterion is not at all applicable in the context of file migration. We prefer to leave it to the reader or system implementor to pick the most suitable operating point. In some cases, a miss ratio of 10% may be tolerable, in other cases, 1% or less may be appropriate.

It is fairly straightforward to compute the values for S(A,P) and M(A,P) for a given algorithm A and parameter P. We make the following definitions to aid in our computations:

Nref(i) - number of times (days) that file i is referenced.

Nref -total number of references to all files (=213,692).

Niref(i) - number of interreference intervals to file i = Nref(i)-1 if Nref(i)>=1; 0 otherwise.

Niref - total number of interreference intervals to all files = 189,222.

Sz(i) - size of file i in tracks (sometimes abbreviated to Sz).

Dayno - the number of days in the measurement period (256 working days).

I(i,j) - the length of the j'th interreference interval for the i'th file.

For each interreference interval I(i,j), the file will be retained in memory after the reference beginning the interval for some number of days Kp(A,P,i,j), where A and P are again the algorithm and the parameter value. We will generally abbreviate this as Kp(i,j). We define this in such a way that Kp(i,j)>=1; that is, the initial day of reference is included in the residence time. Clearly, if Kp(i,j)<=I(i,j), then there is a fault at the time of the next reference, otherwise there is no fault. Let $F(A,P,i,j)$ (abbreviated $F(i,j)$) be either 0 if there is no fault or 1 if there is a fault. Then we compute $S(A,P)$ and $M(A,P)$ as follows:

$$S(A,P)= \sum_{i,j} (\min[Kp(i,j),I(i,j)]Sz(i)/Dayno \qquad (2)$$

$$M(A,P)= \sum_{i,j} F(i,j)/Niref \qquad (3)$$

We note that our method of computation implies a small inaccuracy. We have ignored the boundary condition of a finite measurement period - files in existance at the end of our measurement period have an unknown interreference interval, as do files in existance at the beginning. We have chosen to omit the contribution to $M(A,P)$ in both cases, and we have not counted the first reference to a file as causing a fault. Thus we assume that the file is resident on the disk at the time of the first reference to it. We are interested only in comparative results (among algorithms) and there is no reason to think that the direction of the comparison would have shifted through such a simplification.

It should be clear that there is a tradeoff of space for file faults (instances of a missing file). That is, if $Kp(i,j)$ is reduced, the value of $M(A,P)$ will either increase or remain the same, and the value of $S(A,P)$ will decrease. More generally, for all stack algorithms (Mattson et al., 1970), it can be shown that the (S,M) curve is monotonically nonincreasing for increasing S. All of the algorithms that we consider will be stack algorithms. We also comment that our criterion for algorithm evaluation tends (as is evident from eqs. 2 and 3) to make large files much better candidates for replacement than small files.

From equations 2 and 3, it can be seen that an optimal lookahead algorithm would select any file which has just been referenced and remove it iff for its upcoming interreference interval, Sz(i)(I(i,j)-1)>P; otherwise the file would be retained until the next reference. By varying P, an (S,M) curve is traced out. This algorithm is variously known as VVMIN or VOPT (see Denning and Slutz, 1977 for details); we choose to call it VVMIN to maintain consistency with the related VMIN algorithm (Prieve and Fabry, 1976). VMIN is the algorithm which removes all files (or pages) whose time to next reference is greater than P. VMIN is optimal only in the case of fixed size files or pages. VMIN and VVMIN will be used for purposes of comparative evaluation later in this paper.

V. REPLACEMENT ALGORITHMS - DEFINITIONS AND DERIVATIONS

In this section we describe and/or derive the remaining file replacement algorithms considered. All of the algorithms specified will be of the "variable space" variety, which means that the total volume of on-line files may vary even though the parameter value P for the replacement algorithm remains fixed. This is in contrast to the fixed space algorithms such as LRU, FIFO, MIN, etc., which always maintain a fixed volume of online files or pages.

There are two reasons for dealing only with variable space algorithms. First, the variable space algorithms that we consider allow the decision on whether to keep a file online to be made without reference to other files; thus the computation time is kept low both in a real system and in our trace driven simulations. Second, implementing most fixed space algorithms exactly would require more information than we have. For example, LRU would require the full sequence of file references (or opens and closes) and not just knowledge of which file was used on which day.

At the end of the last section, we described the optimal lookahead replacement algorithms VMIN and VVMIN. We proceed in this section to consider realizable algorithms, beginning with probabilistically "optimal" ones and moving to algorithms which are successively more ad hoc and less likely to perform well.

A. A Stochastically Optimal Algorithm

It was noted earlier that the optimal but unrealizable algorithms VMIN and VVMIN used the known time to next reference to determine when to remove a file. A realizable algorithm must of course deal only with known information, i.e. past history, and therefore it can at best estimate which file will not be used. We derive a strategy to do this as follows:

Let Q(A,H,t,Sz,C,P) be the policy for whether to remove a file; that is, Q(A,H,t,Sz,C,P) specifies whether the file should be kept or removed, given the values of the parameters. A is the algorithm - throughout this section (V.A) we shall consider the stochastically optimal policy (denoted Stochopt, abbreviated Sopt) only. Unless it would cause confusion, we shall omit A as a parameter. H is the entire previous history of reference to the file, t is the time (date), Sz is the size of the file, C is the class of the file and P is the "cost" of fetching the file when it is next referenced (should we decide to remove it). P in this case is the parameter and will be varied to produce the (S,M) curve; its value is only useful in specifying an operating point. We have already assumed that files are independent (!) of each other, so the optimal policy Q includes only information relevant to the file in question. Further, for files with identical values of H, t, Sz, C and P, the policy will be the same.

It is both possible to reduce the number of parameters for Q (since some of them are not helpful) and desirable (since there are more parameters than we can deal with). We first assume that file reference patterns are independent of time, all other things being equal. To some extent this is a simplification, since summer usage patterns should differ from winter, etc., but it doesn't appear to be unreasonable. Thus we replace "t" as a parameter with "a" for the age of the file (in days since it was created). Our analysis (Smith, 1978b) has shown little if any serial correlation between interreference intervals, so we choose to replace H, the entire previous reference history of the file with h, the time since the last reference. We also choose to consider only demand fetch policies; i.e. only that fetch algorithm which fetches a file at the time it is referenced and not before. In this case, the optimal policy Q contains more information than necessary, since the file will only be removed once in an interreference interval.

