A Study of File Sizes and Functional Lifetimes

M. Satyanarayanan

DEPARTMENT OF COMPUTER SCIENCE

CARNEGIE-MELLON UNIVERSITY

1. Introduction

The performance of a file system depends strongly on the characteristics of the files stored in it. This paper discusses the collection, analysis and interpretation of data pertaining to files in the computing environment of the Computer Science Department at Carnegie-Mellon University (CMU-CSD). The information gathered from this work will be used in a variety of ways:

1. As a data point in the body of information available on file systems.
2. As input to a simulation or analytic model of a file system for a local network, being designed and implemented at CMU-CSD [1].
3. As the basis of implementation decisions and parameters for the file system just mentioned.
4. As a step toward understanding how a user community creates, maintains and uses files.

2. Data Collection

2.1. The Environment

The data used in this paper was obtained on a Digital Equipment Corp. PDP-10 Model KL-10 processor [7] with 1 Mword of primary memory and eight 200 Mbyte disk drives, running the TOPS-10 operating system [13]. This machine has been the main computational resource of the CMU-CSD for the past five years. Towards the end of this period, a number of other machines were added to this environment. Though the machine used for this study is now off-loaded by those machines, it continues to play a very important role and is still heavily used. Consequently, it is expected that the data presented here is a good reflection of the file usage characteristics of this community.

2.2. The File System

In the TOPS-10 operating system, every file has a 6-character *file name* and a 3-character *file extension*, and is a member of exactly one *directory*. The file extension indicates the nature of the contents of a file. For example, a Pascal program source would have the extension PAS, while its relocatable object module would have the extension REL. An installation-dependent number of extensions are regarded as "standard" extensions. Though system and user programs often make assumptions about a file based on its extension, there is no mechanism for validating or guaranteeing these assumptions. In practice, it is extremely rare that a standard extension is used for non-standard purposes. A quarter of the files examined had non-standard extensions; such files were ignored for those parts of this study that discriminated on the basis of file type. File names, unlike extensions, have no system-wide significance and were not examined.

A file consists of a sequence of fixed-length *blocks*, which are the units of addressability on the disks. Each block consists of 128 36-bit words. The last block in a file may be only partially written; such blocks were regarded, in this study, as whole blocks. The size of a file is limited only by the amount of secondary storage available. Unlike some file systems, such as OS/VS2 for the IBM 370 [5], a user does not have to estimate the size of a file at the time of its creation.

The operating system maintains, for each file, information regarding its size, its owner, the date it was last written, the date it was last accessed and its physical storage map. This information may be obtained by queries from user programs to the operating system.

In the environment in which this study was done, a manual file migration scheme is used to relieve the paucity of disk space. Every month, the operations staff runs a program which copies onto magnetic tape, and deletes from disks, those files which have neither been written

nor read in the preceeding three months. Files so migrated may be restored to disk at the request of their owners; in practice, very few such requests are received. Each user has a file named MIGRAT.DIR to which the migration program appends details of every file of that user it migrates. The union of a user's current directory and his MIGRAT.DIR entries constitutes the set of all files created, but not deleted, by that user. Users who wish to edit their MIGRAT.DIR files may do so; in practice this is quite rare. Files may also be migrated at the explicit request of users; it is observed that few files are migrated for this reason.

2.3. The Collection Technique

The files in this study fall into two classes: *current files* and *migrated files*. Data for both classes were obtained without any modifications to the operating system. A vendor-supplied utility program which creates a file containing details of every other file in the system was used to obtain data on current files. Data on migrated files was obtained by examining the MIGRAT.DIR file of every user in the system. For both classes the data extracted was organized as a 3-dimensional array with logarithmic age histogram buckets on one dimension, logarithmic size histogram buckets on another dimension, and the set of standard file extensions on the third. This array was created once each for current and migrated files, recorded in a file, and used as a database for software written to answer questions such as "What is the distribution of file sizes for current files with ages in *a given range* and with *a given set of extensions*." Table 6-1 shows an example of the output for one such query.

It should be noted that the data gathered by this method is a snapshot of the file system at one point in time. To examine the temporal behavior of the file properties described here, one would have to take snapshots spaced apart in time and compare the data from each.

2.4. The Quantities Measured

Probably the three most common questions asked about any file are:

1. "What does it contain?"
2. "How big is it?"
3. "How old is it?"

To the designer of a file system, the first question is probably only of marginal relevance. In any case, a precise answer to it requires a complete specification of the contents of a file! Specifying the extension of a file answers this question at one level of granularity. One outcome of this study is, therefore, a histogram of file extensions for any cross-section of the set of files examined. Figure 6-1 shows such a histogram. The integers on the abscissa are mappings from the set of extensions to integers; Table 6-2 gives some of these mappings.

