Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

Physics, Computer Science & Mathematics Division

FILE MIGRATION IN DISTRIBUTED COMPUTER SYSTEMS

Juan M. Porcar
(Ph.D. thesis)

July 1982

Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

LBL-14763
Progres-82.6

File Migration in Distributed Computer Systems

Juan M. Porcar

Physics, Computer Science and Mathematics Division
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

Ph.D. Thesis

July 1982

This research was supported by the Applied Mathematical Sciences Research Program, Office of Energy Research of the U.S. Department of Energy under Contract DE-AC03-76SF00098; and DE-AC03-76SF00515 (SLAC), and by the National Science Foundation grants Nos. MC575-06768 and MCS77-28429.

ACKNOWLEDGMENTS

I wish to thank my advisor, Professor Alan J. Smith, for making this dissertation possible by providing the topic, data to conduct the research, financial support, and staunch direction for almost five years.

My gratitude goes to Professor Domenico Ferrari for his example as a researcher and as a teacher. Without his encouragement and his suggestions, this thesis might have never been completed.

I want to thank Professor Ronald Wolff for serving on my thesis committee.

I am also indebted to the following persons and organizations:

Two generations of members of the PROGRES research group at Berkeley. Their comradeship will never be forgotten. Particular thanks go to my friends Özalp Babaoğlu and Frank Olken for being always ready to listen and to share their knowledge.

Carl Quong and his staff of the Computer Science and Applied Mathematics Department at the Lawrence Berkeley Laboratory for being exceedingly cooperative and for providing the unbounded computer resources required by my research and my mistakes.

The "Comisión de Intercambio Cultural entre España y los Estados Unidos de América" for a Fullbright Foundation Grant to come to Berkeley in the first place.

The National Science Foundation under grants MCS75-06768 and MCS77-28429 and the Department of Energy under Contracts DE-AC03-76SF00098 (to the

Lawrence Berkeley Laboratory) and DE-AC03-76SF00515 (to the Stanford Linear Accelerator Center) for financial support.

Finally my parents and Michelle, for their love and their encouragement during the research and the writing of this dissertation.

TABLE OF CONTENTS

CHAPTER 1

COMPUTER NETWORK CONCEPTS

1.1. Introduction

In the last decade the areas of computing and communications have been converging rapidly. In the communications industry, computers are used extensively for switching, routing, maintenance and a number of so called enhanced services, like message storing and forwarding. At the same time, communication devices have become an essential component of computer systems. Many computers are linked to remote terminals and to other computers. This allows a collection of machines to work together in the solution of a single problem, or to share a database. A collection of autonomous but interconnected computers is called a *Computer Network* [Tan81]. When there is a great degree of cohesiveness and transparency in the operation of the computer network, it is known as a *distributed computer system* or a *distributed system* for short.

In the past several years there has been an increasing interest in the design and the use of distributed systems. We will elaborate briefly on three of the reasons behind this rise in popularity: the need for sharing unique resources, the favorable price/performance ratio of small computers compared to large mainframes, and the changes in the relative pricing of computing and communications.

Computer networks first appeared in organizations that were operating many independent computers, often located far apart. By connecting the machines together, all resources can be made available to all users, irrespective of their physical location. These computer networks, also known as resource-sharing networks, allow users to share expensive hardware, unique software and valuable data. They can also provide better service through load leveling and backup of failed machines.

Large geographically distributed networks have served as testing grounds for new hardware and software technologies upon which more recent distributed systems are based. These technologies include store-and-forward packet switching [Dav79, Tob78], adaptive routing [McQ74] and layered protocols [McQ78].

The best known among the resource-sharing networks is the ARPANET [McQ77], which connects over one hundred computers belonging to universities and defense contractors, spanning half the globe, from Hawaii to Norway. The Arpanet is a unique system but there exist some commercial networks that offer the same type of services. For example, computer manufacturers offer networking products under the form of *Network Architectures*,. A network architecture is a set of hardware and software modules that allow users to turn their geographically dispersed computing facilities into an interconnected network. Two examples of these products are IBM's SNA [Gre80, Hob80] and DEC's DECNET [Wec80] An even more general networking service is provided by the *Public Networks* like Telnet and Tymnet in the U.S. or Datapac in Canada. Almost any machine can be connected to one of these networks by means of a special adapter. Internationally agreed upon protocols, like X.25 [Fol80] permit the transfer of information between machines and between networks [Bog, Uns81].

The second reason that we gave for the recent development of distributed computer systems was the superior price/performance ratio of mini and microcomputers over large mainframes. Networks of small computers are replacing large mainframes in applications where no single task requires the processing power of a large machine. In this type of distributed system, all the machines are usually within the limits of a room, a building, a manufacturing complex or a university campus. Hence the name of *Local Area Network* (LAN) [Thu79].

Besides lower total cost, Local Area Networks provide incremental growth. When a local area network requires more processing power, additional processors can be added without the need for a total system change. This also means that the average system cost over a period of time is lower because of the smaller initial investment.

Distributed systems in general can match the decentralized structure of an organization better than a central data processing facility. Several divisions in a company may want to use their computers for different purposes while still being able to access some global database. To this end, they can optimize their machines by purchasing different processor models, different peripherals or different software packages. This usually results in managers feeling in control of their computing facilities and having a better attitude towards data processing.

The third and last reason for the widespread use of distributed systems is the relative pricing of computing and communications equipment. In the nineteen sixties, information processing was relatively expensive and systems that gathered data from wide geographical areas tended to transmit raw data to a central computing facility, where it was processed, and maybe stored or sent back to the point of origin. More recently, the pricing structure has changed dramatically and communications costs tend to be higher than processing costs. Therefore, it makes sense to use as much local processing power as possible to reduce the volume of data transmitted.

Many distributed computer systems are now in operation whose primary design goal is to reduce the transmission costs of an application. One such example is the on-line inventory control and customer invoicing of Lowes Companies, Inc [Cha77]. This particular system has one small computer with disk storage and up to sixteen terminals at each one of the chain's 140 stores. Most information about store inventory, customers and prices is kept locally and updated once a day from the central corporate computer center. Distributed systems of this same kind can be found in banking, manufacturing, airline seats reservations, etc.

Performance is an important factor in the design of distributed systems. Most performance evaluation studies of distributed systems concentrate on the communications subsystem. Popular topics are the measurement and modeling of various topologies [Sho80, Bux81, Mar81], performance comparisons for different protocols [Hay81, Art81a], routing algorithms [Gop81] and flow control for local and geographically distributed networks [Lam81a, Won78, Lam81b, Ten81, Raz80]. While work in these areas is necessary in order to solve unavoidable engineering problems, there is a more fundamental problem in the performance of distributed computer systems: the partition of the workload and the assignment of its components to the elements of the distributed system. In general terms, the good performance of a distributed system depends on the existence of both parallelism and locality in the application for which it is used. The parallelism is necessary in order to process as many tasks in parallel as possible. On the other hand, there must be a good amount of locality to keep the communications requirements moderate. In a general purpose system, good locality is achieved by a sensible assignment of files to nodes. The goal is to keep the files as close to their users as possible.

In this dissertation we study some of the performance issues related to the transmission of information in distributed computer systems. More specifically, we are seeking algorithms for placing and migrating information in distributed systems, in order to minimize both the volume of data being transmitted along communication links and the delay experienced by the users. In the remainder of the introduction, we will discuss this in more detail, both reviewing past work in the area and presenting our approach to the problem.

1.2. The File Assignment Problem

The distribution of files among the storage elements of a distributed system may have a serious impact on its overall performance. A distribution of files that results in many remote access to files is likely to result in poor system performance. Remote accesses increase the load on the transmission susbsystem, thus delaying other sources of communication, such as terminal traffic. Remote access is also slower than local access because the network delay gets added to the disk access delays. Finally, remote accesses increase the load on the processor and memory in the form of operating system overhead. This overhead comes from the handling of protocols, creation of packets, maintenance of buffer pool space, checksum computation and error handling. In conclusion, any application will perform much better if all its files reside in the local system than if it needs to access remote files. The File Assignment Problem (FAP) is concerned with optimizing performance in the context of where to put the files in a distributed system.

It is not too difficult to formulate the FAP as a general optimization problem by using a simplified model of the workload. This has been done in the past and section 1.3 reviews this approach to the problem. However, it is very difficult to obtain a solution to this problem for any reasonable system size and number of files. Implementing this on a real distributed computer system would be even costlier because the optimization algorithm would have to be run rather frequently to keep the assignment optimal. We will narrow down the scope of our research space, in order to obtain less general but more useful solutions to the FAP.

In the next subsections, we will characterize the type of systems that we are interested in (system model), the type of cost functions that we consider and the type of solution that we are seeking.

1.2.1. System Model

For the purposes of our research, the system model has three relevant components: the computers at the nodes, the way in which the information is stored and the type of network that connects the nodes.

We only consider general purpose computers, with enough processing power and storage capacity to be able to run all their programs locally. This eliminates systems that are used in real-time applications or personal computers that do not have any disk capacity.

We also require that the information be stored in a relatively large number of independent files. This rules out the database management systems, which usually keep most of their data in a single logical entity: the database. One of the reasons for choosing this type of system as the object of our research has to do with our methodology. We will use trace-driven simulations to evaluate our policies and the traces that we have were obtained from general purpose systems used for program development and production runs of scientific and business-oriented programs. It is certainly not obvious (a priori) that the results based on these measurements can be applied to drastically different systems like transaction-based or database management systems.

Our model for the network is a fully connected network with equal communication costs on all the links. In essence, we eliminate all the topology and routing considerations from the problem. This model of the communications subnetwork does not match some real systems, like the ARPANET, where the degree of connectivity is much lower and where the connections have widely varying bandwidth and delay characteristics. However, this is not to say that our results will not extend to most computer networks. In the first place, most local area networks do have fixed routing. This includes ethernet-like networks [Met76, Eth81], rings [Wil80], star-shaped networks [Lud81] and their derivatives. But even for networks with arbitrary topology, there are good reasons for not modeling too closely the structure of the communications subnetwork. The main reason has to do with the layered structure of the protocols that are used in most networks. In the framework of the Reference Model of the *Open Systems Interconnection* [Zim80] for example, all the decisions about routing and delay optimization are done at the *Network Layer* (layer 3 in the OSI model). File migration, on the other hand, should be implemented at the *Presentation Layer* (layer 6) or at the *Application Layer* (layer 7), as a file system service. Since all layers above the Network Layer regard the network as a fully connected grid, and the information on which the optimal routing is done is not available above that layer, it would be very difficult to make decisions at the file migration level that could influence (positively) the traffic conditions in the subnetwork.

1.2.2. Cost Functions

We will consider three types of cost in the operation of distributed systems: storage costs, traffic costs and delay costs.

1.2.2.1. Storage Costs

Storage costs are incurred when information storage devices are bought or leased. The price structure of storage is very dependent on current technology. In particular, there are large economies of scale to be made when buying disk storage (the predominant type of on-line storage in today's systems). Furthermore, disk storage can only be purchased in discrete amounts, adding to the complexity of the price as a function of the total capacity of the system. We will adopt a much simplified price structure where storage costs are proportional to the amount of information stored in a given node. In some cases we will make the extra assumption the the unit cost is the same at all the nodes.

1.2.2.2. Transmission Costs

Transmission costs are incurred when some information is actually sent over communication lines. The best way of thinking about these costs is to assume that the communication between machines is through a public data network. In a public data network (like Telenet or Datapac) one basically pays for the number of packets that are actually sent. Under those conditions, and assuming that packets are utilized efficiently, the transmission costs are just proportional to the amount of traffic generated. We will not consider the problem of packet fragmentation because, in a situation where most of the transfers are large file records or entire files, the utilization of the packets should be of no concern. Trying to model the cost of communications over leased lines or switched circuits is much more difficult because one is paying for the circuit even when it is idle, and the circuit's utilization, in turn, depends on the dynamic characteristics of the traffic.

When measuring traffic in a communications network, it may be convenient to make the distinction between data traffic and control traffic. Data traffic comes from the transmission of useful information. Data traffic is in many ways independent of the details of the communication subsystem and therefore easy to measure. Control traffic can be considered as overhead traffic and it includes acknowledgments, directory inquiries, routing information and so forth. Control traffic is more dependent on the particular implementation of the system. Nevertheless, control overhead is largely proportional to the number of IOs and it may be estimated by measuring this number.

1.2.2.3. Delay Costs

Delay costs are due to the cost of the resources that are allocated to a task and held unproductive while the task is waiting for the completion of a transmission. User time is an example of delay cost. Delays occur because of physical limitations in the transmission of information. The transmission rate measures the amount of data that can be transmitted through a communication channel per unit of time with a nominal probability of error. Given this physical limitation, a data transfer is subject to two sorts of delay: the delay that the transfer would experience if the entire bandwidth of the channel were assigned to it, and the delay due to the fact that it shares the channel with other transfers. In this research, we are interested primarily in the first type of delay. This delay consists of the transmission time and of some unavoidable propagation time in the network (including that consumed by the network interfaces). The transmission time is proportional to the amount of information transmitted. The propagation delay is a characteristic of the circuit and hence is a constant, β, for all transmissions between two points of a network. Therefore, the delay Δ involved in sending x bytes over a network is of the form:

$$\Delta = \alpha . x + \beta$$

1.2.3. Type of Solution

The File Assignment Problem is usually formulated as a constrained optimization problem. Total storage at each node, maximum response time and line bandwidth are given, and an optimal assignment is found. Our treatment, on the other hand, is to minimize the overall traffic and delay and to determine what storage capacity and what line bandwidths are necessary. Eliminating storage and bandwidth constraints from the statement of the problem allows us to consider files independently of each other. This in turn means that a global optimal performance for the system can be achieved by optimizing the operation of each file in the system.

We are interested in solutions to the File Assignment Problem that can be implemented in a distributed file system. In order to be implementable, policies may only rely on past information about the files and the system. Under these conditions, a policy is a mapping from the present state of the file (including some of the file history) into a set of possible actions (migrate the file, generate a new copy, destroy a copy, etc).

Finally, we prefer decentralized policies, that do not require a central database. Decentralized policies can be easier to implement, are more robust in the face of partial failures and may even eliminate some of the traffic.

After this introduction to the File Assignment Problem, we present related work that is relevant to this research.

1.3. Related Work

The File Assignment Problem has received substantial attention in the literature. A large fraction of the work on this topic finds its inspiration in the Plant Location Problem. The Plant Location Problem is well documented in the Operations Research literature [Coo63, Coo64, Alc76]. Given a set of consuming locations spread over a network of distribution channels, the Plant Location Problem deals with the assignment of manufacturing plants to nodes of the network in order to minimize the distribution costs of the goods from factories to consumers. The analogy with the File Assignment Problem is clear when plants are substituted by storage elements and consumers by processes accessing the files.

We now review some work related to our research in the following areas:

(1) Static assignment of files.

(2) Dynamic assignment of files.

(3) Hierarchical file system management.

1.3.1. Static Assignment of Files

The topic of static assignment of files to the nodes of a computer network is by far the most popular among the papers dealing with the File Assignment Problem. A static assignment is a time-invariant mapping between nodes of a computer network and the files of the system. The assignment determines how many copies of each file should exists in the system and in what nodes should the copies be stored. All the papers that we are aware of present the problem as a 0-1 integer program [Cas72, Cas73, Chu76, Hol73, Lun77, Lun78, Mah76, Mor77, Ram79, Wah79]. The unknown variables (one for each file-node pair) indicate whether a given file has a copy assigned at a given node. Most papers make the following assumptions in writing the operating equations.

(1) The amount of information transmitted from any file to any node per unit of time is a known quantity. This information is transmitted at a constant average rate.

(2) Queries are performed from the closest copy of the file. Updates are sent to all the copies of the file.

The cost function is, in most cases, a sum of storage and traffic costs. A few papers consider delay costs. In this type of formulation it is very easy to add constraints to the problem. Most statements of the problem consider constraints in the storage available at each node, in the bandwidth of the transmission lines and on the average response time of queries.

A paper by Eswaran [Esw74] shows that the File Assignment Problem, formulated as a 0-1 integer program, is an NP-problem in the general case. It is hence very unlikely that an efficient algorithm can be found to solve the problem. However, quite some effort has gone in trying to find good heuristics to solve this type of 0-1 integer problem. Two approaches have been taken: purely theoretical and experimental. Among the theoretical papers, there are studies of particular issues, like the maximum number of copies that the optimal solution can have [Bel76, Gra77] or the form of the optimal solution for networks that have an easy topology [Whi70].

The more experimental papers use standard solution methods for integer programs, like branch and bound, with heuristics to limit the solution space. Most of these studies report successful attempts to apply these heuristics to small problems. In these small case studies, the times needed to find optimal solutions are similar to those needed to solve linear programs. However, it must be pointed out that the size of the examples is exceedingly small, various orders of magnitude smaller than the size of real systems [Cha76, Tri80, Fis80].

There are three major problems with the treatment given to the FAP by the papers that we have mentioned so far:

(1) The workload model is too simple. Except for very specialized systems, modeling the access to files by a constant rate over long periods of time is an oversimplification and does not correspond to the observed behavior of file systems in general purpose computers.