Therefore, Q is replaced with the optimal (equivalent) policy K(A,a,Sz,C,P), where K is the number of days the file is to be kept resident from and including its last day of reference, A=Sopt and a, Sz, C and P are as before. K and Q are equivalent, but K is simpler to compute and easier to use.

Referring to earlier notation, Kp(i,j) will in each case be computed using the policy K.

K must be computed probabilistically, based on empirical distributions. We would thus like to measure R(a,Sz,C) which is the empirical distribution of time (in days) between references to a file, given that it is of size Sz, class C and is age a at the date of its last reference. Again the size of the parameter space is too large, primarily in that we have insufficient data to estimate distributions reliably. Therefore, in almost all cases "a" is dropped as a parameter, and Sz is replaced by Logsize (L); i.e. the size class into which Sz falls. The resulting distribution is referred to as g(i,L,C), which is the empirical distribution of the times between references to a file of size class "L" and type class "C". G(i,L,C) is the cumulative distribution and gf(i,L,C) and Gf(i,L,C) are the fitted distributions (see equation 1).

Finally, we make one additional simplification. Experiments which appear in the next section (VI.B) tend to indicate that the class C of the file is not very useful in selecting a file for replacement. For this reason, and because the size of the computation is unpleasantly large (see equation 4), we have also dropped the class C from our computations.

Computing K(Sopt,Sz,P) (these are the only parameters left) is straightforward given the distribution g(i,L,*) ("*" indicates that the distribution has been aggregated over all values for that parameter). Thus:

$$K(Sopt,Sz,P) = \{k \mid \min_k [\sum_{i=1}^{k} g(i-1,L,*)(i-2)Sz +$$

$$+ (1-G(k-1,L,*))((k-1)Sz+P)] \} \qquad (4)$$

This equation simply selects that value of k which minimizes the cost of an interreference interval, where the cost of a fault is given by the parameter P. This is equivalent to specifying the cost of the algorithm as a whole as:

$$\sum_{i,j} \min[Kp(i,j),I(i,j)] \times Sz(i) + \sum_{i,j} F(i,j)P \qquad (5)$$

$$= S(A,P) \times Dayno + M(A,P) \times Niref \times P \qquad (6)$$

where (P x Niref)/Dayno is just the constant of proportionality that relates the relative impact of S(A,P) and M(A,P) on the total cost of the replacement policy. We have selected a policy K(Sopt,Sz,P) that minimizes (6); i.e. it implicitly specifies an operating ~~point. By varying P, a~~ curve in the (S,M) plane is traced out.

There is one problem which will occur in our evaluation of the Stochopt algorithm; for the most part, we shall use the empirical interreference distribution to determine g(i,L,*) and K(Sopt,Sz,P) and then go back and compute M(Sopt,P) and S(Sopt,P) over the same interreference intervals. This, however, is not an important factor. First, we note that there are 189,222 interreference intervals; thus the influence of a single interreference interval in distorting the distribution should be negligible. Further, we also experimented (as explained later) by calculating K(Sopt,Sz,P) for a subset of the files and then measuring M(Sopt,P) and S(Sopt,P) for the remaining files.

In table II, we show the values for K(Sopt,Sz,P) for a range of sizes Sz and for those values of P which yield values of H(Sopt,P) of 1%, 5% and 10%.

B. Expected Time to Next Reference

A simpler approach than that used above to selecting the file whose ((time to next reference) x (size)) is maximal is to estimate the (mean) expected time to next reference. This method is not optimal, since using the expected time to next reference is only an approximation to the approach of the last section. A counter example showing this is provided in Coffman and Denning (1973) for fixed size files (i.e. pages). The reason for this non-optimality can be shown by the following example: Let the distribution of time to the next reference be bi-valued, with the probability .5 that the reference is in 1 day and .5 that the reference is in 99 days; then the expected time to the next reference is 50 days. The optimal policy would likely keep the file for one more day (which means that there is a 50% chance of using the file) and then if it isn't used, discard it. A policy based on the expected time to next reference would probably discard the file immediately, since the expected time is so large.

The expected time to next reference ($E(i,L,C)$, where i is the number of days since the file was last referenced; i=0 implies that the file was referenced that day) can be computed from the interreference distribution g as follows:

$$E(i,L,C) = \sum_{j=i+1}^{\infty} g(j,L,C)(j-i)/(1-G(i,L,C)) \tag{7}$$

The algorithm for file removal is then to remove any file for which

$$(E(i,L,C)-1) \times Sz > P \tag{8}$$

It is also possible to suppress either or both of L and C and just compute $E(i,*,C)$, $E(i,L,*)$ or $E(i,*,*)$. The file removal decision can then be based on only one or neither of L and C. We note that the size of the computation is no longer a problem, so it has not been necessary to reduce the parameter space as was done in Stochopt.

We let "Etnr" denote the file removal algorithm specified by equation 8. Then we can define a policy K(Etnr,Sz,C,P) as follows:

$$K(Etnr,Sz,C,P)=\{i \mid \min_i[\ ((E(i,L,C)-1)\ \times\ Sz\ ) > P\ ]\ \} \tag{9}$$

The fitted distribution gf can be used in place of the empirical distribution g. The algorithm in that case is named "Etnrf" (f for "fitted") and we compute it in the same manner as in equation (7):

$$Ef(i,L,C)=\sum_{j=i+1}^{\infty} gf(j,L,C)\,(j-i)/(1-Gf(i,L,C)) \tag{10}$$

We define the policy K(Etnrf,Sz,C,P) as:

$$K(Etnrf,Sz,C,P)=\{i \mid \min_i[\ ((Ef(i,L,C)-1)\ \times\ Sz\ ) > P\ ]\ \} \tag{11}$$

It occasionally happens that K(Etnr,Sz,C,P) = 1; that is, the file is always removed at the end of the day on which it is referenced. This seems counter-intuitive to many people; thus we define an additional policy "Etnrb" (b for "bound") as equivalent to Etnr expect that all files are kept for at least two additional days after the day on which they are referenced. Thus

$$K(Etnrb,Sz,C,P)=\max(3,\{i|\min_{i}[((E(i,L,C)-1) \times Sz) > P]\}) \quad (12)$$

In figure 4 we show plots of E(i,L,*) and Ef(i,L,*). Figure 5 displays the values of E(i,*,C) and Ef(i,*,C). The dotted lines in all cases show Ef. There are two important observations to be made from these figures. First, the expected time to next reference is generally increasing with the time since the last reference; thus the desirability of retaining a file will decline with the time since the last reference. Second, the values of Ef are a mediocre fit at best to E. Despite this relatively poor fit, it will be observed when we show our experimental results that the K(Etnrf,Sz,C,P) policy is reasonably useful.