The size distribution of files is a crucial factor in deciding many of the file system parameters. The size of a file, measured in blocks, is one of the two quantities of primary interest in this study.

The other important quantity is the age of a file. "Age" is usually understood to mean the interval between the creation of a file and the instant of data collection. However, the original date of creation of a file is not maintained by TOPS-10; only the dates of last modification and last access are available. The difference between these two dates is a measure of the usefulness of the current data in the file. This quantity, the *functional lifetime* of a file, is the second item of interest in this study. For brevity, the term "f-lifetime" will mean "functional lifetime" in the rest of this paper. Fortuitously, it is the f-lifetime of a file, not its chronological age, which is important in the design of file migration algorithms. Further, the file system design described in [1] and referred to in Section 1 is predicated on the assumption that the f-lifetime of files is short — this study was conducted, in part, to verify this assumption.

3. Data Interpretation

3.1. General Observations

A total of about 36,000 current files and 50,000 migrated files were examined in this study. About 99% of the files examined had sizes less than 1000 blocks and f-lifetimes less than 2000 days.[1] Both size and f-lifetime are discrete variables, with minimum values of 1 block and 1 day respectively. However, for ease of data interpretation and analytical approximation, both variables are treated as continuous variables.

Even a cursory examination of the data reveals some interesting facts. As Figure 6-2 indicates, the size distribution is skewed towards small sizes: 50% of the files are less than 5 blocks long and 95% of them are less than 100 blocks long. Figure 6-3 shows that the f-lifetime distribution is also skewed towards the low end, though not as sharply as the size distribution. Nearly 30% of the files have f-lifetimes of one day and 50% of them have f-lifetimes less than 30 days. Tables 3-1 and 3-2 present a subset of the data points used to generate Figures 6-2 and 6-3.

[1] There are some files which are older than the system — they were obtained from other, older, PDP-10 systems.

Size	Cum. Fraction
1 block	0.245
5 blocks	0.518
10 blocks	0.665
100 blocks	0.952
1000 blocks	1.000

Table 3-1: Cum. Dist. Fn. of File Sizes

F-Lifetime	Cum. Fraction
1 day	0.320
10 days	0.410
100 days	0.651
1000 days	0.947
2000 days	0.986

Table 3-2: Cum. Dist. Fn. of File F-Lifetimes

The preponderance of very small files indicates that the TOPS-10 file allocation algorithm, which allocates disk storage in units of 5 blocks called *clusters*, tends to waste a significant amount of storage. However, the size of allocation tables increases as the unit of allocation decreases. Further, sequential access to a file is likely to be faster if successive blocks of the file are close to each other; this is more likely with larger allocation units. The TOPS-10 choice of 5 blocks as the unit of allocation probably represents a reasonable tradeoff between minimizing storage fragmentation and improving performance. Powell [8] discusses these tradeoffs and the use of adaptive disk allocation strategies in the context of the Demos file system. The minimum unit of allocation in that system is one disk sector, 4K bytes, which is of the same order of magnitude as the unit of allocation, 2.5K bytes, in TOPS-10.

The rest of the data analysis discusses three questions:

1. Are the properties of migrated files different from those of current ones?
2. Does the type of a file affect its properties?
3. Does the size of a file influence its f-lifetime?

3.2. Effect of Migration

Figure 6-4 compares the size distributions of current and migrated files. Except at the very low end, there is virtually no difference between the curves. At the low end, there are fewer migrated files than current files. The following explanation may explain this phenomenon: a large number of very short files are created by system programs. Text editors and mail servers are two examples of programs which create short auxiliary files which are used only once. These files are automatically deleted by the programs which created them when they are run a second time, or by users when they run out of disk quotas. Such files are unlikely to remain both unaltered and undeleted for a period of time long enough to qualify them for migration. Consequently, small files are likely to form a smaller fraction of the migrated population than the current population.

Figure 6-5 shows that migrated files tend to have shorter f-lifetimes than current files. To see why this is so, consider how a long-f-lifetime file gets migrated. It would have to get created, then read (but not written) frequently for a long time and then all accesses to it would have to stop for a period long enough for it to qualify for migration. The only obvious files that meet these criteria are the successive versions of commonly used system or user programs. The infrequency of generation of such files leads to the fact that there are fewer long-f-lifetime files in the migrated population than in the current population.

The rest of this paper discusses only current files. Unless otherwise specified, the comments about current files also hold for migrated files with, perhaps, slightly different absolute numbers.

3.3. Effect of File Type

Since it is located in a research-oriented, academic environment, the machine on which this study was conducted is used primarily for two activities: document preparation and program development. Nearly half the files examined were created in conjunction with one of these two activities: program sources files, program object files, document processor input files, and document processor output files. This section examines the characteristics of these four classes. The remaining half of the files was highly fragmented, with no clearly identifiable, large classes. Detailed study, discriminating on the basis of file type, of that set of files is unlikely to yield any fresh insights.