(2) There is no validation work based on measurements or simulation.

(3) The solution is difficult to implement beyond the design stage. In order to use this approach during system operation, a central location should collect all the information regarding rates of access and run the optimization. The optimization is basically done using mean values for the parameters. Unless the workload is very stable, it is unlikely that the resulting optimal assignment will perform very well.

The work that we describe in the next subsection addresses some of these criticisms.

1.3.2. Dynamic Assignment of Files

A few papers in the literature deal with the dynamic allocation of files in distributed systems. One approach that has been suggested is to solve repetitively one of the static formulations [Lev78, Ros73, Ros75]. This takes care of our objection about the stationarity of the access rates, but makes the problem even more difficult from the computational point of view. Furthermore, none of the authors has looked into the problem of the adaptive estimation of the access rates.

A more promising approach has been proposed in the automatic control field [Seg76, Seg79]. The main tool here is dynamic programming, both deterministic and stochastic. The deterministic dynamic program provides the true dynamic optimum once the period of the experiment is divided in a number of periods where the access rates remain constant. The complexity of the problem grows with the number of such intervals, so the dynamic solution is obtained at a high price. The stochastic version is also treated and solved in a rather elegant manner. Our problem with this result, again, is the model of file reference. The access rate is modeled as a continuous random variable with a fixed number of different values that vary according to a Markov model. While this makes the model solvable, it is not the behavior that we have observed in real systems. In particular, users are supposed to access files with time-varying rates that are independent of each other. What we have observed, rather, is that there is a strong structure in the order in which users access a given file and this characteristic of the file referencing process is important when designing migration algorithms.

There are a number of papers which find closed form expressions for the optimal number of copies under some rather restrictive conditions [Cof80a, Gra77, Cof80b, Cof81]. These papers come from the database management field and their optimality criterion involves both the traffic considerations and the reliability of the system.

Finally, we note that some of the papers on the subject of task assignment in distributed systems have used solution methods that could be used in the dynamic assignment of files. In particular, a series of papers [Bok79, Rao79, Lee77] use methods of Network Flow Theory [For62] to minimize the traffic originated by a collection of tasks executing in a distributed computer system. In this formulation, the processors are represented by the nodes of the network and the amount of information exchanged by the tasks is represented by the flow in the links of the network. The minimization of the flow produces an assignment of tasks to nodes that minimizes the traffic in the communications network.

The same approach can be adapted to the file placement problem by assigning as flows on the network the traffic induced by the migration of files. Solving the minimal flow problem then yields the optimal assignment of the files to the nodes of the computer network.

1.3.3. Hierarchical File Systems Management

Our research is very much influenced by a small number of studies on file migration between secondary and tertiary storage [Str77, Smi81a, Smi81b, Art81b]. These studies pay much more attention to the workload characteristics of the file system than any of the papers that we have described in the previous sections. Moreover, they provide some type of validation through the use of trace-driven simulation. The traces used were obtained during normal operation of large installations. All solutions are based on the existence of some sort of *Locality* [Cof73] in the reference process and on the fact that *Working Set* policies are quasi-optimal in the presence of locality.

The difference between hierarchical file system management and distributed file system management is that the latter has one more dimension: the position of the active copies. Active copies of the file can be at more than one location. In the hierarchical case, the files move between levels of a single system and there is only one active copy of the file, if any, at the top level. If the file is not referenced for a period of time, it starts migrating to lower levels until it ends up in an archival store. When the file is updated, only the top level copy is updated. Having stale copies at lower levels is not a problem because they are never accessed while there exists a copy at a higher level.

The distributed case has more degrees of freedom. In the first place, referencing the file does not imply that the file is transferred to the location of the process accessing it. It can be accessed remotely. Furthermore, there are many 'top levels', one at every node, and each of the top levels could have an active copy at a given time. Also, in the event of a file update, all the copies must be updated.

Especially relevant to our work is Smith's paper on Long Term File Migration [Smi81b] where he introduces the concept of policy as a mapping from the current state of the file to the value of the control parameter (there the working set window size). We use this concept in our description of migration algorithms.

1.4. Objectives and Contributions of this Research

The emphasis of this research is on the evaluation of policies for placing and migrating files in computer networks. We will restrict our attention to shared files; i.e. files that are used by more than one user in the system. Files accessed by only one user should be stored at the node from where the user normally accesses the system. A different strategy could be adopted if there were constraints in the amount of storage at certain nodes (or big differences in storage price), but we do not consider this aspect of the problem.

One of the objectives of the thesis is to characterize the process of referencing shared files. This process has four main aspects: the order in which users access the file, the interreference times, the frequency of updates compared to the total number of accesses, and the fraction of the file that is accessed when it is opened. We will model this referencing process by a semi-Markov process where the states are the nodes of the network.

The evaluation of the migration policies will be done, when possible, analytically, using the above mentioned model. Trace-driven simulations will be used both to validate the models and to evaluate the policies that are not tractable analytically. A brief outline of the thesis follows.

Chapter 2 consists of an exploratory analysis of the traces that are used in the rest of this work. The initial data analysis is needed to learn some basic facts about the traces (the number of shared files, the number of users, etc), and also to discover files or users that must be declared outliers. Examples of these outliers are files used by the system, like spooling files or logs.

Chapter 3 is devoted to the generation of synthetic distributed systems. This is necessary because the traces that we have were obtained from centralized systems. In this chapter we perform a partitioning of the user community and obtain distributed systems that have the same number of users and files, the same requirements of processing and I/O activity as the traced systems but where users are located in a number of imaginary nodes in a computer network. The procedure has some merit in itself because quite a few installations will be facing this problem as they move from a large mainframe system to a distributed environment.

Chapter 4 present the single-copy policies for placement and migration of files in distributed systems. We look at this type of policy in the first place because it is easier to model and to implement. As a matter of fact, most of the services provided by current networks are some sort of single-copy mechanism [Hui81, Hwa80].

Chapter 5 is devoted to the study of the more ambitious policies that maintain several copies of each file in the network. We will limit our attention to the policies that maintain only up-to-date copies of the files.

CHAPTER 2

EXPLORATORY DATA ANALYSIS

Summary

Successful optimization of computer systems requires the study of actual systems behavior. This chapter presents an exploratory analysis of traces of activity from two large computer installations (SLAC and Hughes Aircraft). Distributions of variables such as interreference times, number of users per file, file size, number of opens per file and fraction of file size accessed per open are presented.

In the systems that we analyze, shared files (files used by more than one user) are responsible for about 25% of I/O activity even though they only represent a very small fraction of the number of files. This suggests that it is important to optimize the management of shared files in distributed computer systems, where shared files generate unavoidable traffic on the communications network.

Introduction

In this chapter, we present an exploratory analysis of the data that we use in the rest of the thesis. The data consist of traces of computer systems activity. These traces were obtained from large computer installations during periods of normal operation.

We have two main goals in conducting this exploratory data analysis:

(1) Classify the files that appear in the trace. All files in a computer system cannot be treated as coming from a homogeneous population because

different classes of files serve different purposes. Temporary files, for example, are usually created to store information while a job is running. Users are often times unaware of the existence of these files and have no control over them. Permanent files, on the other hand, are used for long term storage and users are usually responsible for the space that they use up. It is normal that these two types of files show completely different types of behavior. The exploratory data analysis also allows one to detect files that are used in unusual ways and that should not be included in our studies. One example of this kind of outliers are the zero-sized files that are used as locks or time-stamps by some applications.

(2) Measure the system's activity. It is necessary to have measurements of the system's activity in order to devise methods to improve its performance. Literally thousands of measurements can be taken from the traces that we study. In this chapter we present the measurements that are related to the design of file placement and migration algorithms.

Section 2.1 describes how the traces were obtained and explains some of their characteristics. The following two sections are devoted to the study of two different installations. Section 2.2 presents the data from the Stanford Linear Accelerator Center. Section 2.3 repeats the analysis for the Hughes Aircraft system.

2.1. Obtaining the Activity Traces

The traces of system activity were generated in two steps. First, IBMs' System Management Facility (SMF) was used to obtain *raw* traces. Then, *reduced* traces were created by extensive processing of the raw traces. We will describe these two steps in the following subsections.

2.1.1. The System Management Facility (SMF)

The System Management Facility (SMF) [IBM78, IBM73] was primarily conceived as an accounting tool. As such it collects data on system performance and on the use of resources by jobs and job steps. It also collects data on creation and use of data sets (files). SMF organizes the data in records and writes these records onto files on disk or tape. It also backs up these data sets to archival storage. All these capabilities were used to generate the raw traces.

As it comes out of the system, the data is not very usable. In the first place, there are problems with the way the data is collected by SMF (see [Dur78] for details). In addition, the format of the traces is not well suited for statistical analysis and for use in trace-driven simulations. We will describe these problems while explaining how they have been solved in preparing the reduced traces.

2.1.2. The Generation of the Reduced Traces

Reduced traces bring improvements in at least five areas:

(1) The records as archived by SMF may be out of chronological order. Fortunately, each record contains a time stamp in microseconds. This is enough to reconstruct the original series of events. The reduced traces have all their records in chronological order. This facilitates sequential processing of the traces and it makes possible the use of traces as input to trace-driven simulations.

(2) The traces generated by SMF have a problem related to warm start. When tracing is turned on, some jobs are running, some files are open, and many files exist in the file system. This may be a nuisance in a

trace-driven simulation and the expedient solution of forgetting the first portion of the tape may cause further problems. The reduced traces have dummy records for the jobs that are running at the beginning of the trace. They also have dummy open records for the files that are open at the beginning of the trace. Dummy records are also provided if problems with the hardware or with the operating procedures produce inconsistent events.

(3) Naming problems are common in the SMF traces. Keeping track of renaming operations and of files with the same name on different volumes is more easily done once and for all during the generation of the reduced traces than every time the traces are used. Unique ID's (small integers starting at 0) have been assigned to all files, volumes, opened files, user accounts, jobs and job steps.

(4) SMF provides mostly "right parentheses" for the events that it records. For example, it produces a job record at the end of each job and a close file record after each use of a file. A trace in this form can be very difficult to use in trace-driven simulations. The reduced traces have "left parentheses" for all events and they provide in those all the information that is available about the event, even if that implies "future knowledge".

(5) Finally the reduced traces contain information about the size of files. This information is not directly available in the raw traces. Rather, it has to be computed from the amount of information that has been stored in the file since creation. The size information has been very useful in our studies.

We will not discuss here the format of the reduced traces. A paper by Richardson [Ric80] contains a full description of the record format and the operating procedures to generate reduced traces.

2.2. The SLAC Installation

The SLAC Computer Center is a large installation that serves a community of physicists and scientific programmers. Programmers develop their programs using Wylbur [Faj73], an interactive text editor and remote job entry system. They then submit their jobs to the batch input queue, through the ASP [Vin80] subsystem. The batch jobs interact with the file system through the usual interface of IBM's OS operating system. Job output is spooled by ASP and the users can look at their output files using again the Wylbur editor.

We have described the mechanism of job submission because it has an important relationship with SMF and with our traces. Wylbur runs as a single job (task) for all users. Since it never ends during the normal operation of the system, SMF does not report on the resources that it uses. In particular, the SMF traces do not contain any indication of the files used from Wylbur. In other words, our SLAC trace is exclusively a trace of the batch subsystem.

2.2.1. Basic Numbers

The trace spans a period of 13 days (310 hours), starting Saturday, the 29th of January, 1978, at 11:00pm. During this period of time, 552 users submitted 25,039 jobs. Of these users, 369 accessed at least one shared file. A shared file is a file that has been opened by more than one user during the span of the trace. The definition does not require that the file be opened by more than one user at the same point in time.

Table I. Basic Job Counts. Slac Trace.

Number of hours	310
Number of wk-days	10
Number of wkend-days	3
Number of jobs	25039
Number of jobs/wk-day	2198
Number of jobs/wkend-day	1017
Number of accounts	552
Accounts sharing files	369

Because SLAC has strong ties with Stanford University, activity on nights and weekends is usually high compared to other non-academic installations. For example, the average number of jobs processed during weekend days is about half the number of jobs processed during week days. Table I contains more information about jobs and accounts at SLAC.

During the 13 days that the system was traced, about 152,000 different files were accessed. This includes any file that was created, opened, closed or scratched during that period. Of these files, 142,000 are temporary files. Even though they represent more than ninety percent of the files accessed in the system, it must be remembered that they are mainly used to hold temporary information between job steps. Since they are not involved in the long term storage of data, we will not consider these files any further. Table II contains more information about them.

About 2000 files are sequential files on tape. These files are usually very large and too expensive to be stored permanently on disk. Because of their size, we will not include tape files in our studies. It must be noted that it is not practical to transmit very large files across networks with today's technology. It would take hours to transmit the information stored in a reel of magnetic tape across a typical ARPANET link, for example.

About 500 files are "system" files, including dummy libraries, SMF data sets, etc. Our reason for not considering these files any further has to do with the special ways in which they are used. On the one hand, many system files are heavily used in read-only mode. These include language processors, system libraries and the like. In the context of distributed computer systems, the best thing to do is to provide each node of the computer network with a copy of each of these files. Most of our algorithms would end up doing this anyway and we can save much effort by not including these files in our simulations. On the other hand, files like system logs, that are frequently written by many users must be regarded as a system peculiarity that would be implemented differently in a distributed system. For example, accounting files could be kept in independent files, one in each machine, and the accounting and billing programs would do an explicit merging of the files if necessary.

The remaining 6,310 files are permanent user files, stored on disk. These are the files that contain the long term information of the computer system. We have a special interest in the files that are used by more than one user account during the span of the trace. We call them *shared files* even if they are never used concurrently by more than one user. There are 495 shared files in the SLAC trace (shared files are a subset of the permanent files).

Some of the data in Table II are self explanatory, like the number of files. Other entries require further clarification. For example, the files listed as *existing initially* are files that did exist at the beginning of the tracing period and that were deleted during that period. For these files the SMF scratch record provided the creation date. Files that existed at the begin-

Table II. Basic File Counts. SLAC Trace.

	TAPE	TEMP	PERM	SHARED	SYS	TOTAL
Number of files	1994	142576	6310	466	1028	151908
Number of files initially	32	8	308	84	162	510
Number of files at end	1994	0	1955	386	675	4624
Number of read-write files	1461	-	5482	297	808	7751
Volume of files (MBytes)	55615	108974	2336	363	524	167449
Ave. files created/wk-day	145	12470	478	29	88	13182
Ave. files created/wkend-day	181	5958	509	59	49	6696
Ave. files scratched/wk-day	0	12473	373	7	33	12880
Ave. files scratched/wkend-day	0	5949	207	3	7	6163
Ave. opens/wk-day	441	21412	11735	3174	5261	38848
Ave. opens/wkend-day	317	10059	7230	1342	3335	20941
Ave. reads/wk-day	524687	800864	364676	235407	198777	1889004
Ave. reads/wkend-day	384978	616373	178193	79462	97407	1276951
Ave. writes/wk-day	557625	1295246	166629	52806	50491	2069992
Ave. writes/wkend-day	707264	730911	90212	11613	5918	1534305

ning of the trace and that were not scratched within the span of the trace appear as being created at the time when they are first referenced.

Another point that requires clarification is what we mean by *read-write* files. Files can be opened in one of five modes:

(1) CREATE: the file is opened in this mode when it is being created.

(2) INPUT: the file is opened in read-only mode.

(3) UPDATE: the (existing non-sequential) file can be read and/or modified.

(4) APPEND: the existing sequential file is extended with new information.

(5) OVERWRITE: the existing sequential file is erased and overwritten with new information.

By read-write files we mean files that have been opened at least once in either APPEND, OVERWRITE or UPDATE mode. Note that opening a file in UPDATE mode does not imply that the file is going to be modified.

The *volume of files* was computed as the sum of sizes in megabytes of all the observed files. Since many files, especially temporary files, are deleted during the span of the trace, the full reported volume of files was never present in the system. It is rather the amount of space requested from the system during the tracing period.

The figure for the number of writes to the files has the same problems as that of the read-write files above. Again, we are counting as writes all of the I/O operations that occurred during any open in APPEND, OVERWRITE or UPDATE mode. Since many of the opens in question are UPDATE mode opens and a fraction of the I/O operations in UPDATE mode are reads, the actual number of writes could actually be smaller than reported here.

What fraction of the system activity is generated by shared files? Only 7% of the permanent files are shared, and shared files only account for 15% of the volume of permanent files. However, shared files are quite heavily used. They are responsible for 27% of all opens to permanent files, for 64% of all reads and for 31% of all writes. In other words, shared files are an important subset of the permanent files as far as system activity is concerned. As a first approximation (distributions will be shown later) shared files also tend to be larger and to be opened more frequently in read-only mode than other permanent files

One important aspect of file sharing is the number of users that share a particular file, both for reading and for writing. Table III shows the distributions for these variables.

The table has been cut after twenty users but it still covers 98% of the values observed. The maximum observed for the number of users is 55, the mean is 3.7 and the median and the mode are both 2 users. Another impor-

Table III. Number of users per shared file. SLAC trace.