~~Earlier in section V we elected to drop file age as a~~ parameter for a replacement algorithm, on the basis that the space of parameter values was too large. We do try one age based algorithm in our experiments, however. Define RA(i,L,C) to be the rate of reference to a file of age i, given that it is of the specified size and class. Specifically, RA(i,L,C) is the probability that a file that is exactly i days old will be referenced on the i'th day of its lifetime. In (Smith, 1978b), values for RA(i,L,C) are given. We iteratively compute the expected time to next reference as a function of the file age i, and the file size and class as:

$$Ea(i,L,C) = RA(i+1,L,C)+(1-RA(i+1,L,C)) \times (1+ Ea(i+1,L,C)) \quad (13)$$

Figure 5

EXPECTED TIME TØ NEXT REFERENCE

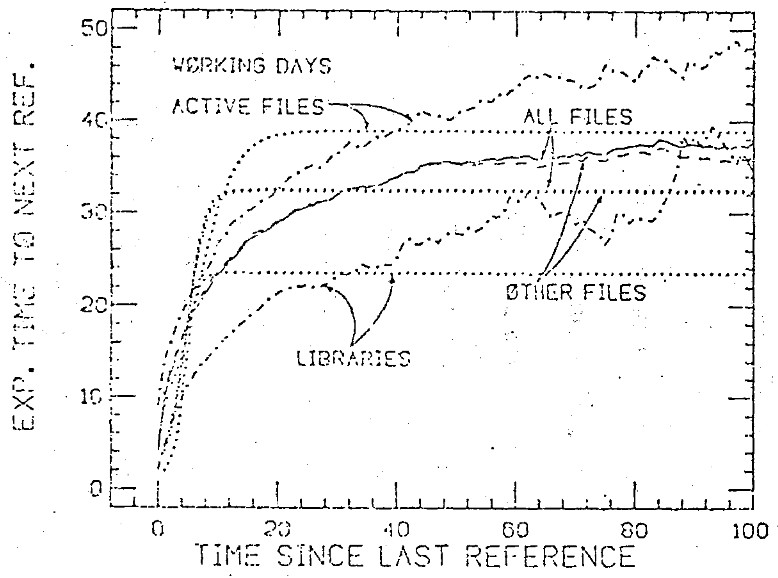
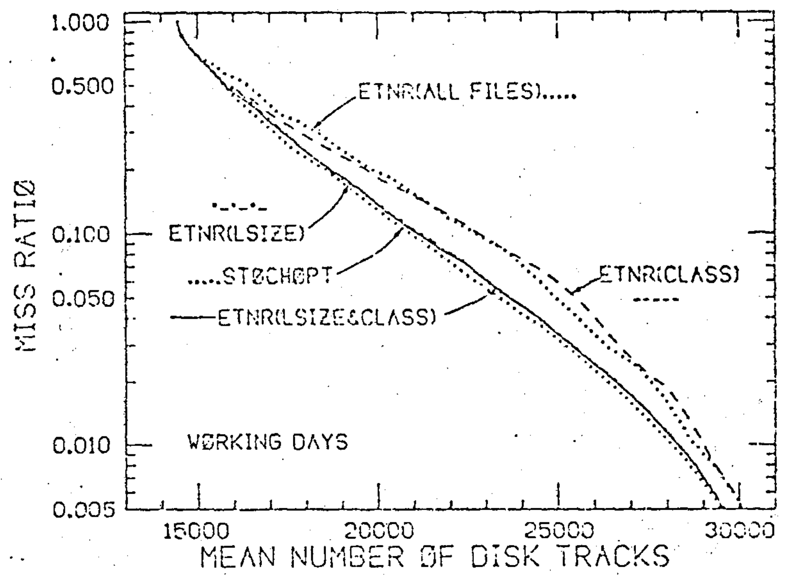