Figure 6-6 shows the effect of file type on file size. Object files and document processor output files tend to have much larger sizes than source files and document processor input files. The size characteristics of the entire population resembles that of source and document processor input files.

Figure 6-7 shows the effect of file type on file f-lifetimes. Document processor files tend to have much shorter f-lifetimes than program files. It is possible that this is due to the fact that once a document is complete, people tend to read the hard copy rather than the machine-readable copy. Important programs, on the other hand, tend to be used many times after they are debugged. Certain program source files are read long after they are debugged; for example, useful macro definitions are often included in other programs.

Table 3-3 summarizes the important characteristics of different file types. Probably the most important lesson to be learned in this section

is that the type of activities engaged in by a user community strongly influences the size and f-lifetime properties of the files created by it. Files in a commercial data processing environment or a fusion research center can be expected to exhibit markedly different characteristics from those reported here.

Type of File	Number	Size		F-Lifetime	
		Mean	Std Dev	Mean	Std Dev
Program Sources	4010	21.84	47.63	363.6	731.3
Object Files	3474	53.99	116.3	414.6	681.4
Doc. Proc. Input	7085	29.28	70.95	137.5	322.7
Doc. Proc. Output	872	61.6	111.04	45.2	207.9
Entire Population	35652	23.89	66.83	238.9	531.9

Table 3-3: Effect of File Type on File Sizes and F-Lifetimes

3.4. Size/F-Lifetime Correlation

How does the size of a file affect its f-lifetime? Since the environment contains no large, frequently-modified databases, the most likely type of large files are infrequently-modified databases, or frequently-used and rarely-modified system programs such as compilers and editors. Small files, on the other hand, are likely to be temporary files of various sorts, or files associated with use-once-and-throw-away programs. Intuitively, therefore, one would expect large files to exhibit longer f-lifetimes than small files.

Figure 6-8 shows the f-lifetime distribution of files, with size as a parameter[2]. Suprisingly, the curves indicate that large files tend to have shorter, not longer, f-lifetimes than small files. The largest average f-lifetime is, in fact, that of 1-block files! Table 3-4 summarizes the information in Figure 6-8.

Size	Number	F-Lifetime	
		Mean	Std Dev
1 block	8745	264.8	633.1
10 blocks	762	231.4	512.8
91-100 blocks	207	170.5	308.8
401-500 blocks	101	123.1	344.5
901-1000 blocks	13	120.2	240.6

Table 3-4: Effect of Size on F-Lifetime

The data was closely examined to see if any particular file type, with markedly different characteristics from the total population, was causing this anomaly. Perhaps document processor output files, whose f-lifetimes tend to be short, perturb the data.[3] However, the data fails to support this hypothesis — document processor output files tend to have short f-lifetimes independent of size, and they do not form more than 9% of the total population larger than 100 blocks. Removing document processor output files from the total population does not change the size/f-lifetime correlation. One file type, with extension MSG, exhibits an average size nearly four times and an average f-lifetime one-tenth that of the total population. Files with this extension are used as repositories of electronic mail messages which have been received and read by users, but not yet deleted by them. Table 3-5 presents the size/f-lifetime correlation for MSG files. However, excluding such files does not change the size/f-lifetime behaviour. In fact, no single file type, constituting 5% or more of the total population, is responsible for this characteristic of files. For example, Table 3-6 shows the effect of excluding MSG files and document processor output files — there is still a falloff of f-lifetime with size. One can therefore take this to be a characteristic of files in general. If one were to assume that every bit of information stored in a file system is equally likely to be modified, the probability of a file being modified is directly proportional to the size of the file. Under this assumption, large files are indeed likely to have shorter f-lifetimes than small files.

Size	Number	F-Lifetime	
		Mean	Std Dev
1 block	7	414.0	581.0
10 blocks	16	50.9	115.0
91-100 blocks	13	25.8	71.5
401-500 blocks	11	29.5	76.5
901-1000 blocks	3	1.0	0.0

Table 3-5: Size/F-Lifetime Correlation for MSG Files

Size	Number	F-Lifetime	
		Mean	Std Dev
1 block	8725	266.5	640.3
10 blocks	717	243.0	528.2
91-100 blocks	182	189.3	322.7
401-500 blocks	80	142.4	375.1
901-1000 blocks	9	173.1	272.1

Table 3-6: Effect of excluding MSG and Doc. Proc. Output Files

4. Analytic Approximation

4.1. General Discussion

Analytic approximations to the size and f-lifetime distributions are investigated here for two reasons:

- To obtain a simple and computationally efficient means of generating random size and f-lifetime variables.
- To see if a model useful in analytic performance evaluations can be postulated for file sizes and f-lifetimes.