Number	All		Readers		Writers	
of Users	Freq	Cum	Freq	Cum	Freq	Cum
0	0.0	0.0	0.16	0.16	0.37	0.37
1	0.0	0.0	0.10	0.26	0.33	0.70
2	0.59	0.59	0.40	0.66	0.19	0.89
3	0.15	0.74	0.12	0.78	0.041	0.904
4	0.080	0.82	0.055	0.835	0.018	0.922
5	0.041	0.861	0.040	0.875	0.0	0.922
6	0.021	0.882	0.012	0.887	0.0093	0.9313
7	0.021	0.903	0.020	0.907	0.0031	0.9344
8	0.017	0.920	0.0093	0.9163	0.010	0.9444
9	0.010	0.930	0.010	0.9263	0.0015	0.9459
10	0.010	0.940	0.010	0.9363	0.0	0.9459
11	0.0077	0.9477	0.0062	0.9425	0.0015	0.9474
12	0.0077	0.9554	0.0062	0.9487	0.0015	0.9489
13	0.0046	0.9600	0.0031	0.9518	0.0031	0.9520
14	0.0031	0.9631	0.0031	0.9549	0.0	0.9520
15	0.0015	0.9646	0.0015	0.9564	0.0	0.9520
16	0.0046	0.9692	0.0046	0.9610	0.0	0.9520
17	0.0	0.9692	0.0	0.9610	0.0	0.9520
18	0.0031	0.9723	0.0031	0.9641	0.0	0.9520
19	0.0015	0.9738	0.0031	0.9672	0.0	0.9520
20	0.0031	0.9769	0.0015	0.9687	0.0	0.9520

tant characteristic, from the Writers columns, is that 70% of shared files are only written by one or less users (creation of the file is not counted as a write operation).

2.2.2. Activity Over Time

The number of permanent files opened per unit of time is a good indicator of system activity. In Fig. 2.1, the unit of time chosen is 4 hours. The plot of the number of opens per four-hour period shows quite conclusively that the system's activity cannot be considered stationary. The data has no obvious trend, either in average or in maximum values. However, there is a very strong seasonality with the hour of day and day of week. The same comments apply to the shared files.

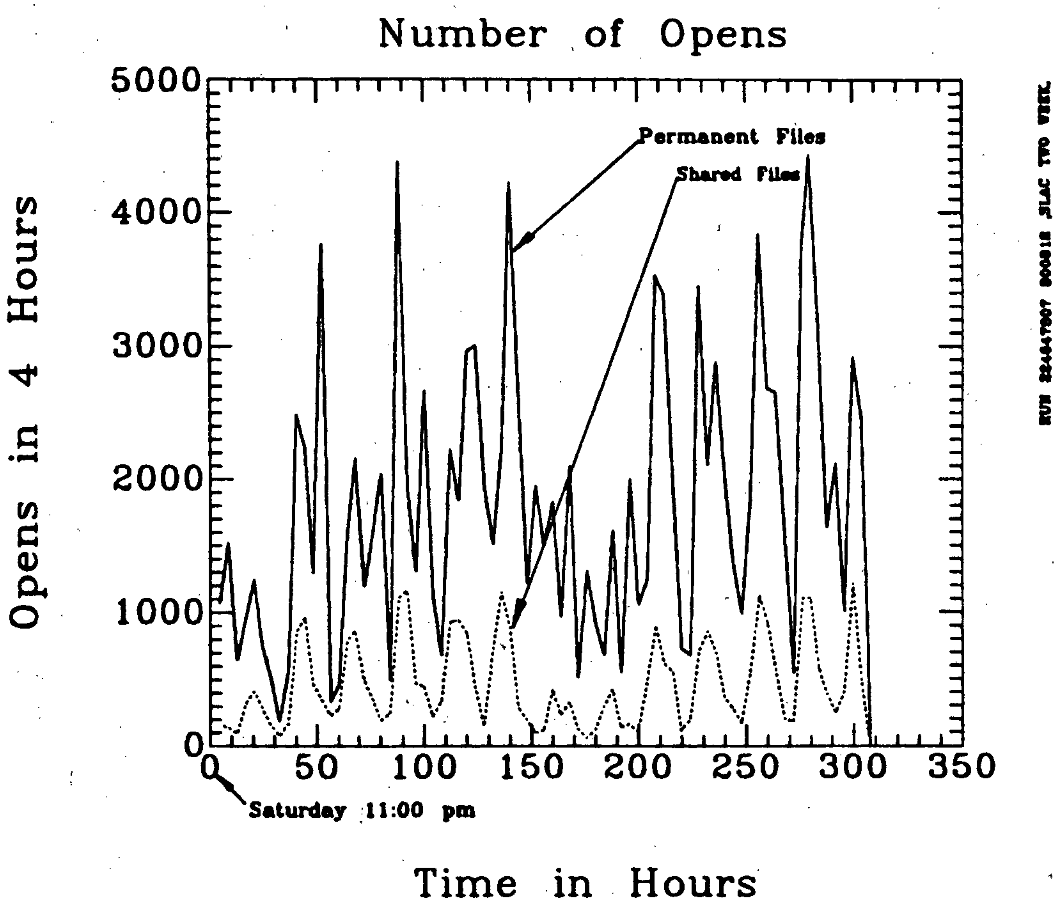

Fig 2.1. Number of opens per 4 hour period. Slac trace. Weekends can be seen clearly. Activity is about half that of week days. The peaks of activity are centered around noon.

Files can be opened basically in two modes: read-only mode and read-write mode. The distinction between read-only and read-write usage of files is important to us because it affects the consistency of replicated data. Fig 2.2 shows the number of IO operations in read-only mode for permanent and shared files. There is no visible trend in the activity from one day to the next but there are wide changes of activity during the day. We also note that read operations from shared files are a big fraction of the total number of reads.

When files are opened in read-write mode, our traces do not show whether the IO operations performed on the file are reads or writes. We have to lump them together and assume that they are potential writes. Fig. 2.3 shows that the number of IO operations in read-write mode is about half that of reads. The fraction of writes to shared files is smaller than that of the

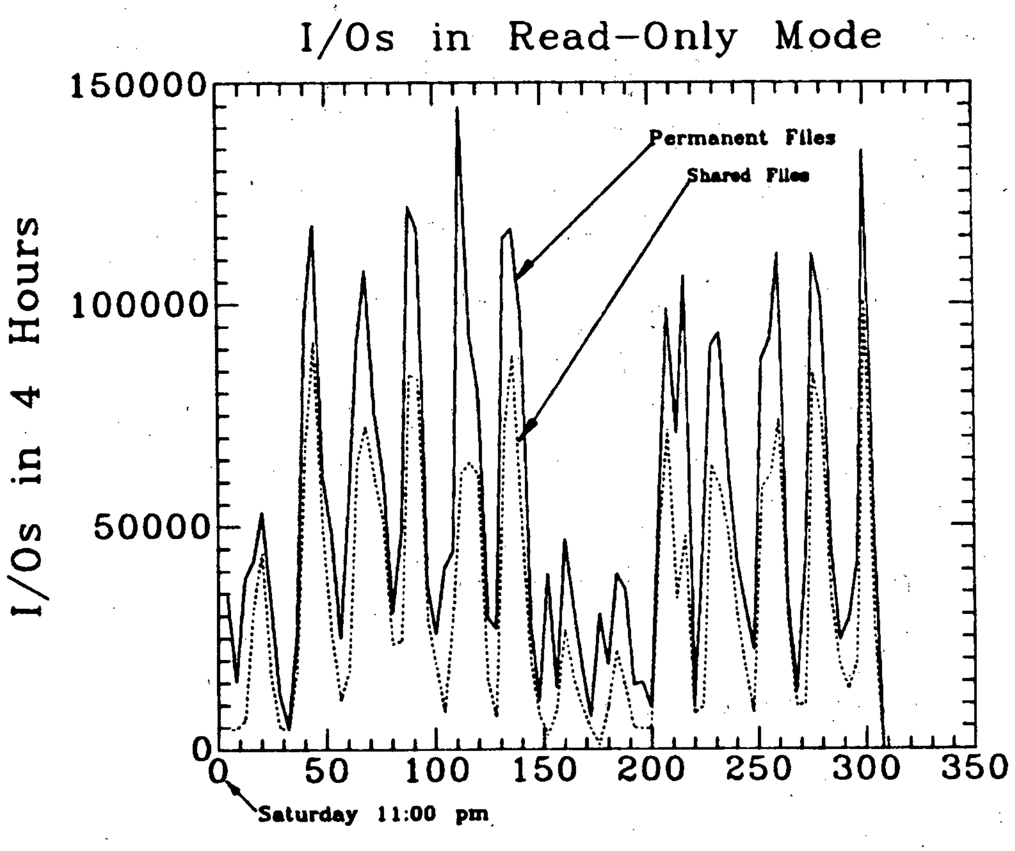

Fig 2.2. Number of read operations per 4 hour period. (Slac trace). Shared files are a big fraction of the total activity. Peak values for permanent and shared files do not show any visible trend.

reads from them. Within the limitations of our data, we have to conclude that shared files are accessed in read-only mode more often than the general population of permanent files. This should encourage the use of multiple copies of the files.

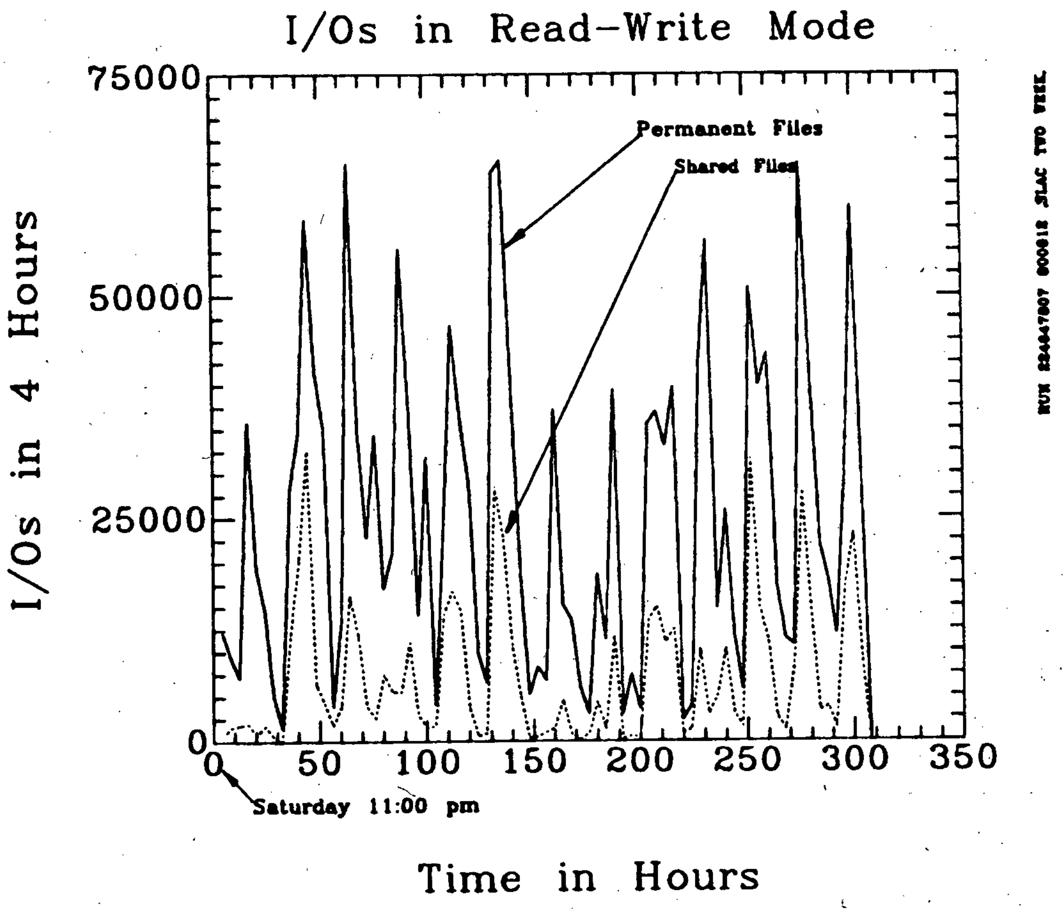

Fig 2.3. Number of IO operations in read-write mode per 4 hour period. (Slac trace).

2.2.3. File Usage Characteristics

File size is an important parameter when working with files. It has been shown that it is a good predictor of file usage [Smi81a]. Fig. 2.4 shows that files on disk tend to be smaller than files on tape. Let us point out that at Slac huge amounts of data from physics experiments are stored on tapes to be analyzed later on. Likewise, the shared files tend to be larger than the permanent files as a class. Most disk files have sizes between 10 kilobytes and 10 megabytes. The exact distribution can be seen on Fig. 2.4.

It was pointed out in [Smi81a] that most files are used (opened) a small number of times. If we look at Fig. 2.5 we see indeed that 60% of all permanent files are used only once or twice. However, shared files, and specially shared files on disk, are used many more times. Actually 95% of these files are opened more than twice [Cof73].

A measure that is relevant to the study of file migration is the amount of IO activity per open. As can be seen in Fig. 2.6, files on disk have fewer IO's per open than files on tape. In particular, 40% of the opens (for permanent files on disk) result in less than three IO operations being performed.

For policies that try to minimize the communications traffic, a very important measurement of activity is the amount of information that is actually accessed during an open. A good way to measure this amount of information is as a fraction of the size of the file. For the permanent files on disk and for the shared files on disk, only a small number of the opens result in an access to the whole file . One of the reasons for this is the large number of shared files that are partitioned data sets (libraries) and concatenated data sets. In both cases, only a small fraction of the total amount of information is needed at each open. In 20% of the opens, the file is accessed repeatedly

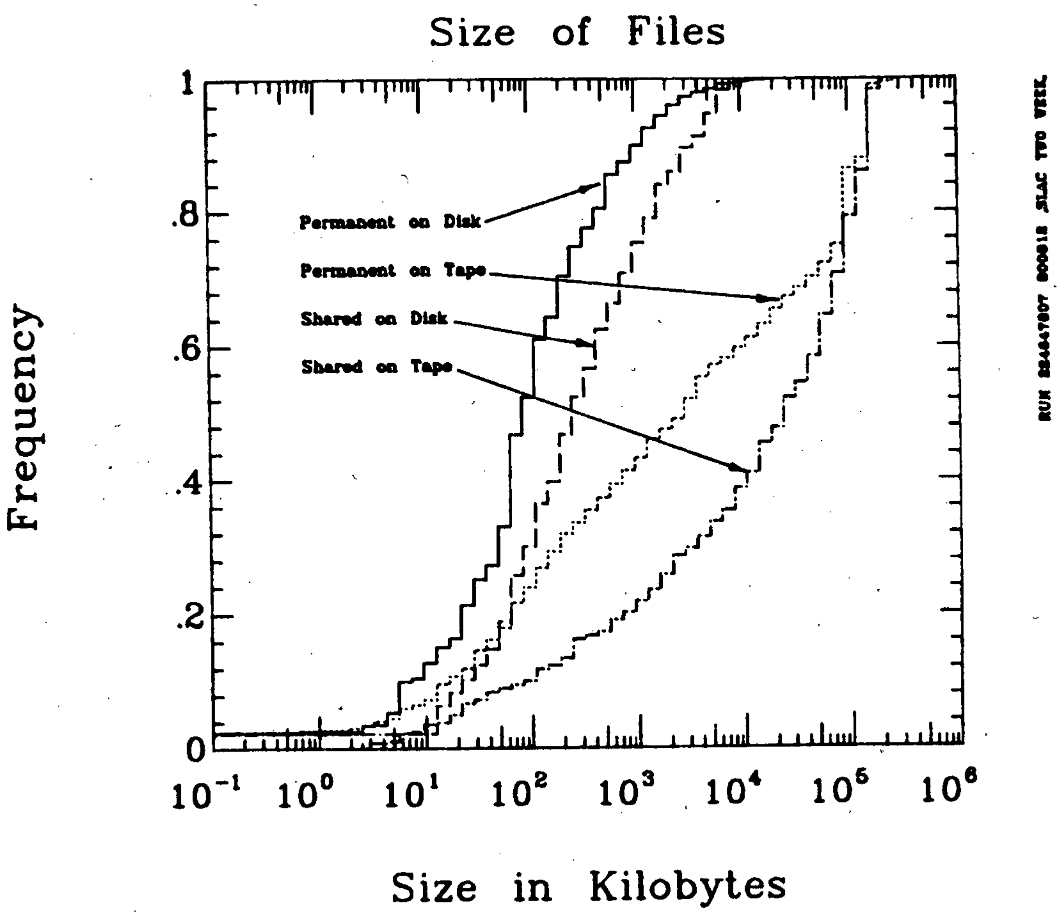

Fig 2.4. File size distributions for permanent and shared files. Statistics for these distributions are shown in Table 2.4. SLAC trace.

Table 2.4. Size of Files in Kilobytes (SLAC)

Type of File	Min	Max	Mean	Median	Std. Deviation
Perm. on Disk	0.	94000.	549.	80.	2290.
Perm. on Tape	0.	339000.	41500.	2700.	61400.
Shared on Disk	3.2	15025.	1200.	240.	2150.
Shared on Tape	0.	338000.	56900.	25000.	15200.