Figure 7

MISS RATIØ VS. SIZE



MISS RATIØ VS. SIZE

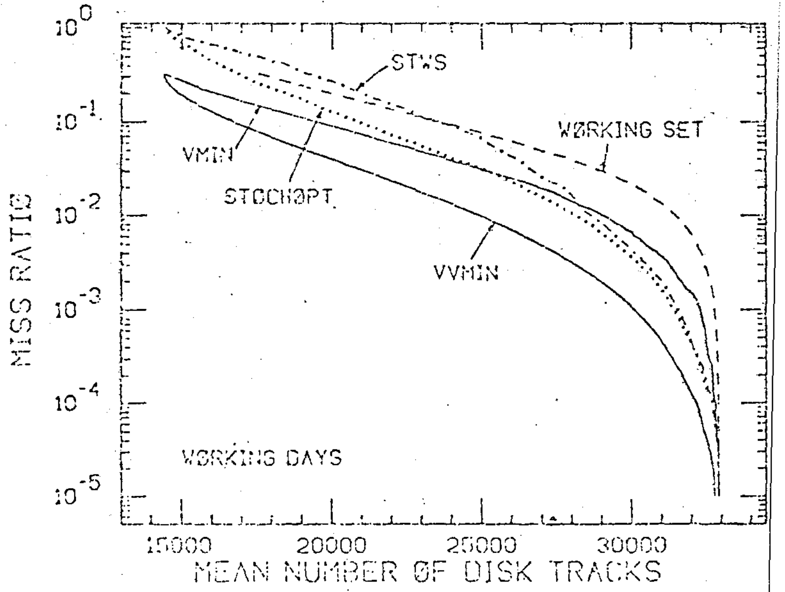


Figure 6

MISS RATIØ VS. SIZE

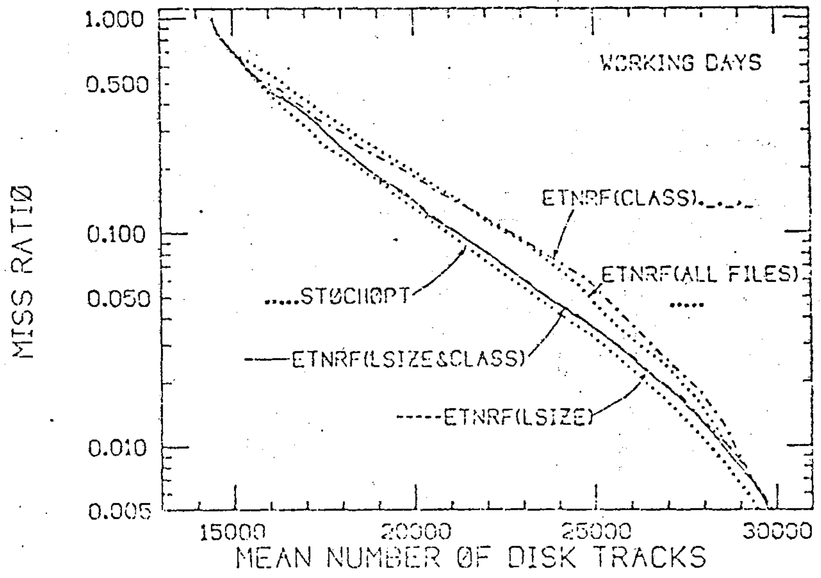


Figure 8

where a value is selected for some Ea(i,L,C) for which i is sufficiently large; the function behaves in such a way (since (1-EA(i+1,L,C))<1) that for sufficiently smaller i, the boundary condition is irrelevant.

The algorithm works as follows: Compute the value for Ea(i,L,C) for the file on the day it is referenced. If (Ea(i,L,C) x Sz) > P, remove the file immediately; otherwise hold the file until the next reference. (This algorithm would likely be improved by recomputing each day the file was on-line). We refer to this algorithm as "Etnra" and the policy can be specified as:

$$K(\text{Etnra},Sz,C,a,P) = 1 \text{ if } ((Ea(a,L,C)-1) \times Sz > P); \quad \text{infinity otherwise.} \qquad (14)$$

This algorithm is used in our experiments later.

C. Time and Space-Time Algorithms

All of the realizable algorithms defined thus far (Stochopt, Etnr, etc.) have relied on interreference time distributions for the file reference process. It is also possible to define some algorithms which do not use measured data, but implicitly assume some model of file reference behavior. In this section, we define algorithms called Working Set (WS), Space-Time Working Set (STWS) and STP**y.

Working Set (Denning, 1968) is that algorithm which removes a file when it has been unreferenced for P or more days. This algorithm was originally designed for main memory paging, and in that circumstance it has been shown to work very well. It has the defect in this case that it takes no account of file size, and thus small files are as likely to be removed as large files. We let "WS" refer to the working set algorithm. Then the policy K(WS,P) may be defined as:

$$K(WS,P)=P+1 \qquad (15)$$

Space Time Working Set is the straightforward and obvious extension of working set; it removes any file for which the product (time since last reference) x (file size) is greater than the parameter P. The implicit assumption here is that the file that is likely to incur the largest cost of retention to the next reference is that which has already accumulated the largest retention cost since the last reference. The policy in this case may be written:

$$K(STWS,Sz,P)=\lfloor P/Sz \rfloor+1 \qquad (16)$$

It can be observed from figures 4 and 5 that the expected time to next reference climbs (initially) quite steeply with time since last reference. In (Smith, 1978b) it was shown that larger files are used more frequently than smaller files. These two facts would suggest that a modification of STWS which weighted time since last reference more heavily than file size should perform better than STWS. We therefore define a class of algorithms which we denote as STP**y (STP stands for space-time product, y is a parameter and ** is the exponentiation operator in many programming languages). For the algorithm STP**y, the following is computed: (Sz x (time since last reference)**y); that is, the time since the last reference is raised to the (real valued) exponent y. If the value of the expression given is greater than the parameter P, the file is removed. This policy can be expressed as:

$$K(STP^{**}y, Sz, P) = \lfloor (P/Sz)^{**}(1/y) \rfloor + 1 \qquad (17)$$

Our selection of exponentiation as the way to increase the weighting of time relative to file size is based simply on convenience; many other functions could have been defined instead.

D. Bernoulli Process Algorithms

In Stritter (1977) it was stated that most files observed displayed reference patterns that could be characterized as "Poisson". We assume that Stritter treated what was a discrete time time series as a continuous time time series. In any case, we found in (Smith, 1978b) that of those files testable, the majority could not be characterized as Bernoulli (and presumably therefore not Poisson). Never the less, we decided to experiment with some file replacement algorithms based on the assumption that the actual file reference process was Bernoulli for each file (with a possibly different rate for each file). The Bernoulli process (geometric interarrival times) is such that the expected time until the next reference is constant, regardless of the time since the last reference. Thus the replacement decision can be made immediately after reference to a file; the file will either be removed immediately (that night) or be kept until the file is used again. The only (!) problem is to estimate the rate at which the file is being referenced, or equivalently, the expected time to the next reference (which is the reciprocal of the rate of reference).

A simple way to estimate the time to the next reference in a Bernoulli process is to use the "exponential" estimator as follows: Let Z(i) be the current estimate of the expected time to next reference for the file. Let an additional interreference interval occur of length W. Then the new value of Z(i+1) is

$$Z(i+1) = y\, Z(i) + (1-y)\, W \quad \text{for } 0 <= y <= 1 \qquad (18)$$

This type of estimator is discussed by Denning and Eisenstein (1971). It is not optimal if the reference process is completely stationary (in which case the simple average of previous interreference intervals is better), but if there is a slow trend in the reference process, this estimator should adapt better than the simple average.

We initialize Z (which we can call Z(0)) to the mean time between references for the size and class of the file in question. Thereafter, a separate value of Z is kept for each file.

Our notation for the algorithm described is "Exp=y" where y is the parameter shown in equation (18). It is tedious to specify this algorithm precisely as a policy (since it depends on all of the past history in the computation of Z, the expressions get cumbersome), but the following should be clear enough:

$$K(Exp=y, H, Sz, C, P) = 1 \text{ if } Z>P; \text{ infinity otherwise} \qquad (19)$$

where Z(0)=E(0,L,C) and Z(i+1) is computed from Z(i) as shown in equation (18). H is the complete past history of the file.

y should take on. Our experiments tested this algorithm with five values of y: y=0.0, .25, .50, .75 and 1.00. y=0.0 means that the estimate for the next interreference interval is that it will be of the same length as the current interval. y=1.0 says that every interreference interval is expected to be of the same length as the mean interreference interval for this size and class. The other values for y have intermediate meanings.