[2] There are too few files of size 500 blocks or more to obtain a smooth cumulative distribution function; a discrete function is therefore shown for such files.

[3] This suggestion was offered by one of the reviewers of this paper.

The models examined in this paper were motivated by the following observations:

- At least to a first approximation, the size and f-lifetime of a file one creates is independent of the files one has created in the past.
- Both size and f-lifetime distributions are sharply skewed toward the low end.
- A Markovian model is analytically the most tractable [2, 3].

The simplest model meeting these criteria is an exponential distribution. If the size distribution is exponential with mean M, the probability that a random file has a size less than X is given by $1 - e^{-X/M}$. Both the mean and standard deviation of such a distribution are equal to M. Unfortunately, almost all the size and f-lifetime distributions observed have standard deviations between two and three times that of the corresponding means. This implies that a simple exponential model is certain to be unsuitable.

A hyperexponential model can exhibit coefficients of variation (i.e., ratio of standard deviation to mean) greater than unity. A k-stage hyperexponential consists of k simple exponentials with means M_1, M_2, ..., M_k, weighted so that they have probabilities α_1, α_2,.....,α_k of being chosen. Figure 4-1 shows such a model.

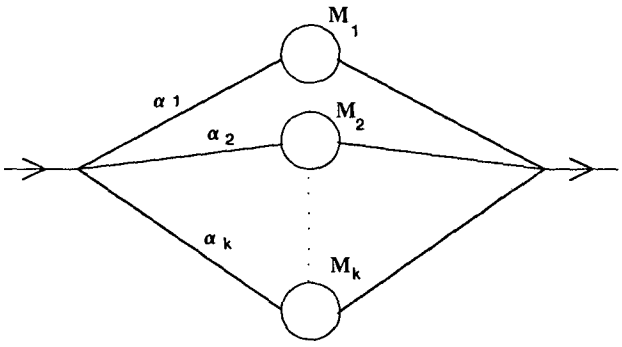

Figure 4-1: A k-Stage Hyperexponential Server

To generate a value for the random variable represented by this model, one proceeds in two steps:

1. With probability α_i, select one of the k stages.
2. Generate one value from an exponential distribution of mean M_i.

Each of the k stages can be viewed as being one population class with a simple exponential distribution. There is no guarantee, however, that such classes correspond to any clearly identifiable types of files. To fit a hyperexponential model to empirical data one needs to determine the number of stages, k, the means M_i, and the probabilities α_i. The next two sections discuss two alternative approaches for estimating these parameters.

4.2. The Moment Matching Method

In this method we try to find a hyperexponential model whose first few moments match the corresponding moments of the empirical distribution. If the empirical distribution were truly hyperexponential, one could find a model with all its moments matching. Otherwise the model is only an approximation to the empirical distribution.

The p^{th} moment of a k-stage hyperexponential is related to its parameters (α's and M's) by the following relationship:

$$\alpha_1 M_1^p + \alpha_2 M_2^p \ldots\ldots \alpha_k M_k^p = (p^{th} \text{ moment})/p!$$

This is easily derived using the moment generating function technique [3]. By using an iterative solution technique on 2k-1 such equations and the constraint $\alpha_1 + \alpha_2 + \ldots + \alpha_k = 1$, one can solve for the 2k unknowns, α_1 to α_k and M_1 to M_k.

Figure 6-9 compares the empirical size distribution of current files with a 2-stage hyperexponential fit.[4] The first three moments of these two curves are identical. The two curves differ by no more than 5% at all points except at the very low end. Figure 6-10 shows the distribution of f-lifetimes of current files versus a 2-stage hyperexponential fit. Clearly the fit is not as good as for file sizes, especially at the low end, where the hyperexponential grossly underestimates the empirical distribution.

Adding more stages to the hyperexponential, thereby matching more moments, yielded negligible improvements in the fit (less than 0.5% except at the very low end, where the improvement was close to 1%.) The moment matching technique is thus only of limited usefulness in analytically approximating the empirical data presented here.

Fig. No.	α_i	M_i
6.9	0.089 0.91	164.9 10.08
6.10	0.835 0.165	77.49 990.8
6.11	0.518 0.433 0.048	2.0 23.83 252.1
6.12	0.407 0.337 0.256	1.66 70.0 795.0
6.13	0.32 0.132 0.385 0.162	1 (const) 7.97 132.1 1082.8

Table 4-1: Derived Parameters for Fitted Curves

[4] Table 4-1 presents the derived parameters α_i and M_i for all the fitted curves discussed in this section and the next.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.