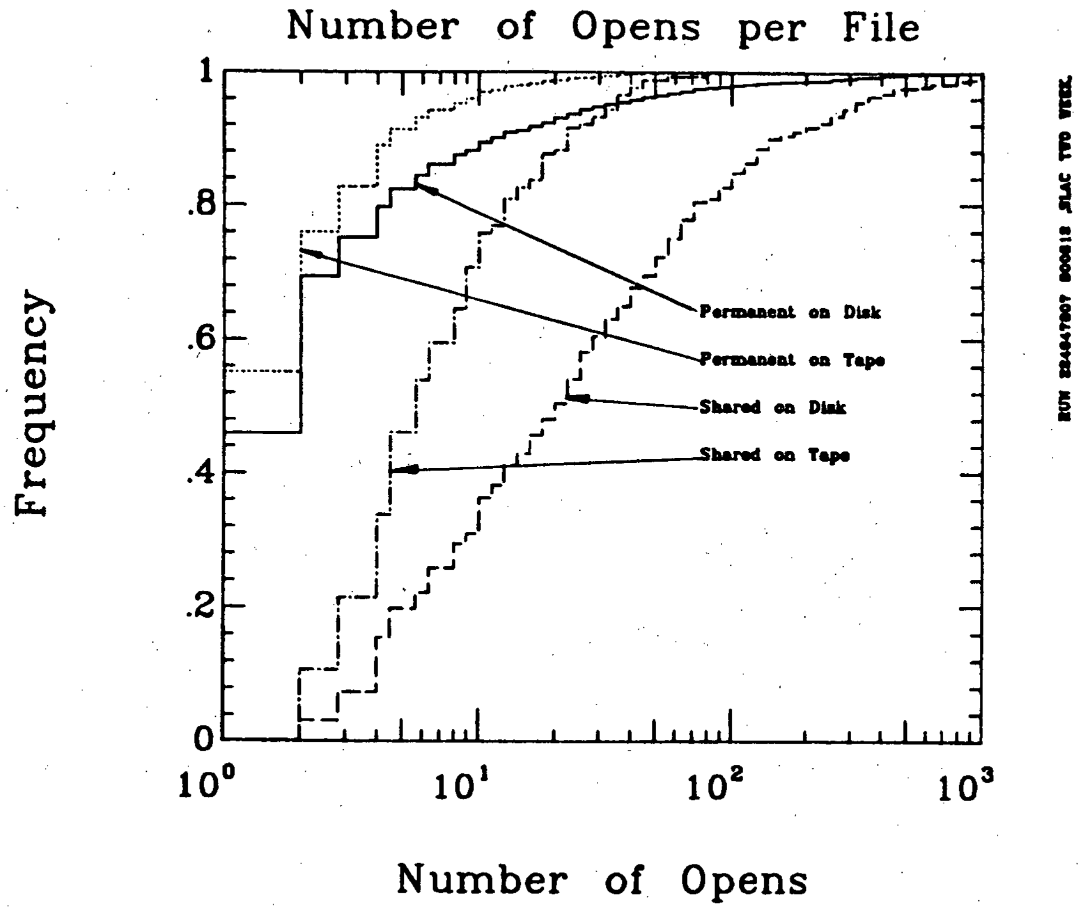

Fig 2.5. Distribution of the number of opens per file. The parameters for the distributions are shown in Table 2.5. SLAC trace.

Table 2.5. Number of Opens per File (SLAC)

Type of File	Min	Max	Mean	Median	Std. Deviation
Perm..on Disk	0.	2661.	12.	2.	66.
Perm. on Tape	1.	86.	2.6	1.	4.7
Shared on Disk	2.	2661.	76.	22.	11.
Shared on Tape	2.	86.	10.	5.	11.

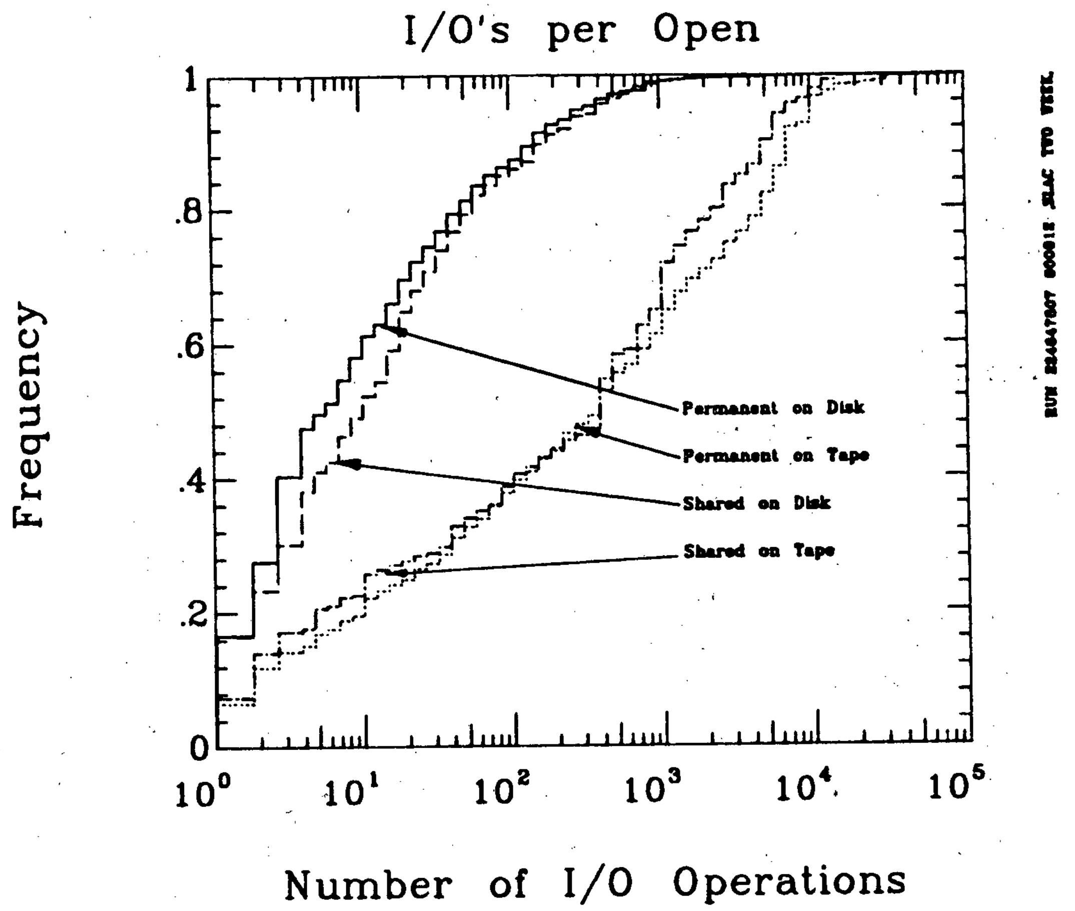

Fig 2.6. Distributions of the number of IO operations per open. SLAC trace. Statistics for these distributions are presented in Table 2.6.

Table 2.6. Number of IO's per Open (SLAC)

Type of File	Min	Max	Mean	Median	Std. Deviation
Perm. on Disk	0.	30156.	77.	5.	398.
Perm. on Tape	0.	70012.	2630.	280.	5400.
Shared on Disk	0.	27300.	88.	9.	437.
Shared on Tape	0.	21100.	1640.	280.	2900.

(as in a multiple-pass algorithm) and the total amount transferred is hence larger than the size of the file itself. Fig. 2.7 shows the measured distribution.

There are quite a few measurement problems associated with determining the amount of information accessed per open as well as the size of certain files. The number of bytes transferred during an open is determined by multiplying the block size of the file by the number of I/O's associated with the open. The problem here is that the number of I/O's as counted by SMF can differ from the number of I/O records actually transferred (sometimes SMF counts the I/O operations required to open and close the file; sometimes the block size is not reported).

A serious problem in determining the size of files is created by the existence of concatenated data sets. Concatenated data sets are obtained by concatenating a number of files and assigning them a new name (this procedure is often used for subroutine libraries during linking of programs). SMF reports only the name of the first file in the concatenated data set and yet the size is that of the whole data set. This make very difficult to keep track of the size of files that only appear as members of concatenated data sets.

For the shared files, we wanted to know whether the size of the file is a good indicator of the fraction that is accessed at each open. We divided the files in five classes according to size.

Fig. 2.8 shows the distribution of the percentage of file accessed per open for the five classes of files. If we disregard the very small files (< 10 Kbytes), the fraction of file transferred per open is smaller for larger files.

Our last set of measurements is concerned with the frequency of usage of the files as shown by the distribution of the interopen time, i.e., the time between a file being opened and it being opened again. As indicated above,

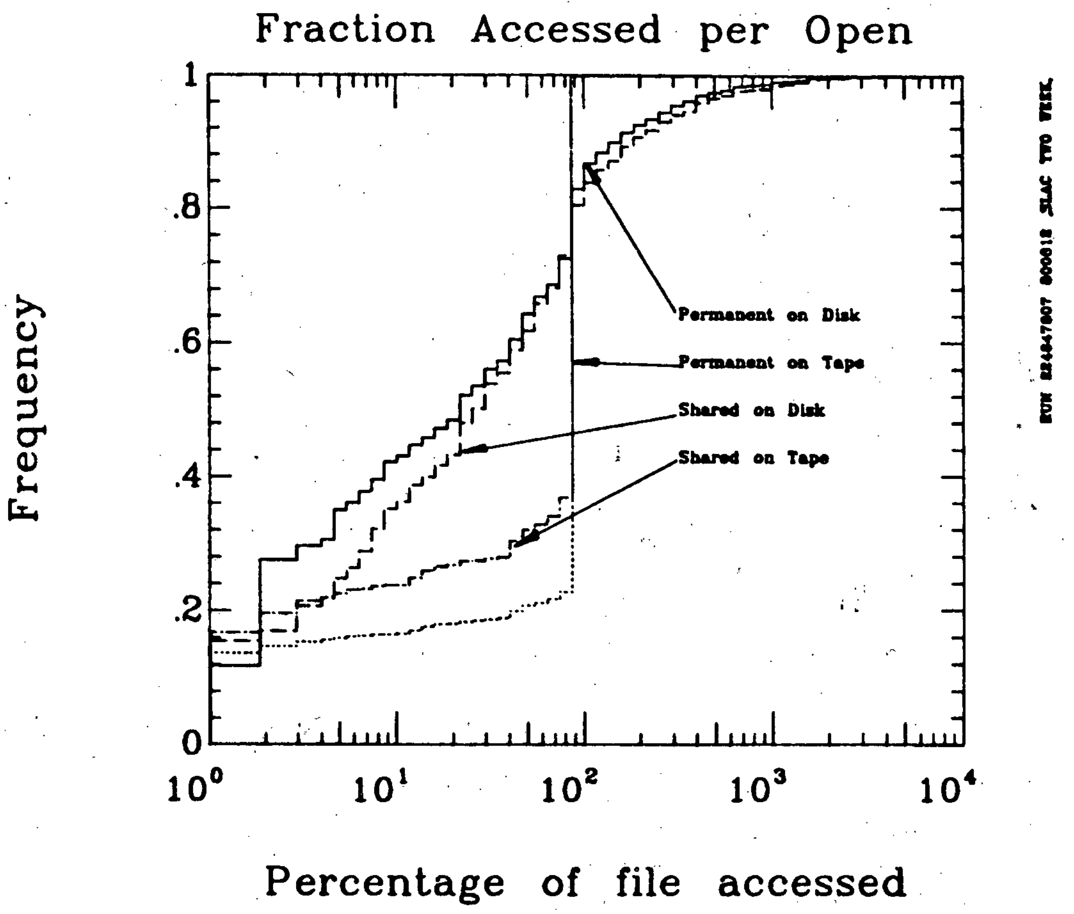

Fig 2.7. Distributions of the percentage of the file size accessed per open. SLAC trace. Statistics of these distributions are shown in Table 2.7.

Table 2.7. Percentage of File Accessed per Open (SLAC)

Type of File	Min	Max	Mean	Median	Std. Deviation
Perm. on Disk	0.	72500.	98.	21.	693.
Perm. on Tape	0.	100.	80.	100.	38.
Shared on Disk	0.	44122.	113.	29.	620.
Shared on Tape	0.	100.	69.	100.	42.

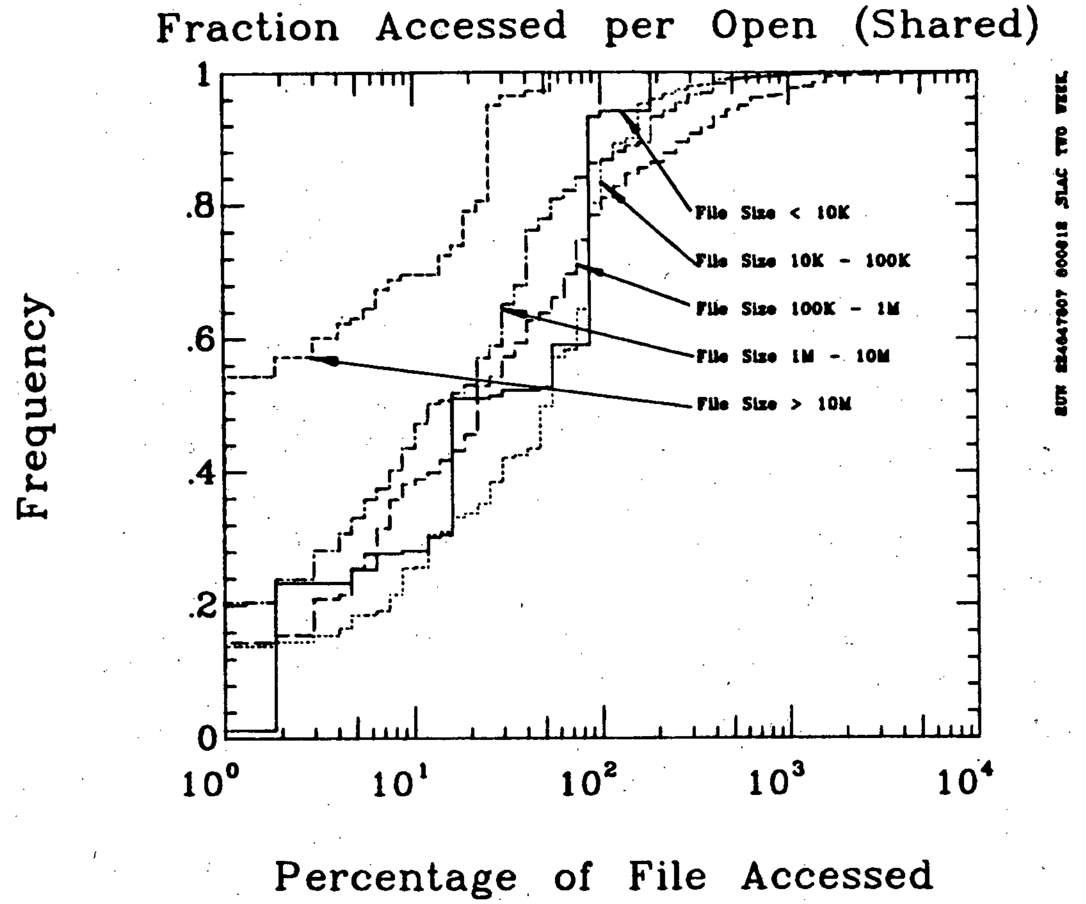

Fig 2.8. Distributions of the percentage of file size accessed per open for the five size classes. SLAC trace. More statistics in Table 2.8

Table 2.8. Percentage of File Accessed per Open (SLAC)

File Size	Min	Max	Mean	Median	Std. Deviation
< 10 KBytes	0.	200.	55.	18.	55.
10 - 100 KBytes	0.	8720.	79.	50.	204.
100 - 1000 KBytes	0.	44122.	154.	22.	859.
1 - 10 MBytes	0.	4778.	63.	12.	172.
> 10 MBytes	0.	59.	9.4	1.	14.

we have not considered the time after the last close of the file. Various solutions to the right censoring problem can be found in [Smi81b]. Fig. 2.9

shows the distribution of interopen times for the permanent files at Slac. Again we note the fact that tape files have larger interopen times than disk files. Interopen times for tape files are two to three orders of magnitude larger than those for the disk files. More than 80% of the interopen times for the disk files are under one hour.