VI. EXPERIMENTAL RESULTS

A. Methodology

As noted earlier, our data consists only of one bit for each file for each day specifying whether that file was used that day. Therefore, we have had to do our experiments in a manner compatible with that restriction. Thus we assume the following: all files referenced on a given day are fetched at midnight + e (e->0) of that day and are retained on the disk until the end of the 24-2e hour period at midnight-e. At that time, file migration takes place, and any file which is to be removed is removed at that time. Thus every time a file is referenced, it remains on a disk for at least one full day. Also, it is possible for a file to be referenced on two consecutive days and still experience a file fault on the second of those days. We have stated earlier that we are omitting the intial file faults from our fault counts (see section V.A) and are omitting the space contribution by a file after the time of its last reference.

Our algorithm for computing M(A,P) and S(A,P) is as given in equations 2 and 3; we considered each interreference interval in turn for each algorithm and parameter value.

B. Miss Ratio Comparisons

Figures 6 through 13 give the performance of each of the algorithms described in the last two sections. Each is discussed below.

Figure 6 shows the behavior of the VVMIN, VMIN, STWS, Working Set and Stochopt algorithms. We see that VVMIN, as expected, is the best of all of the algorithms by a substantial margin; it experiences a miss ratio about one-third as high as the best realizable algorithm (Stochopt) througout much of the range of operation. VMIN, conversely, performs relatively poorly because it doesn't consider file sizes, even though it is a look-ahead algorithm. Working Set also does very poorly for the same reason. STWS, which does take into account the file size, acts fairly well above about 28,000 tracks, but isn't very good for smaller space allocations.

The Etnr class of algorithms (equations 7-12) are shown in figure 7. We see that Etnr(Lsize,class) and Etnr(Lsize) perform well and are very close to Stochopt. Etnr(all files) and Etnr(class) perform relatively poorly. Interestingly, Etnr(class) is not uniformly better than Etnr(all files) and Etnr(Lsize,class) is not uniformly better than Etnr(Lsize). Although it was noted earlier that the Etnr algorithms were not in any sense optimal, it was expected that the more accurate E(i,-,-), the better the performance of the algorithm. We find that this is not necessarily so. We also note that the critical item in computing the Etnr policy is the parameter Lsize; the class has little if any effect. This tends to validate our decision to exclude the class in the computation of Stochopt. (Although the functions E(i,*,C) (figure 5) vary widely with the class, the number of interreference intervals for which F(i,j) switches between 0 and 1 is so small that the miss ratio curves are barely affected. I.e. only about 30% of the interreference intervals belong to libraries or active files, and of those, a very large fraction (about 65%) are interreference intervals of 1 to libraries.)

The fitted function Ef(i,L,C) is used for the results presented in figure 8. Comparing this figure with figure 7, we see that the use of Etnrf is almost as satisfactory as the use of Etnr despite the poor quality of the fit between E(i,L,C) and Ef(i,L,C). This comparison is shown again directly in figure 9 where the Etnr(Lsize,class) and Etnrf(Lsize,class) policy results are both given. Their closeness is again evident.

Figure 9 also shows the performance of Etnrb (keep a file at least 2 extra days; otherwise use the Etnr algorithm) and Etnra (remove based on expected time to next reference calculated as a function of age at last reference). Keeping all files a couple of days as a minimum appears to be a poor policy; the original method of optimization is better. Etnra performs very poorly.

A variety of space-time algorithms are compared in figure 10. As we observed earlier, an algorithm which weighted the time since last reference more heavily than the file size would be expected to perform better than STWS. We tested STP**1.2, STP**1.4 and STP**.8 against STWS (which is STP**1.0) and the results appear in figure 10. STP**1.4 appears to provide the best performance of any of these

Figure 9

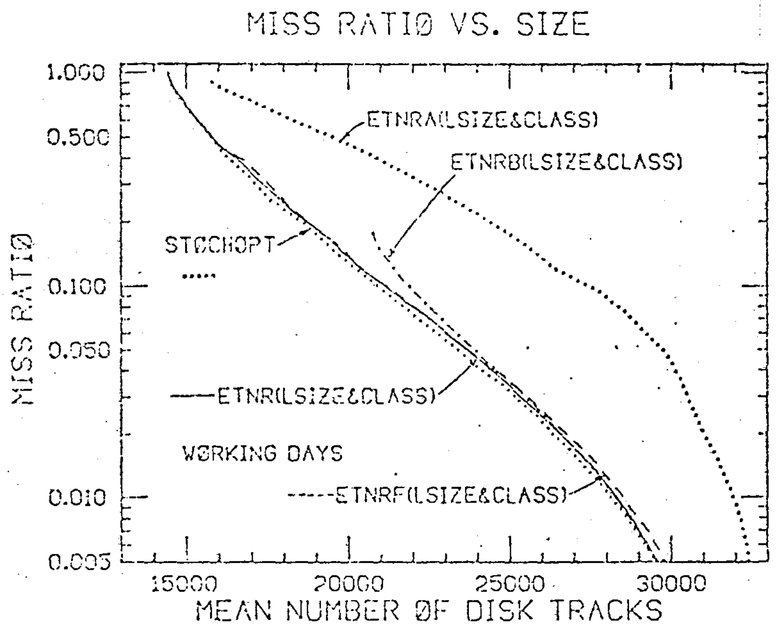
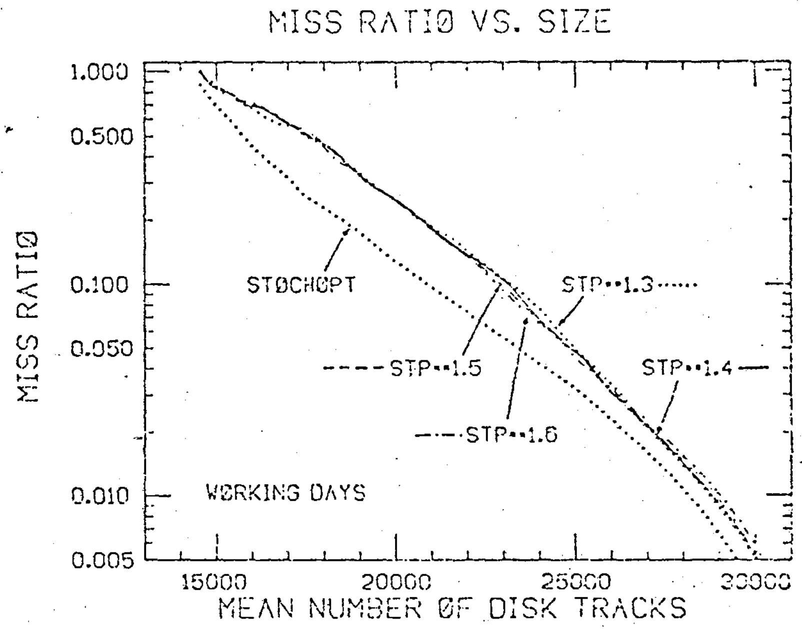


Figure 11



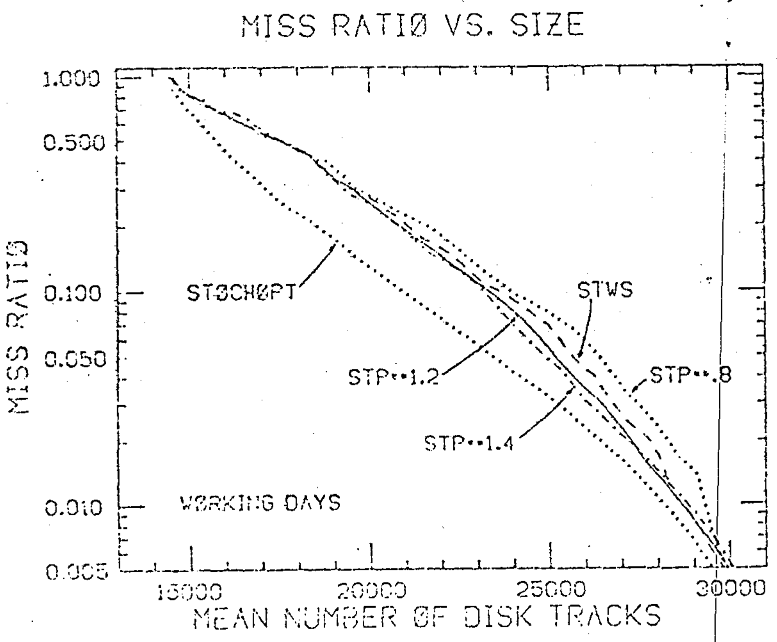


Figure 10

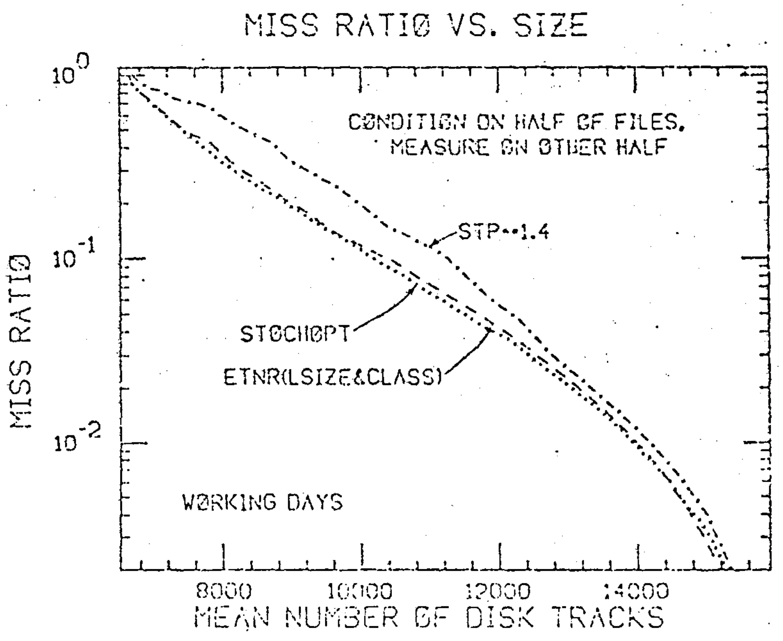


Figure 12

algorithms. Additional parameter values were tested (STP**1.5, STP**1.6, STP**1.3) and these results are shown in figure 11. These three and STP**1.4 all perform about equally well, but none of them seems to be a good substitute for Stochopt.

The problem of creating Stochopt on the basis of measured data and then testing it against the same data was mentioned in section V.A. A test for the robustness of Stochopt and Etnr(L,C) was run by creating the Stochopt and Etnr(L,C) policies (equations 4,9) using half of the files on the system and then measuring the miss ratio on the other half of the files. This was done and the results appear in figure 12, where measurements for Stochopt, Etnr(L,C) and STP**1.4 are given. It can be seen that Stochopt and Etnr continue to perform much better than STP**1.4. (We also observe that Stochopt is no longer uniformly better than Etnr).

The results of our experiments with the Bernoulli process algorithms are given in figure 13, where it can be seen that none of these algorithms are even close to performing acceptably. This constitutes fairly strong evidence that the bulk of the file reference processes cannot be characterized as Bernoulli or Poisson.

C. Parameter Selection

Each of the algorithms presented in this paper contains a parameter, the value of which specifies when to keep or remove a file from the disk. In general, our interest is not in the parameter value per se, but in the value of the parameter that will yield an acceptable miss ratio. The first question is: what is an acceptable miss ratio? The average user who is logged on uses a mean of 3.41 files that day. If fetching a