It has been noted in [Smi81a] that the distribution of interopen times varies with the size of the file. Fig. 2.10 shows these distributions for the five

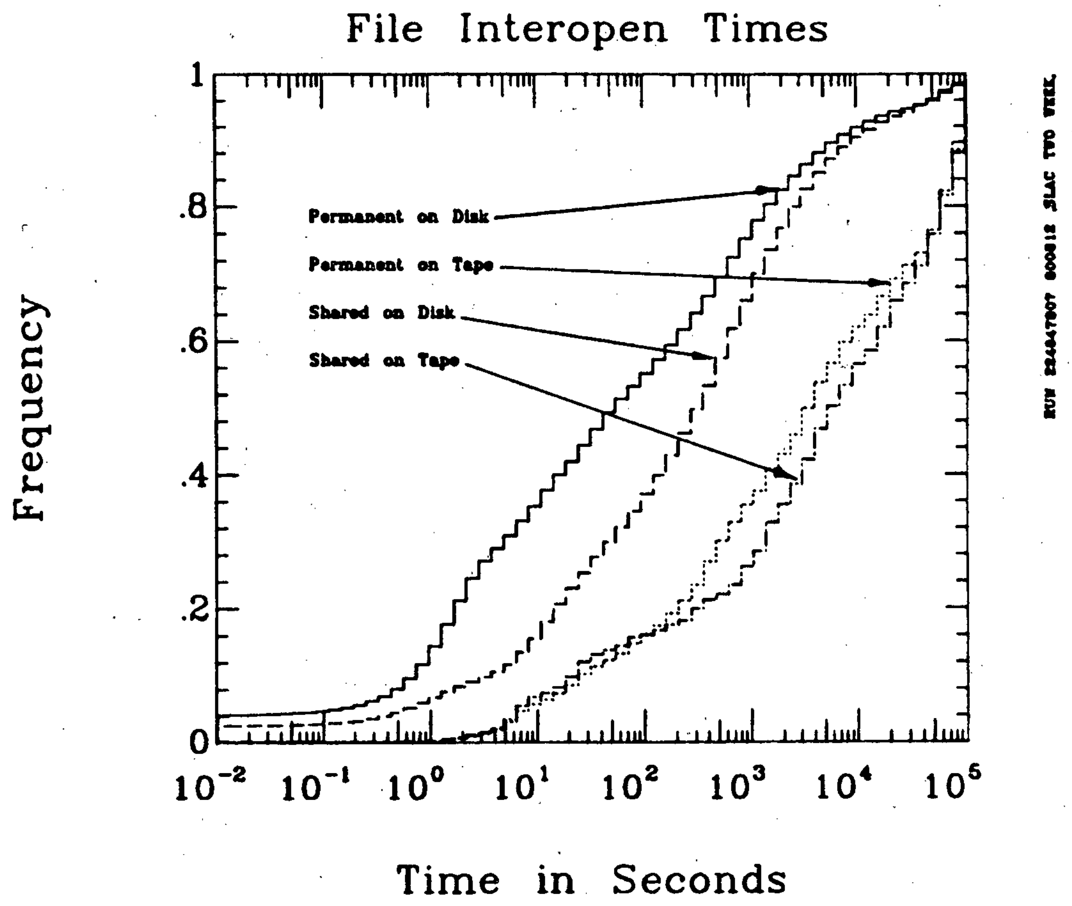

Fig 2.9. Distributions of interopen times for the SLAC trace. More information about the distributions is given in Table 2.9.

Table 2.9. Interopen Times in Seconds (SLAC)

Type of File	Min	Max	Mean	Median	Std. Deviation
Perm. on Disk	0.	967000.	8350.	50.	44300.
Perm. on Tape	0.25	949000.	50200.	20000.	114000.
Shared on Disk	0.	862000.	8030.	360.	36000.
Shared on Tape	0.70	949000.	45800.	70000.	95800.

size classes that we used in Fig. 2.8.

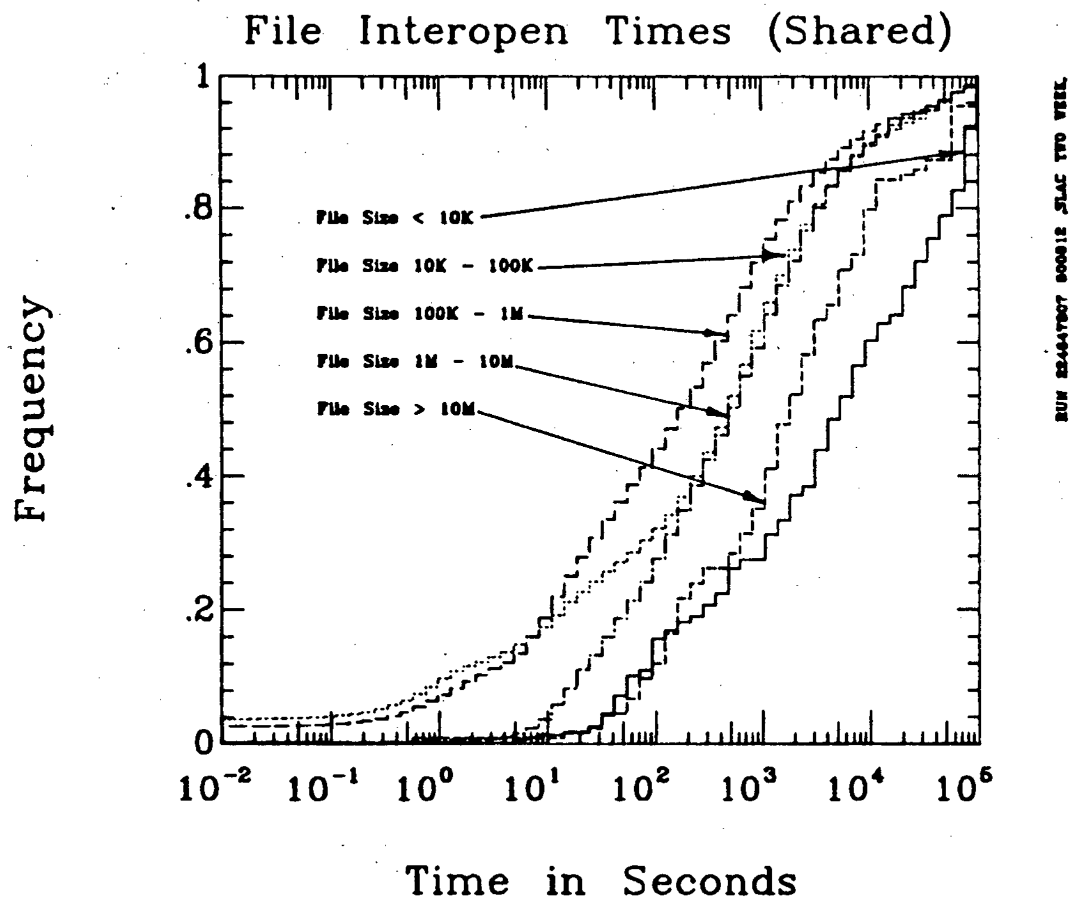

Fig 2.10. Distributions of interopen times for shared files for the SLAC trace. More statistics for the distributions are presented in Table 2.10.

Table 2.10. Interopen Times in Seconds (SLAC)

File Size	Min	Max	Mean	Median	Std. Deviation
< 10 KBytes	5.	521000.	37000.	5100.	75000.
10 - 100 KBytes	0.	862000.	8131.	500.	33900.
100 - 1000 KBytes	0.	850000.	7650.	170.	38000.
1 - 10 MBytes	0.	686000.	7510.	450.	30000.
> 10 MBytes	0.5	338000.	19800.	1900.	52000.

From figure 2.10, if we disregard the case of the very small files (< 10 Kilobyte), it can be seen that smaller files have shorter interopen times. In particular, interopen times of less than two minutes are almost nonexistent for files larger than one megabyte. We have observed some very long sequences of repeated opens to the same file. These opens are very close to one another (typically within one second) and they are partly responsible for the large proportion of short interopen times. We suspect that this is the result of some non-standard usage of the file system. However, this peculiarity does not affect much our experiments since none of our algorithms take decisions in such short periods of time.

A similar analysis of the data is now presented on the trace data originated at the Hughes Aircraft computer center.

2.3. The Hughes Installation

The Hughes Aircraft computer center is in many respects similar to the SLAC center. For example, the hardware and the basic file system are almost identical. On the other hand, the interface to the user is quite different because the Hughes installation uses a standard IBM product, TSO, to support its time-sharing users.

This fact has two important consequences in our study of the file system. The first is that files created under TSO are easy to recognize because the installation uses a convention for naming files. The second is that, since TSO runs as a single job step for each user, the open time of the files accessed under TSO is reported by SMF as being the starting time of the TSO step. This produces some errors in the evaluation of interopen times, for example.

2.3.1. Basic Numbers

The trace spans a period of 9 days (191 hours), starting Monday, the 19th of September, 1977, at 0:00 am. During this period of time, 1637 users submitted 25,039 jobs. 854 of these users were involved in accessing shared files. This installation has three times more users than SLAC but the number of jobs executed per week day is about the same. The weekend activity is much lower, though. On weekends, SLAC runs roughly half the number of jobs that it runs on week days. The activity at Hughes on weekends is below 20% that of regular week days. Table IV contains more information about jobs and accounts at Hughes.

The trace for the Hughes system shows that 170,000 files were accessed during the 9 days during which the system was observed. When compared to the SLAC system (150,000 files in 13 days), it appears that the Hughes system has a higher level of I/O activity. Table V also shows that the number of I/O

Table IV. Basic Job Counts. Hughes Trace.

Number of hours	191
Number of wk-days	7
Number of wkend-days	2
Number of jobs	16815
Number of jobs/wk-day	2273
Number of jobs/wkend-day	450
Number of accounts	1637
Accounts sharing files	854

operations (reads + writes) on week days at Hughes is twice the number at SLAC. Of the 170,000 files, 18,343 are permanent and 2,301 are shared files. This again represents from three to four times the number of files at SLAC.

All the comments that were made about the contents of Table II hold for Table V. In particular, we note that the number of writes is particularly high compared to the number of reads. This is due, as explained earlier, to the failure by SMF to indicate the real type of I/O operation that is performed on files.

The shared files also in this system are responsible for a large fraction of file activity. Forty percent of the opens and forty percent of the IO activity of permanent files comes from shared files.

The number of users per shared file has a distribution very similar to that of the SLAC system. Table VI shows the distributions for these variables. In this case, we have cut the table after 10 users because that covers 99% of the observed cases. The maximum observed is 109 users per file, the mean

Table V. Basic File Counts. Hughes Trace.

	TAPE	TEMP	PERM	SHARED	SYS	TOTAL
Number of files	5020	133797	18343	2301	14485	171645
Number of files initially	316	0	7992	1149	1507	9815
Number of files at end	5020	0	8831	1813	4153	18004
Number of read-write files	4044	-	14117	2001	13338	31500
Volume of files (MBytes)	43448	34107	4697	462	9135	91387
Ave. files created/wk-day	588	17480	2507	323	1802	22378
Ave. files created/wkend-day	452	5718	396	21	935	7501
Ave. files scratched/wk-day	0	17482	1224	59	1241	19946
Ave. files scratched/wkend-day	0	5713	473	39	823	7009
Ave. opens/wk-day	1096	37186	15983	7465	16684	70948
Ave. opens/wkend-day	798	14422	1449	472	5473	22142
Ave. reads/wk-day	869770	528670	290716	126255	1253803	2942959
Ave. reads/wkend-day	852920	368797	107609	12051	713455	2042781
Ave. writes/wk-day	743056	2136479	248226	93795	1899712	5027473
Ave. writes/wkend-day	1077515	1275317	133746	12943	675122	3161699

Table VI. Number of users per shared file. Hughes trace.

Number	All		Readers		Writers	
of Users	Freq	Cum	Freq	Cum	Freq	Cum
0	0.0	0.0	0.11	0.11	0.13	0.13
1	0.0	0.0	0.22	0.33	0.37	0.50
2	0.69	0.69	0.45	0.78	0.40	0.90
3	0.19	0.88	0.14	0.92	0.072	0.972
4	0.065	0.945	0.042	0.962	0.013	0.985
5	0.014	0.959	0.011	0.973	0.0056	0.9906
6	0.0082	0.9672	0.0073	0.9803	0.00086	0.9915
7	0.0043	0.9715	0.0039	0.9842	0.0	0.9915
8	0.0021	0.9736	0.0017	0.9859	0.00043	0.9919
9	0.0013	0.9749	0.00086	0.9868	0.0	0.9919
10	0.00086	0.9758	0.00086	0.9876	0.0	0.9919

is 2.6 users/file and the median and the mode are both 2 users. An important characteristic, from the Writers columns, is that 50% of shared files are only written by one or less users (creation of the file is not considered a write operation).

2.3.2. Activity Over Time

The number of files opened per unit of time is a good indicator of system activity. In Fig. 2.11, the unit of time chosen is 4 hours. The plot of the number of opens per four hours shows quite conclusively that the system activity cannot be considered stationary. The data has no obvious trend, neither in average nor in maximum values. This figure can be compared with the corresponding figure from SLAC. In this case the activity on weekends is actually very low.

2.3.3. File Usage Characteristics

Most characteristics of the SLAC files can be found in the files of the Hughes system. Fig. 2.14 shows a difference in size between tape and disk

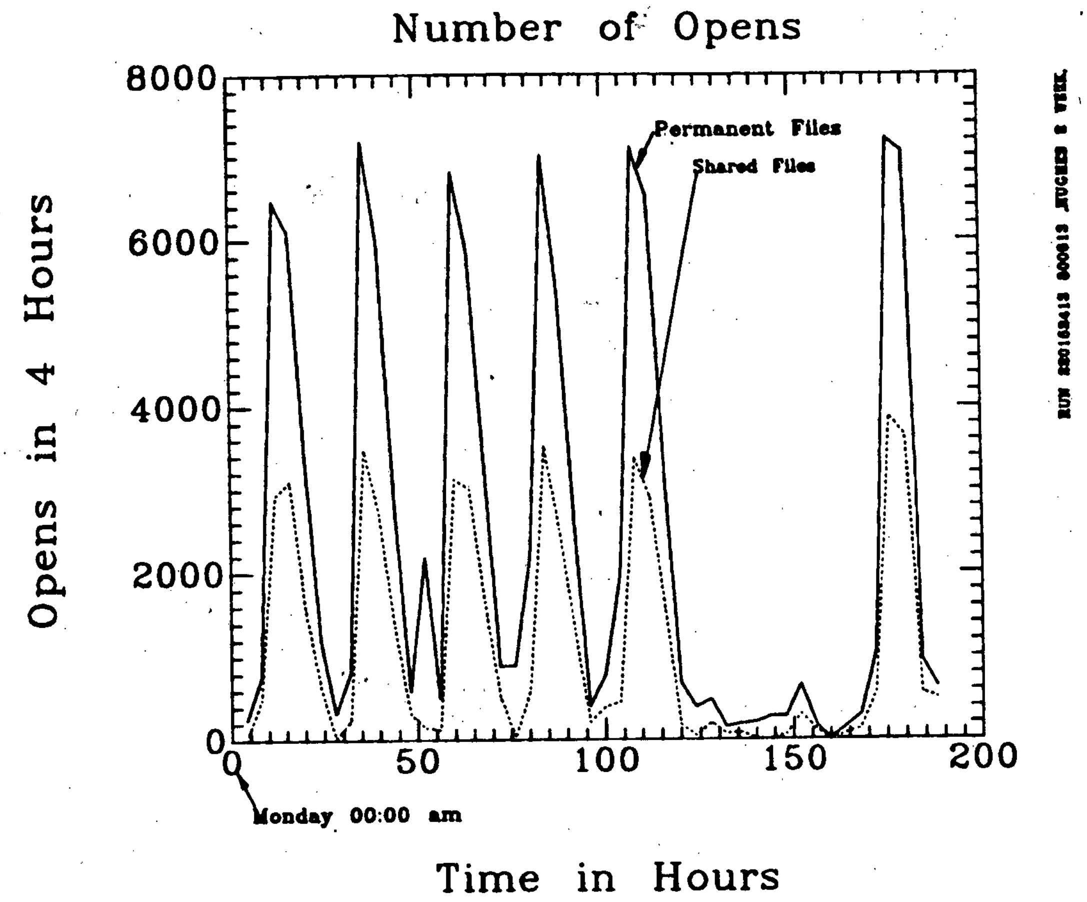

Fig 2.11. Number of opens per 4 hour period (Hughes trace). Weekends can be seen clearly. The number of opens is almost zero during the weekend. The peaks of activity are centered around noon.