Figure 13
MISS RATIØ VS. SIZE

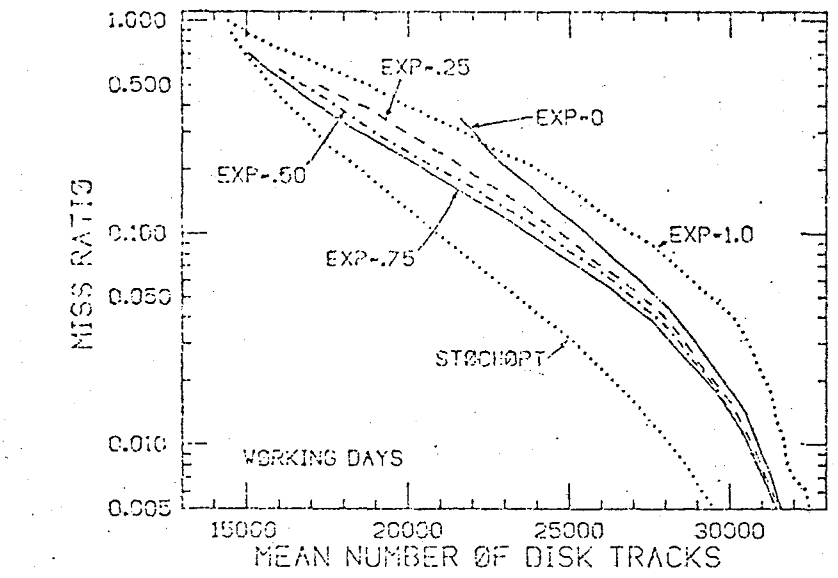
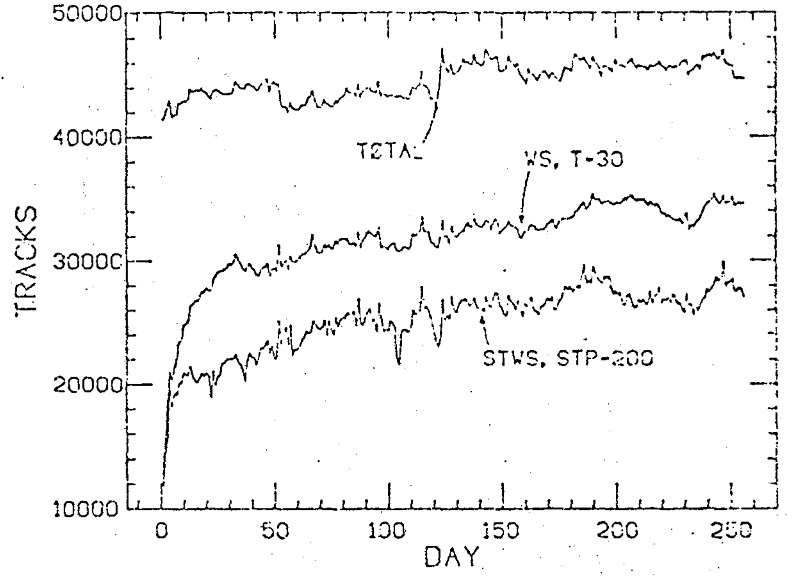


Figure 15
VØLUME ØF ØN-LINE FILES



MISS RATIØ VS. PARAMETER VALUE

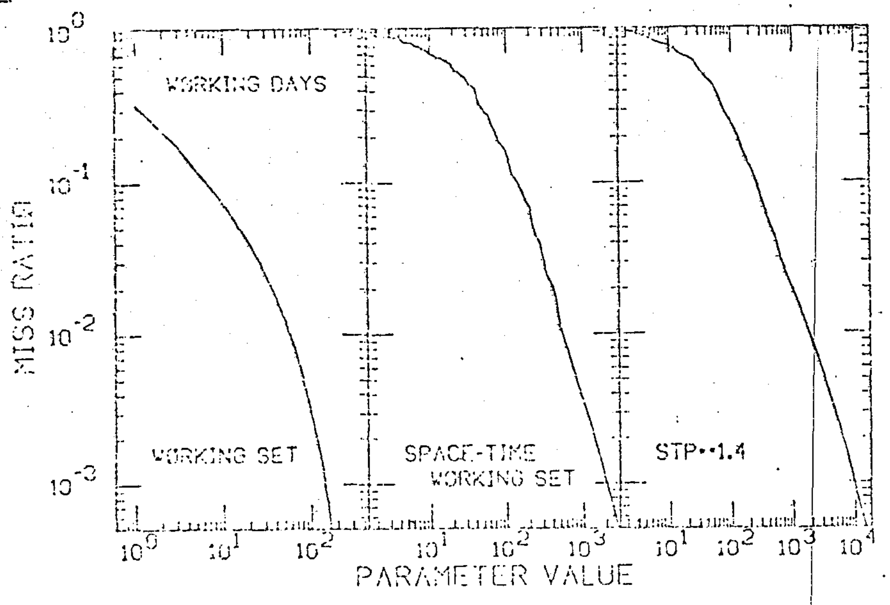


Figure 14

file from mass storage to disk were to take one minute, then a 1% miss ratio would imply .0341 man-minutes/user/day. A mean of 183.5 users log on per day, so the loss would be 6.26 man-minutes/day. A 10% miss ratio would give a figure exactly ten times as large or a little more than a man-hour per day. The one minute figure may be far too optimistic, however, since it assumes no queueing delays. (The average access time on most mass storage devices, such as the IBM 3850, the Ampex Terabit memory, the CDC 38500, etc. is on the order of 1 minute or less.) For example, at the Lawrence Livermore Laboratory, access times, including queueing (and device malfunction) delays, can stretch to hours to get a data set off of the photostore and tens of minutes are typical. We note in particular a batch arrival phenomenon, by which most users will log on early in the day and attempt to read several of their files; thus the mass storage device would be very congested early in the day. Our intuitive feeling is that miss ratios on the order of 1% to 10% would be the maximum tolerable by most users.

For several of our algorithms, the approximate parameter values that yield 10%, 5% and 1% miss ratios are given in table III. Thus the reader wishing to implement one of our algorithms can just use the figures given. The policy K(Sopt,Sz) appeared earlier in table II.

A more complete mapping of parameter values into miss ratios is given in figure 14 for the Working Set, STWS and STP**1.4 algorithms.

D. Variable Space Buffering

Figure 15 shows the volume of on-line files vs. time using the working set algorithm with a parameter P=30, the

STWS algorithm with a parameter P=200 and the total volume of on-line files in the original, unmigrated system. As is evident, the volume of on-line files varies considerably from day to day and month to month. We note that the figures for Working Set and STWS do not become stationary (i.e. reach warm start) until 30 and 200 days respectively from the start of the measurement period. Initially, all files are assumed to be offline.

The difficulty with this day to day variation in the volume of on-line files is that (a) the parameter value used to reduce the number of on-line files may have to change from day to day and (b) enough space has to be left on the disks after a migration run that user file fetches are unlikely to cause space to run out during the following day. We also note that many of our file migration algorithms favor the larger files for replacement; thus the files fetched are also likely to be large.

Table IV shows the approximate mean volume of the files fetched/day for several of the algorithms discussed and for three different miss ratio values. Also given is the volume of files fetched as a ratio to the mean volume of on-line files.

VII. CONCLUSIONS, APPLICABILITY OF RESULTS, FUTURE WORK

In this paper we have developed and evaluated a number of algorithms for the migration of files from disk to mass storage. Most of these algorithms have been of the class that use a reference as a renewal point; that is, history prior to the most recent reference is not used. We found that the Stochopt algorithm, which uses the measured file interreference time distribution, performed well and the Star algorithm, which also uses this information, did almost as well. Algorithms which used less or no information about file reference patterns generally performed poorly. About the best of the other algorithms was the STP**1.4 algorithm, which might provide acceptable performance.

The evaluation of the algorithms presented was done using file reference data taken at SLAC for Wylbur text editor data sets. There is no reason to feel that text editor or time sharing data sets (e.g. TSO) would be referenced very differently in another system; we therefore believe that our results are applicable to such systems. Our data, however, is not concerned with large data files, system files or scratch files and no conclusions can be drawn about migrating such files based on the experiments described in this paper. Further, our results are for "long term" file migration; that is, migration that occurs over periods of days, weeks or months. In many systems, it would be necessary for migration to occur over time periods of hours and our experiments here provide no guidance in such cases.

An important point, which must be noted, is that for file migration to be effective it must cause no significant inconvenience to the user. If migration is performed in such a way that the user has to expend significant effort in retrieving his files or has to wait an "unreasonably" long time, users will develop defense mechanisms against migration. That is, users will either write programs that will, or will themselves open every file they own sufficiently frequently that their files never get migrated. Precisely this type of behavior has been observed by the author at the Lawrence

Livermore Laboratory (which scratches disk copies of files over periods of hours) and at IBM Research, San Jose, (which migrates files over periods of weeks). A good implementation for file migration would leave the user in complete ignorance of the actual location of his file; occasionally, getting to a file might take one or two minutes rather than 10 or 20 seconds. (That one or two minutes can easily been hidden in the response times of many systems.)

We found that if file migration is properly implemented, it can substantially reduce the volume of on-line files without inflicting an unacceptably high miss ratio on most users.

A number of additional investigations are possible using the data available. Algorithms which use the file age more intelligently can be investigated. Prefetch algorithms, based on more sophisticated policies than the class of demand policies that we consider, may be worthwhile. Placement algorithms can be considered to some extent. Some marginal improvements can be obtained by investigating these additional items; we believe, however, that in this paper we have presented the bulk of the useful information. We very much hope that similar data will become available for other computer systems in order to make comparisons possible and in order to investigate the range of applicability of our results.

ACKNOWLEDGEMENTS

Many thanks to Dr. Edward Stritter who collected this data and made it available to the author for this research. **This work was supported by the U. S. Department of Energy under Contract W-7405-ENG-48.**

BIBLIOGRAPHY

Donald L. Boyd, "Implementing Mass Storage Facilities in Operating Systems", Computer, 11, 2, February, 1978, pp. 40-45.

R. H. Chaffee, M. A. Challenger and E. S. Russell, "File Migration Task Force Study", June, 1977, Stanford Linear Accelerator Center.

Wesley Chu and Holger Opderbeck, "Program Behavior and the Page Fault Frequency Replacement Algorithm", Computer, November, 1976, pp. 29-38.

Edward G. Coffman and Peter J. Denning, Operating Systems Theory, Prentice Hall, Englewood Cliffs, New Jersey, 1973.

J.P. Considine and J. J. Myers, "MARC: MVS Archival Storage and Recovery Program", IBM Sys. J., 16, 4, 1977, pp. 378-397.

Peter J. Denning, "The Working Set Model for Program Behavior", CACM, 11, 5, May, 1968, pp. 323-333.

Peter J. Denning and Bruce Eisenstein, "Statistical Methods in Performance Evaluation", Proc. ACM Workshop on Computer Performance Evaluation, April, 1971, Harvard University, Cambridge, Mass., pp. 284-307.

Peter J. Denning and Donald R. Slutz, "Generalized Working Sets for Segment Reference Strings", Computer Science Report, revised September, 1977, Purdue University.

DOE, 1977, "Proc. DOE/NCAR Mass Storage Workshop", December, 1977, National Center for Atmospheric Research, Boulder, Colo., published May, 1978.

Donald E. Eastlake, "Teriary Memory Access and Performance in the Data Computer", Proc. Third Int. Conf. on Very Large Data Bases, Tokyo, Japan, October, 1977, pp. 259-267.

Roger Fajman and John Borgelt, "Wylbur: An Interactive Text Editing and Remote Job Entry System", CACM, 16, 5, May, 1973, pp. 314-322.

IBM, 1978, "MVS Hierarchical Storage Manager Release 1 is Available", DPD Program Product Announcement, IBM Corp., Armonk, N.Y., April, 1978.

Jeremy Knight, "CASHEW - A Proposed Permanent Data Storage System", Computer Center Report, Lawrence Berkeley Laboratory, May, 1976.

V.Y. Lum, M.E. Senko, C.P. Wang and H. Ling, "A Cost Oriented Algorithm for Data Set Allocation in Storage Hierarchies", CACM, 18, 6, June, 1975, pp. 318-322.

R. L. Mattson, J. Gecsei, D. R. Slutz and I. Traiger, "Evaluation Techniques for Storage Hierarchies", IBM Sys. J., ‘9, 2, 1970, pp. 78-117.

Howard Morgan, "Optimal Space Allocation on Disk Storage Devices", CACM, 17, 3, March, 1974, pp. 139-142.

Howard Morgan and K. Dan Levin, "Optimal Program and Data Locations in Computer Networks", CACM, 20, 5, May, 1977, pp. 315-322.

Barton G. Prieve and R. S. Fabry, "VMIN - An Optimal Variable Space Replacement Algorithm", CACM, 19, 5, May, 1976, pp. 295-297.

Ron Revelle, "An Empirical Study of File Reference Patterns", IBM Research Report RJ 1557, April, 1975.

Adrian Segall, "Dynamic File Assignment in a Computer Network", IEEE Trans. on Automatic Control, AC-21, 2, April, 1976, pp. 161-173.

Alan Jay Smith, "Bibliography on Paging and Related Topics", submitted for publication, August, 1978a.

Alan Jay Smith, "Long Term File Migration, Part I - File Reference Patterns", August, 1978b, submitted for publication.

Edward P. Stritter, "File Migration", Stanford Computer Science Report STAN-CS-77-594, (Ph.D. Dissertation), January, 1977.

UCC, 1977, "Out of Disk Space?", Advertisement, Datamation, December, 1977, p. 57.

David Zehab and Stephen J. Boies, "The SFS Migration System", IBM Research Report RC 6944, January, 1978.