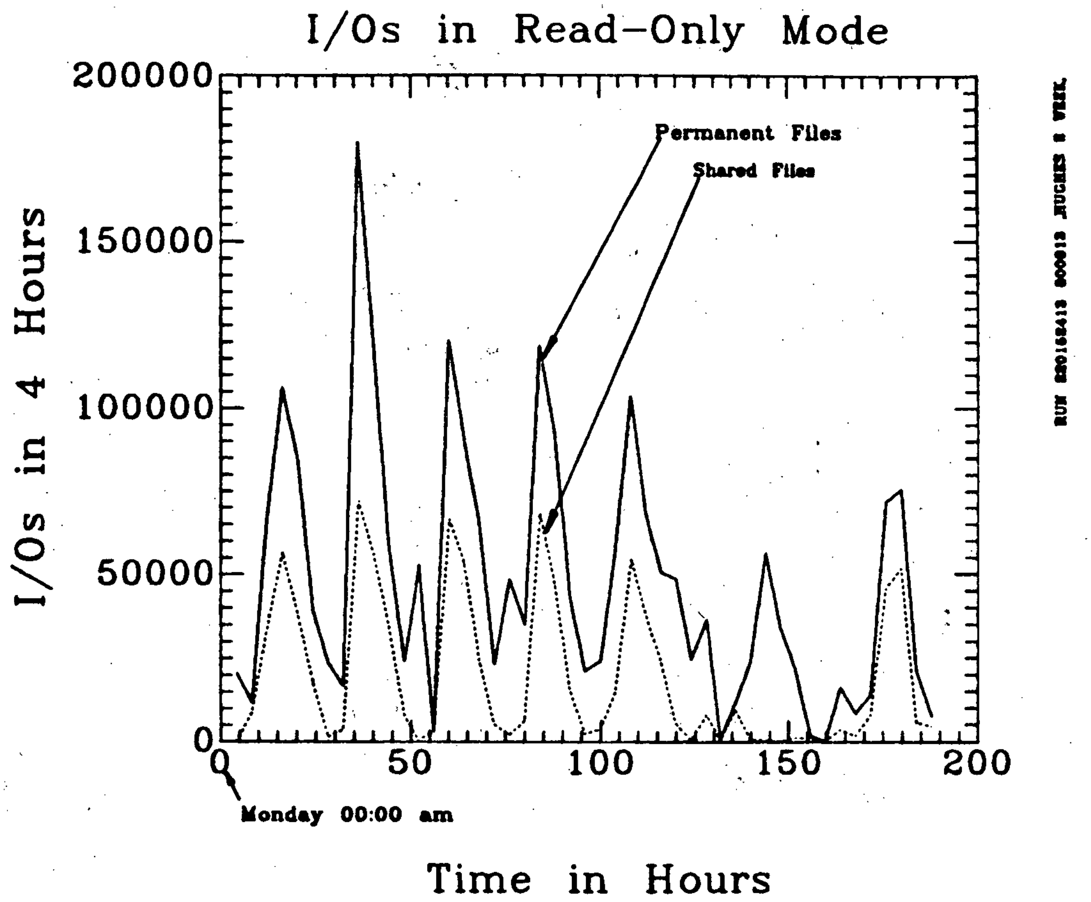

Fig 2.12. Number of read operations per 4 hour period. (Hughes trace). Shared files are a good fraction of the total activity. Peak values for permanent and shared files do not show any visible trend.

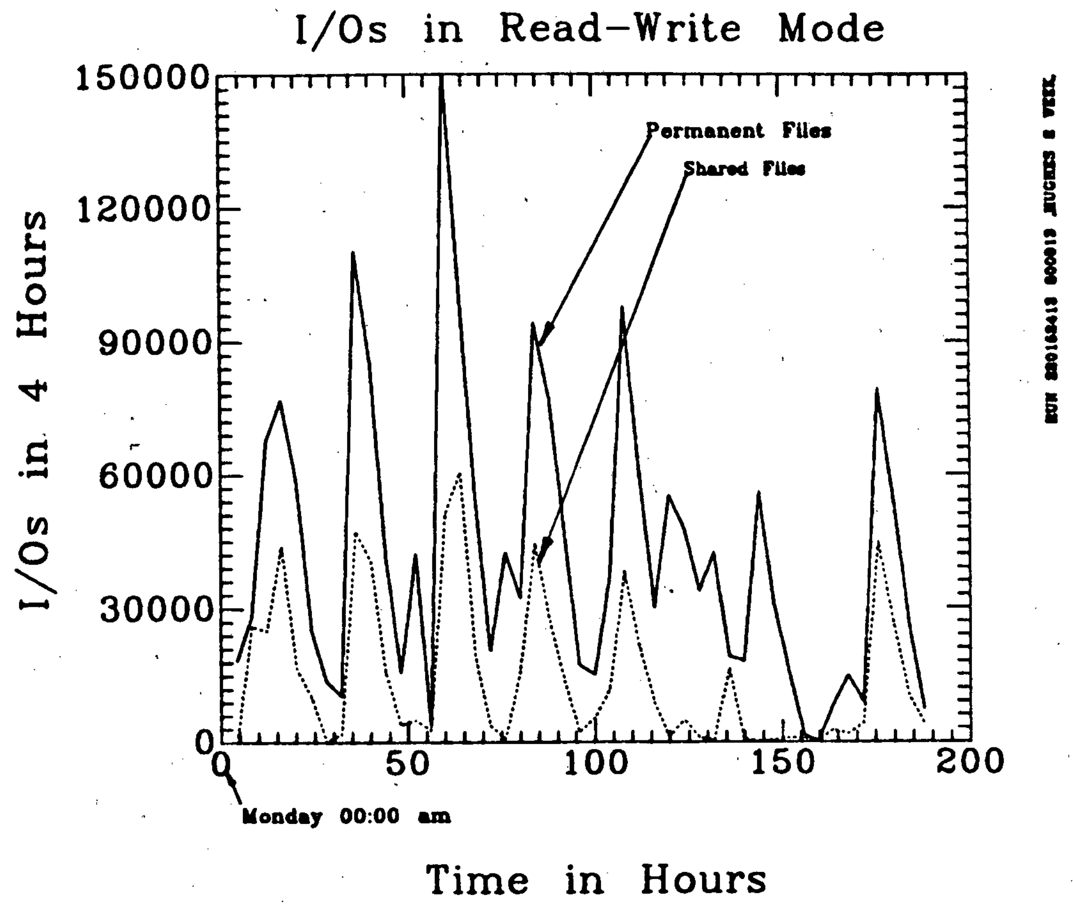

Fig 2.13. Number of IO operations in read-write mode per 4 hour period. (Hughes trace).

files of two orders of magnitude. Two comments about the disk files:

(1) There is very little difference (if any) between the shared files and the general population of permanent files.

(2) There is a large proportion of permanent files (almost 40%) that have exactly the same size: 12 kilobytes. This is probably a consequence of the way TSO files are allocated.

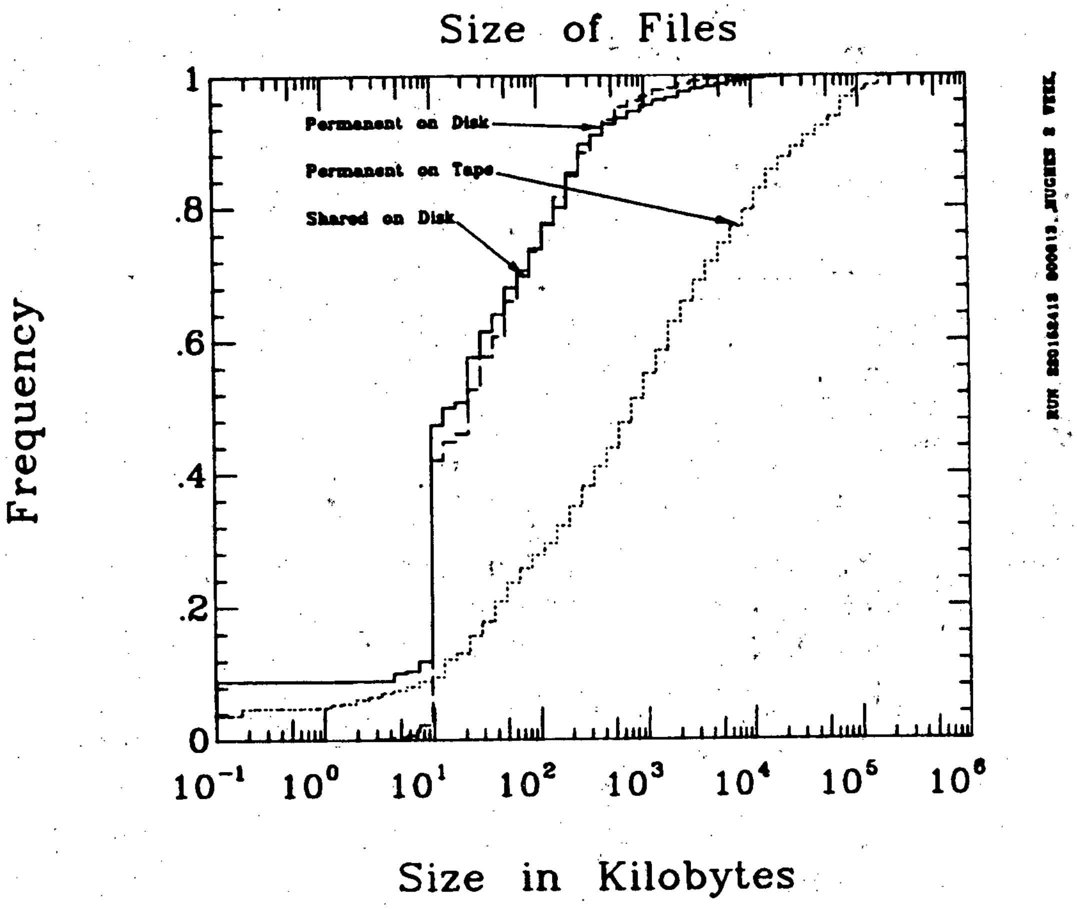

Fig 2.14. File size distributions for permanent and shared files. Hughes trace. More information about files sizes can be found in Table 2.14.

Table 2.14. Size of Files in Kilobytes (Hughes)

Type of File	Min	Max	Mean	Median	Std. Deviation
Perm. on Disk	0.	73700.	350.	18.	1940.
Perm. on Tape	0.	1310000.	12700.	700.	39000.
Shared on Disk	4.	40000.	215.	24.	1140.

Fig. 2.15 confirms that most permanent files are used only a few times. However, shared files are used many more times on the average. Actually

90% of the shared files are opened more than twice.

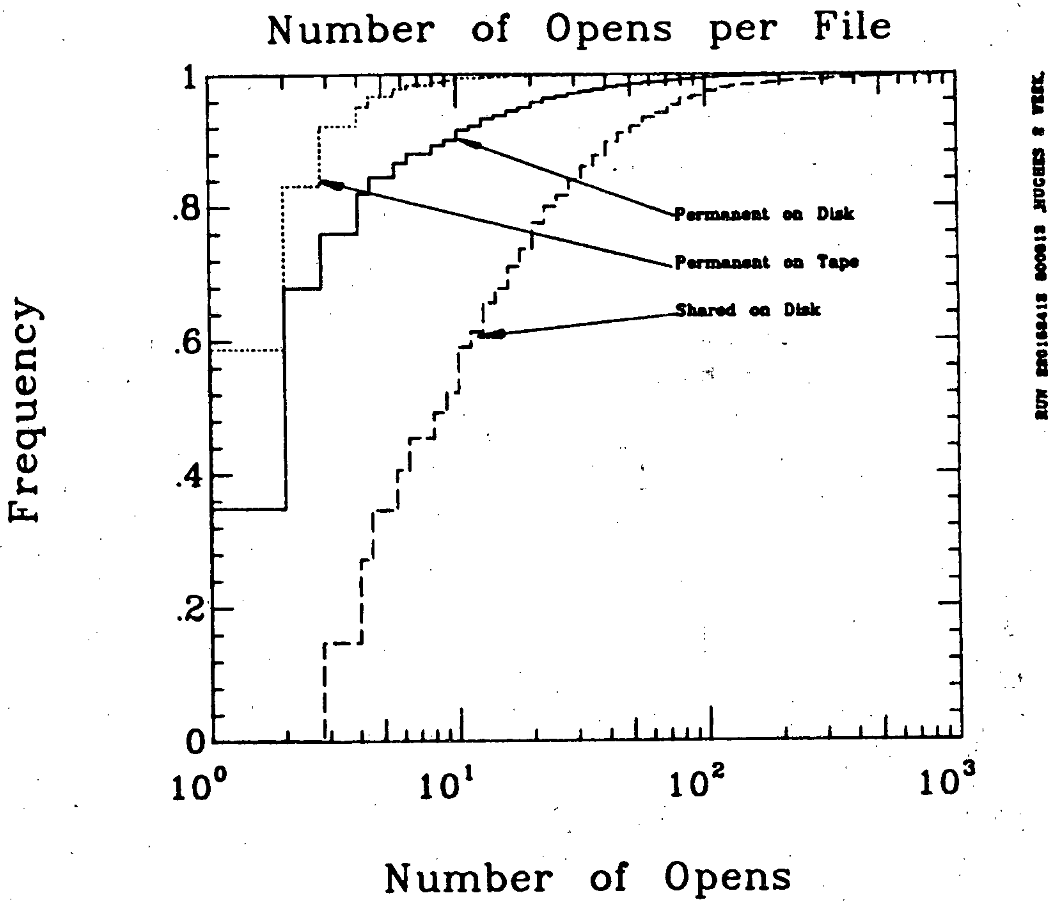

Fig 2.15. Distributions of the number of opens per file. Hughes trace. Moments for these distributions can be found in Table 2.15.

Table 2.15. Number of Opens per File (Hughes)

Type of File	Min	Max	Mean	Median	Std. Deviation
Perm. on Disk	0.	1046.	5.9	2.	26.
Perm. on Tape	1.	45.	1.8.	1.	1.9
Shared on Disk	3.	1046.	23.	9.	61.

Figure 2.16 shows again how similar are the shared files in this installation to the rest of the permanent files. It is also remarkable how similar is this distribution to that of Figure 2.6, the corresponding distribution for SLAC.

Fig. 2.17 contains the distribution of the fraction of file accessed per open in the Hughes system. Fig. 2.18 shows the same distribution broken

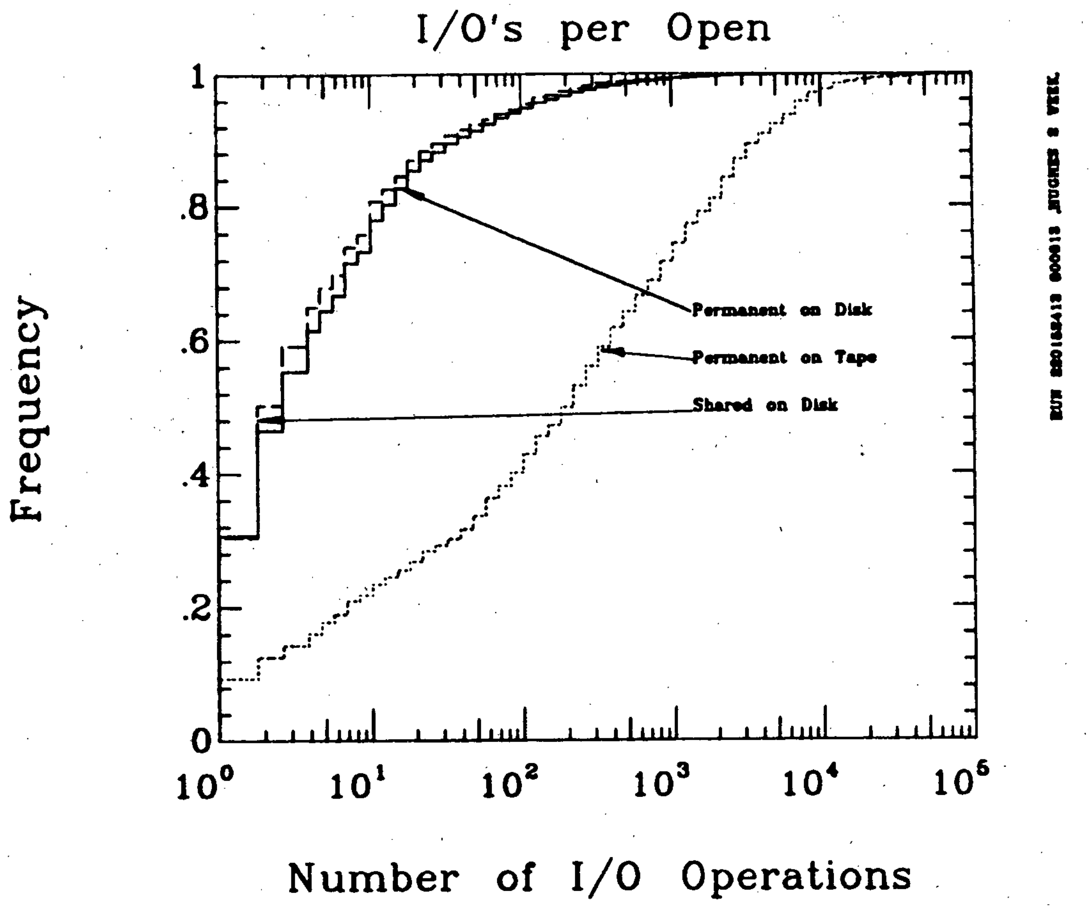

Fig 2.16. Distributions of the number of IO operations per open for the Hughes trace. Moments for the distributions are shown in Table 2.16.

Table 2.16. Number of I/O's per Open (Hughes)

Type of File	Min	Max	Mean	Median	Std. Deviation
Perm. on Disk	0.	30000.	39.	2.	270.
Perm. on Tape	0.	123000.	1630.	200.	5060.
Shared on Disk	0.	8500.	29.	2.	170.

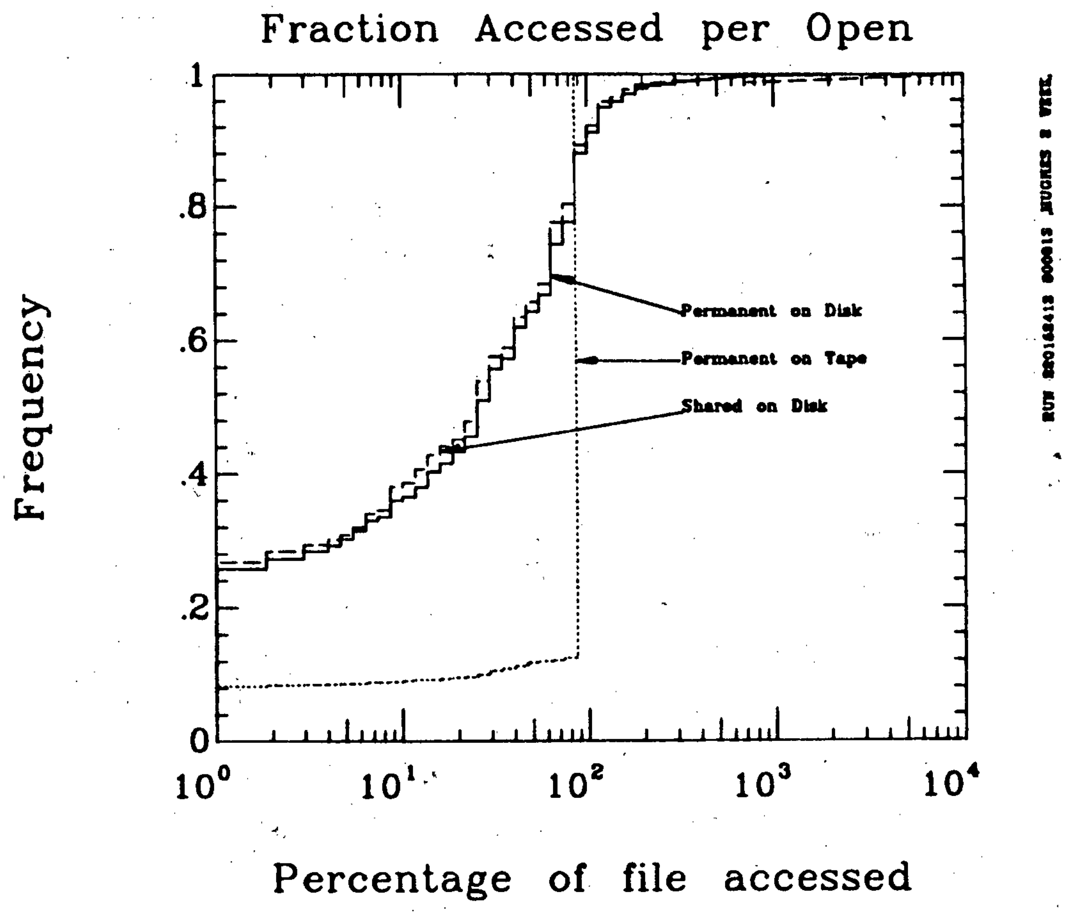

Fig 2.17. Distributions of the percentage of the file size accessed per open. Hughes trace. Moments for these distributions are shown in Table 2.17.

Table 2.17. Percentage of File Accessed per Open (Hughes)

Type of File	Min	Max	Mean	Median	Std. Deviation
Perm. on Disk	0.	63700.	57.	25.	320.
Perm. on Tape	0.	100.	88.5	100.	29.
Shared on Disk	0.	33000.	53.	25.	270.

down by size classes.

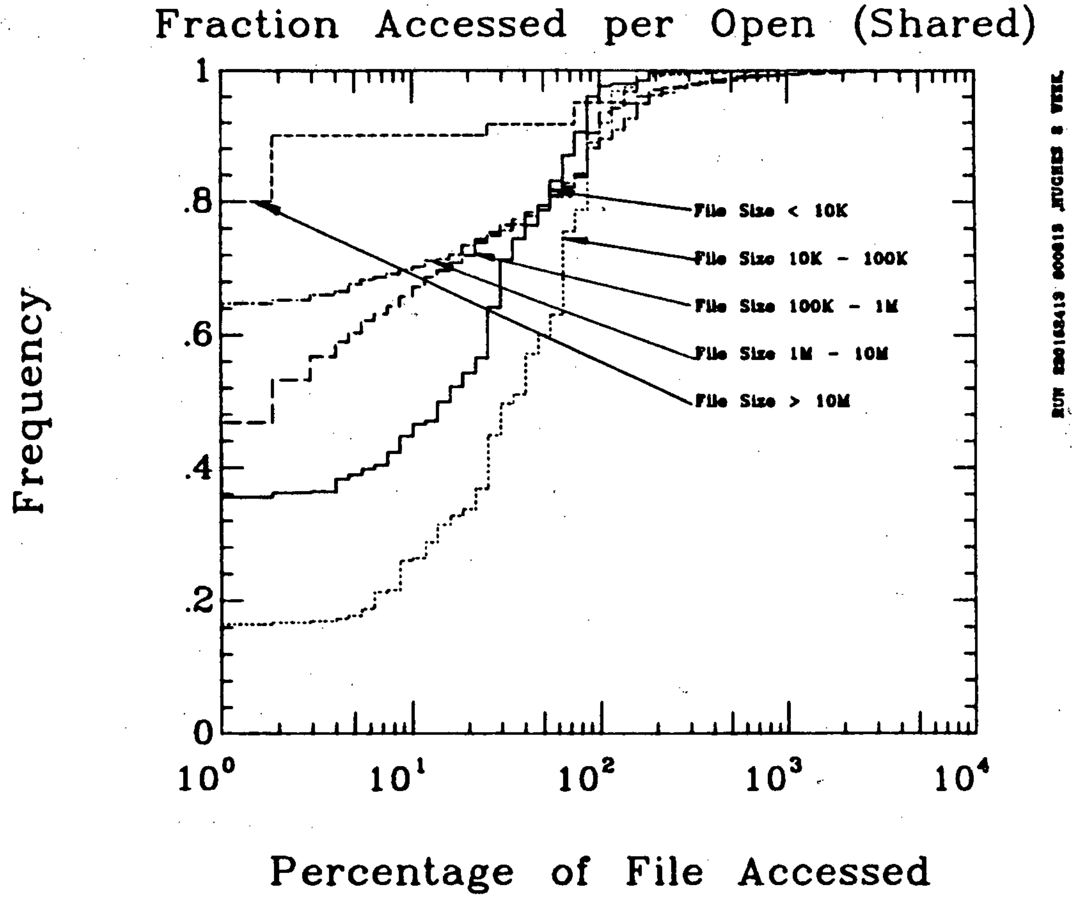

Fig 2.18. Distributions of the percentage of file size accessed per open for the five size classes. Hughes trace. See Table 2.18 for the moments of the distributions.

Table 2.8. Percentage of File Accessed per Open (Hughes)

File Size	Min	Max	Mean	Median	Std. Deviation
< 10 KBytes	0.	370.	29.	15.	40.
10 - 100 KBytes	0.	33900.	53.	35.	198.
100 - 1000 KBytes	0.	14000.	58.	2.	444.
1 - 10 MBytes	0.	3500.	47.	1.	174.
> 10 MBytes	0.	170.	11.	1.	37.

Our last set of measurements is concerned with the frequency of usage of the files as shown by the distribution of the interopen times. Fig. 2.19 shows the distribution of interopen times for the permanent files at Hughes. Fig. 2.20 is again the distribution of the interopen times when the files are put in classes according to their size. The classes are the same that are used in Fig. 2.18.

2.4. Conclusion

In this chapter, we have conducted an exploratory analysis of two large scientific computer installations: SLAC and Hughes Aircraft. The analysis has revealed many characteristics of the two systems. However, we would like to comment explicitly on three points:

(1) The file parameters from both systems are in the same order of magnitude. This is particularly true of the distributions of such measures as the number of opens per file, the size of the files, the fraction of the file accessed per open and the interopen times. Table VII gives a comparison of the means and medians of these distributions.

(2) The permanent files used by more than one user (shared files) are responsible for an important fraction of the system activity, (see Table VIII) and it is very likely that this will still be true in a distributed environment.

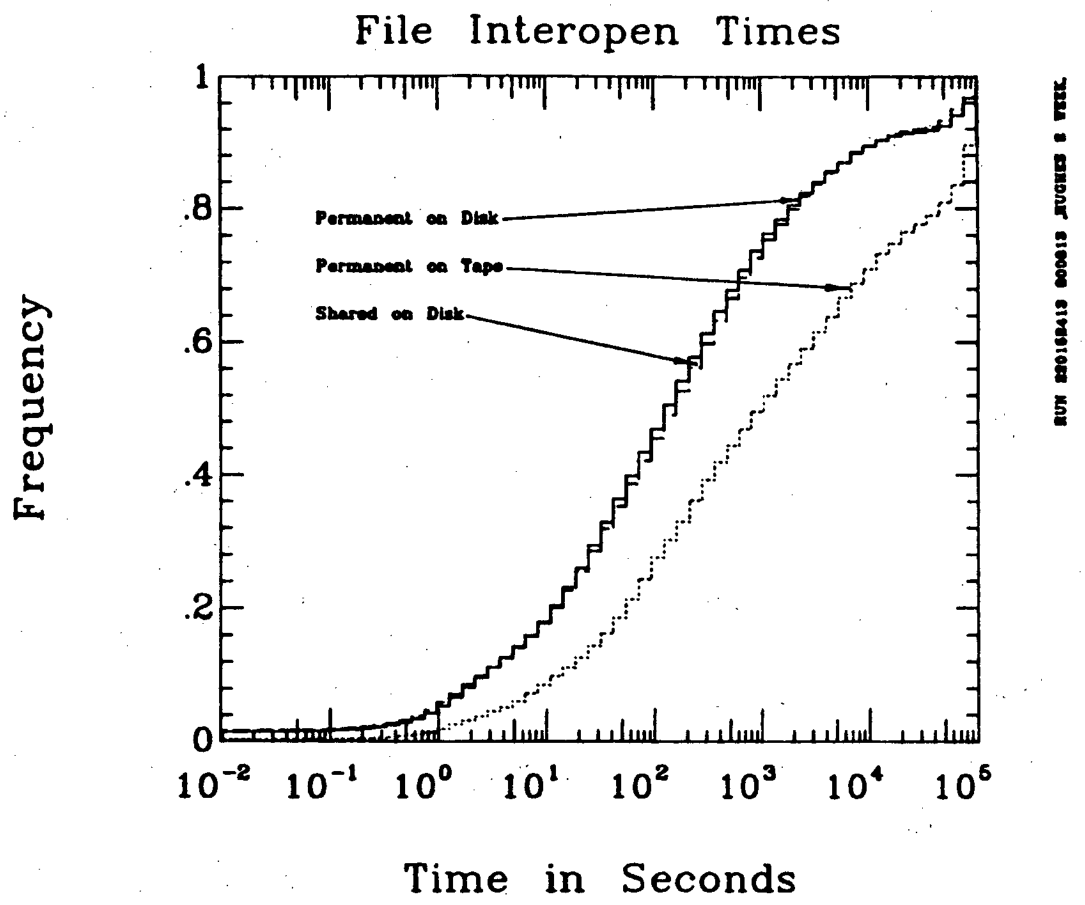

Fig 2.19. Distribution of interopen times for the Hughes trace. More information about the interopen times can be found in Table 2.19.

Table 2.19. File Interopen Times in Seconds (Hughes)

Type of File	Min	Max	Mean	Median	Std. Deviation
Perm. on Disk	0.	867000.	13200.	130.	52300.
Perm. on Tape	0.	652000.	35600.	1050.	83500.
Shared on Disk	0.	867000.	11900.	140.	46000.

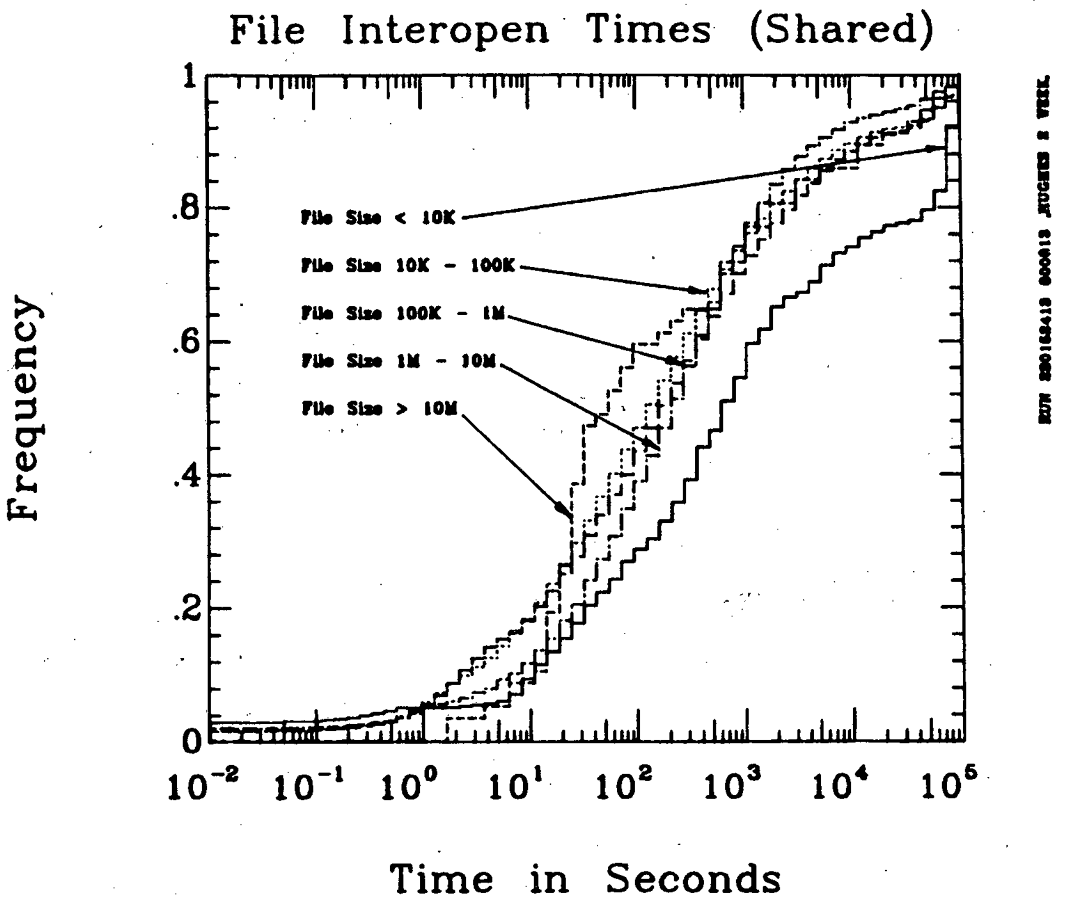

Fig 2.20. Distributions of interopen times for shared files. Hughes trace. Moments for the distributions are given in Table 2.20.

Table 2.20. File Interopen Times in Seconds (Hughes)

File Size	Min	Max	Mean	Median	Std. Deviation
< 10 KBytes	0.	460000.	33300.	580.	76400.
10 - 100 KBytes	0.	867000.	11800.	130.	46500.
100 - 1000 KBytes	0.	777000.	12500.	205.	47000.
1 - 10 MBytes	0.	853000.	7880.	210.	40800.
> 10 MBytes	2.	524000.	15100.	55.	70300.

Table VII. Comparison of Means and Medians

	SLAC		Hughes	
	Mean	Median	Mean	Median
Size of Files in Kilobytes	1200	240	215	24
Opens per File	76	22	23	9
I/O's per Open	88	9	29	2
% accessed per Open	113	29	53	25
Interopen Time in Sec.	8030	360	11900	140

Table VII. Fraction of Permanent File Activity due to Shared Files

	SLAC	Hughes
Number of Opens	0.27	0.20
Number of Reads	0.65	0.43
Number of Writes	0.32	0.38

(3) The distribution of the number of users per file is similar in both systems and it shows that the most common case of file sharing, by far, is when a file is shared by two users. This may simplify the problem of file migration in a distributed system.

CHAPTER 3

PARTITIONING OF CENTRALIZED COMPUTER SYSTEMS

Summary

This chapter presents heuristic methods for partitioning centralized computer systems (the resulting distributed systems will be used in later chapters). The heuristics are based on certain aspects of the systems workload such as file referencing, file sharing among users and requests for processor and I/O resources.

We test various partitioning heuristics on two real centralized systems (SLAC and Hughes) using trace-driven simulations. We conclude that a partition based on the number of user inversions (changes in the active user of a file) achieves the lowest overall traffic in the communications network of the resulting distributed system. The synthetic distributed computer system obtained by this procedure is used in the next two chapters to test file migration policies.

Introduction

In the last two chapters of this dissertation, trace-driven simulations are used to compare the performance of various algorithms for placing and migrating files in distributed systems. Traces of activity from real distributed systems are not available at the present time. Hence, it becomes necessary to create synthetic distributed systems with data collected from non-distributed, real computer installations. This is possible because the traces of activity available from centralized computer systems are traces of logical actions, and it can be argued that user behavior does not substantially depend on whether the system is centralized or distributed as long as performance remains acceptable. All that has to be done to obtain a synthetic distributed system is to partition the set of users of a centralized system and to assign the resulting groups of users to the nodes of the distributed system. In this chapter, a procedure to partition the users of a computer installation will be presented and the procedure will be applied to two real systems.

Section 3.1 introduces the distribution problem. Various heuristics to solve this problem are then presented in section 3.2. Section 3.3 is a detailed description of the partitioning procedure. In section 3.4, we describe the simulations used to evaluate the synthetic systems and the results obtained.

3.1. The Distribution Problem

In the last few years, many computer installations have gone through a process of system partitioning. For example, the Computer Center at the University of California at Berkeley was operating, in the academic year 1976-1977, a large machine (a CDC 6400) that supported most of the research and the instructional workloads. By the end of the 1981-1982 academic year, the CDC machine will no longer be in operation and the Computer Center will be operating, instead, almost twenty machines of different makes and models. Conceptually, the transition from the situation in 1977 to the present system can be described as the partitioning of the user community of the old CDC machine into a number of sub-communities and the assignment of the sub-communities to the new smaller machines. The actual procedure has been much more complicated because the global user community has grown extraordinarily in the last five years and because the Computer Center has introduced its new machines at different times during this period. Also, the partitioning has not always been legislated from above but rather prompted by the needs of users.

In distributing its users, the Computer Center must consider basic aspects of capacity planning like the number of terminals, the processing power and the storage capacity of each individual machine. The university environment puts additional constraints on the assignment of users to machines. For example, the members of a class are usually assigned to the same machine for reasons of basic fairness (same down times, same throughput). Another constraint is that some researchers must be assigned to particular machines in order to use specific hardware or software capabilities. Finally, the machines operated by the Computer Center are for the moment linked by a network of extremely low bandwidth and they must be considered as independent for most purposes. This implies that users needing access to each other's files must reside on the same machine. The methods used by the Computer Center for assigning users to machines have been ad hoc and based on past experience and power struggles among the sectors of the user community.

System partitioning may be needed for other purposes. In our case, we want to partition some real centralized systems to obtain synthetic distributed systems. We need the distributed systems to run trace-driven simulations of migration algorithms. We will assume that we do not have administrative constraints of the type that the Computer Center has to deal with. This will enable us to use a more systematic approach to system partitioning.

3.1.1. Statement of the Problem

We can now state the distribution problem. Let C be a centralized computer system defined by the set U of its users and by a trace of its activity during the time period [0,T]. The trace contains a record for each job execution and for each file creation, deletion, open and close during [0,T].

Let D be a distributed system built around a fully connected communications subsystem. The distribution problem consists of finding a partition of the user community U such that, when its subsets (classes) are assigned to the nodes of D, the aggregate volume of traffic in the communications network of the system is minimized.

Without additional constraints, the problem is not well defined. In the first place, we must specify how we are going to manage the files in the system. This is important in that it will determine what is the aggregate traffic in the system. Our approach will be to use two different policies (to be defined later) and to optimize the partition with respect to both policies. This will give us some assurance that the partitioning procedure does not depend on a particular file management policy.

As stated, the distribution problem has the degenerate and trivial solution of assigning all the users to the same node of the network. This has the effect of reducing the traffic to zero. In order to obtain non-trivial solutions, we will require that the partition have a given number of nodes and that each node contain at least one user. This way, we will be able to obtain distributed computer systems with any number of nodes.

In a real distribution case like the one involving the UCB Computer Center, the computing power and the storage capacity of the machines at the nodes of the distributed system are given. In our case, there is no clear indication about how to choose these values. A canonical assumption is that all the machines at the nodes of the distributed system have similar characteristics. We will make this assumption and therefore require that the CPU and IO requirements of the groups of users be as homogeneous as possible. This will achieve a long-term load balancing in the system.

3.2. Solution of the Distribution Problem

We now present a procedure for partitioning a centralized system. We recall from our discussion of the File Assignment Problem in Chapter 1 that we are mostly interested in the management of shared files. We argued that temporary files and permanent single-user files should be stored at the node where they are used and that system files should be replicated at every node of the distributed systems. Under these conditions, only shared files will generate internode traffic. Since we want to create distributed computer systems that minimize this traffic, we would like to cluster together the users that share roughly the same set of files. A partition of the set of users that eliminates completely the traffic among components is not generally achievable because of the overlap in the use of shared files by different users. Therefore, we must resort to a clustering procedure based on the statistical properties of the users and of the shared files.

We use a heuristic procedure based on the clustering of users according to the way in which they share files. We do not use an optimal procedure for two reasons:

(1) Sensitivity to the migration policy. In order to use an optimal procedure, we would have to define a cost function. In our case, the cost function would be some combination of traffic, delay and storage costs. Unfortunately, the amounts of traffic, delay and storage needed by any

given partition depend on the choice of migration policy for the system. A partition that would optimize such a cost function would be explicitly linked to a particular migration policy. On the other hand, our purpose for partitioning systems is to study the performance of migration policies. It would be very difficult to compare migration policies on the bases of a partition that is optimal for one of the policies to start with.

(2) Computational complexity. In the first place, if there are N users and we want to partition the system K ways, the number of different partitions is on the order of K^N. In addition, computing the cost of one of the partitions involves running a trace driven simulation. No matter how efficient is the searching procedure for the optimum, it is obvious that the whole process will be very expensive.

Our heuristic procedure is much less expensive in terms of computation and it can be carried out independently of any particular migration policy, even though some migration policies are used to determine the goodness of the partition.

The procedure has three logical steps:

(1) Define a number of measures of proximity P_{ij} for each pair of users $(i,j) \varepsilon U \times U$. Each one of the measures must reflect how much two different users access the files that they share. Users that share many files or that access heavily the files that they share should have high proximities. Based on each measure, obtain a partition of U by some standard clustering algorithm, and create the synthetic distributed system by assigning partition classes to nodes. The number of nodes is given in each case and the assignment will try to balance the processing and I/O requirements of the users in each node (a precise algorithm will

be described later).

(2) Measure the goodness of the partitions. This is done by running a trace-driven simulation of each of the systems (two, four, eight, sixteen nodes) using two different migration algorithms and collecting measures of the induced traffic.

(3) Choose the proximity measure (and therefore the partition) that results in the lowest traffic.

The first of these steps, the partitioning procedure, is described below.

3.3. The Partitioning Procedure

The partitioning procedure is best described in three steps:

(1) Various measures of proximity between users are defined.

(2) The set of users is partitioned according to their proximities, using a standard clustering algorithm. The result is a set of components.

(3) One or more of the resulting components are assigned to each node of the synthetic distributed system.

Each step is now described in more detail.

3.3.1. Definition of Proximity

In this section we define seven measures of proximity P_{ij} between users of a computer system. All measures are defined for a period of time [0,T]. For a given computer system, proximities can be arranged as a symmetric matrix of size $n \times n$, where n is the cardinality of U.

3.3.1.1. Random

A random partitioning of the system can be obtained by defining a random measure of proximity between each pair of users. One way of achieving this is to assign to each pair of users a number between 0 and 1, drawn from a standard uniform distribution.

$$P_{ij} \sim U(0,1)$$

A random partition of the system cannot be expected to produce very good results in terms of reducing traffic. Indeed, we will use this partition as a yardstick to measure how much the other algorithms can reduce the traffic below what is obtained with a totally random partition.

3.3.1.2. User Group

In many computer installations, users are assigned to user groups. User groups usually follow the structure of the organization so that users in the same group belong to the same project, the same class or the same administrative unit. In many cases, users in the same group share data files and libraries of procedures that have been developed for their particular needs. Belonging to the same user group is an indirect measure of the degree of sharing of the files. This particular measure can only be used with the SLAC data because the Hughes installation does not maintain an assignment of users to groups. Since users belong to exactly one user group, the user groups generate a total partition of the set of users that can be obtained directly from a list of the users and their user groups. However, in ordcr to make this partitioning method more similar to the others that we will introduce, we define a proximity P_{ij} as follows:

$$P_{ij} = \begin{cases} 1 \; \textit{if users i and j belong to the same User Group} \\ 0 \; \textit{otherwise} \end{cases}$$

3.3.1.3. Number of Shared Files

This measure of proximity is based on the number of files shared by each pair of users during the interval [0,T].

Let $H_i = \{h_i^1, h_i^2, \cdots, h_i^{n_i}\}$ be the set of files accessed by user i belonging to the set of all shared files in the system. Let $H_j = \{h_j^1, h_j^2, \cdots, h_j^{n_j}\}$ be the set of files accessed by user j from the set of all shared files in the system. Furthermore, let $n_{ij} = |H_i \cap H_j|$. The proximity measure is:

$$P_{ij} = n_{ij}$$

3.3.1.4. Shared Space

This measure of proximity, closely related to the previous one, takes into account the size of the files involved. Let us define the function $s(h)$ as the function which returns the size of file h in bytes. In addition, let $H_{ij} = H_i \cap H_j = \{h_{ij}^1, h_{ij}^2, \cdots, h_i^{n_{ij}}\}$ be the set of the files shared by users i and j and $n_{ij} = |H_{ij}|$ the cardinality of this set. We define P_{ij} as follows:

$$P_{ij} = \sum_{k=1}^{n_{ij}} s(h_{ij}^k)$$

This measure should perform better than the previous one when used with migration policies that move entire files in the event of remote accesses since it takes into consideration not only the number of files shared but also their sizes.

3.3.1.5. Shared Transfer

This proximity measure is based on the amount of information accessed in the shared files, i.e., the sum of all the bytes transferred to and from a file by all the jobs run by a user. This measure of proximity, under a slightly different form, is proposed in [Buc79]. In essence, we consider the amount of information transferred to and from a file that two users have in common. Let a_i^k be the amount of information in file k accessed by user i during the time interval [0,T]. Let us define:

$$P_{ij} = \sum_{k=1}^{n_{ij}} [\, min(a_i^k, a_j^k)]$$

The reason for using the min function as opposed, say, to the average or the sum of the transfers is that we expect the migration algorithms to be able to detect the best position for the file (in this case the user with the highest access rate). If this is indeed achieved, it is the amount transferred by the other user (the min of the two transfers) that actually originates traffic.

3.3.1.6. Number of Inversions

Let h be a shared file. h undergoes an *inversion* when it is opened successively by two different users. It is easy to see that, if h is shared by m users during the time interval [0,T] and is opened n times, then h undergoes at least $m - 1$ inversions and no more than $n - 1$ inversions. Let $I_{ij}(h)$ be the number of times that h is opened first by user i and then by user j with no intervening opens. The measure of proximity based on the number of inversions is:

$$P_{ij} = \sum_{k=1}^{n_{ij}} [I_{ij}(h_{ij}^k) + I_{ji}(h_{ij}^k)]$$

This measure of proximity is specially useful when the migration policy moves the entire file from one user to the next. In that case, it tends to keep together users that would cause a large number of transfers of the file.

3.3.1.7. Inversions Space

This measure is very similar to the previous one except for the fact that inversions are weighted by the size of the file.

$$P_{ij} = \sum_{k=1}^{n_{ij}} s\,(h_{ij}^{k}) \times [I_{ij}(h_{ij}^{k}) + I_{ji}(h_{ij}^{k})]$$

3.3.2. Clustering Algorithm

At this point, we could use any of the many existing clustering algorithms [Har75] to partition the set of users according to any of the proximity measures. We have chosen the one described by Zadeh in [Zad71] because it allows to change easily the target number of components of the partition.

This clustering method, based on the similarity matrix is fully described in [Zad71] and we describe here the main steps of the algorithm only for completeness.

The first step is to turn the proximity matrix, made of all the proximities among users, into a similarity matrix. The reason for doing this is that a proximity measure (or relation) is not transitive and therefore is not an equivalence relation. However, it is easy to derive a similarity relation from a proximity relation by calculating the transitive closure of the proximity matrix. A similarity relation is an equivalence relation (it is reflexive, symmetrical and transitive) and hence is suitable to effect a partition on a set, in this case the set of users.

A similarity matrix actually defines a family of partitions. To obtain a member of the family of partitions, we need to specify one more parameter: a similarity threshold. If the similarity between two users is greater than the threshold, then they belong to the same component of the partition. Otherwise they belong to different components. The method works as follows.

(1) Pick any value from the matrix. The choice of this value determines the number of components in the partition. A small value will produce a few, large components. A large values will yield many small com-

ponents.

(2) Use this value as a critical value, and set all elements less than it to zero and all greater than or equal to it to one.

(3) Interpret the matrix as the connectivity matrix of a graph. The cliques of the graph are the partitions (similarity classes) of the set of users.

At the end of this procedure, we might have obtained more components than there are nodes in the distributed system. Next, one has to assign components of the partition to nodes of the distributed system.

3.3.3. Assignment of Components to Nodes

The goal of this procedure is to assign user subsets to the N nodes of the distributed system in such a way that the total processing power and I/O bandwidth of all nodes be approximately the same fraction of the total processing and I/O requirements R. In practice, for the two real systems that we have partitioned, it turns out that assigning nodes based on the processing needs results in systems that are also reasonably well balanced in the I/O requirements. To achieve this goal, we use the following strategy:

(1) By choosing the appropriate critical value, obtain a partition with a larger number of components than the number of nodes of the system, such that no single partition has requirements exceeding $\frac{R}{N}$. Obtaining this value is an iterative process. An initial critical values is chosen arbitrarily. The partition is obtained, the processing and I/O requirements are computed, and the results sorted. If any of the component requirements exceeds $\frac{R}{N}$, a new (higher) critical value is picked and the procedure is repeated. If the requirements of the larger component are not within the order of magnitude of the target, a smaller critical value

is chosen and the procedure is repeated. A binary search of the space of critical values eventually yields an adequate partition.

(2) Assume that a good partition has been obtained. The components of the partition are then sorted by processing and I/O requirements and the top N components are chosen as seeds for the assignments. The remaining components are assigned, in decreasing order of resource requirements, to the node that can accommodate them without getting more than $\frac{R}{N}$ of the total system resources. In practice, the procedure has worked very well. It must be noted that the partitioning procedure usually produces scores of very small components and these are very useful towards the end of the assignment procedure to even out the processing and I/O requirements of the nodes.

The assignment of components to nodes is a problem that could be formulated as an integer program and hence could be solved by standard mathematical programming methods. Since the solution of the integer program can be computationally expensive and the assignment is not a crucial part of our study, we have used the heuristic procedure described above.

3.4. Choice of a Partitioning Strategy

The next step in our choice of a partitioning strategy is to determine which one of the seven proximity measures yields the system with the lowest average communication traffic. As was mentioned earlier, we determine that by running trace-driven simulations of the synthetic distributed systems. We perform two sets of experiments, using two different file migration policies. The first policy maintains a static assignment of files to nodes. The second one dynamically assigns the file to the node of the current user (these policies are more precisely defined in the next section).

Conceptually, each partitioning strategy could have been tested using more than two policies. Unfortunately, this would require a number of experiments that is beyond our capabilities. Therefore, we have chosen two single-copy policies that are easy to implement and different enough so that a partitioning strategy performing well under both policies can be considered a good strategy independently of the migration policy.

3.4.1. The Migration Algorithms

The first of the two algorithms is *RIO* (Remote I/O). RIO is a static algorithm and it can be described by the two following rules:

(1) A file is placed at the node where it is created. If the trace does not contain the creation record for the file, then the file is placed at the node where it is first used during the span of the trace.

(2) Whenever a reference is made to the file from a remote node, the required records are transferred to or from the remote node. The records are not cached at the remote node.

The second algorithm is *MRU* (Most Recently Used). MRU maintains a single copy of the file at the node where it has been most recently used. When using MRU, users located at different nodes are not allowed to access simultaneously a file.

The next section presents the results of the simulations using these migration policies.

3.4.2. Experimental Results

The partitioning procedure that we have described in the two previous sections have been performed on the SLAC and the Hughes Aircraft systems.

For the SLAC trace, all the experiments have been repeated twice. The first time, the entire trace has been used to determine the system partition. Then the whole trace has been used to run the trace-driven simulations and to measure the traffic. The second time, only the first half of the trace has been used to determine the partition. Then the second half has served as input to the simulation. Our objective in performing this second experiment is to see whether the proximity measures that we had chosen would be able to predict the utilization of the files in the future. The experimental results show that there is little difference in the ranking of proximity measures in both cases.

3.4.2.1. SLAC Trace

We look first at the results obtained by running the RIO migration algorithm (Figs 3.1, 3.2, 3.3, 3.4).

All partitioning strategies tested perform better than random partitioning in terms of average traffic. For all the strategies, the traffic function has a decreasing slope as the number of nodes increases. The partitioning based on the user group is the best, followed by the ones based on the inversion space and the number of inversions.

When the experiment is repeated on the second half of the trace (Fig 3.2), the results change very little. The average traffic is actually lower. This is not due to the partitioning procedure, but rather to the different characteristics of the data in the second half of the trace. The User Group proximity becomes even better than the other policies and the Shared Space policy now ranks in third place, indicating that it is a very robust measure of proximity.

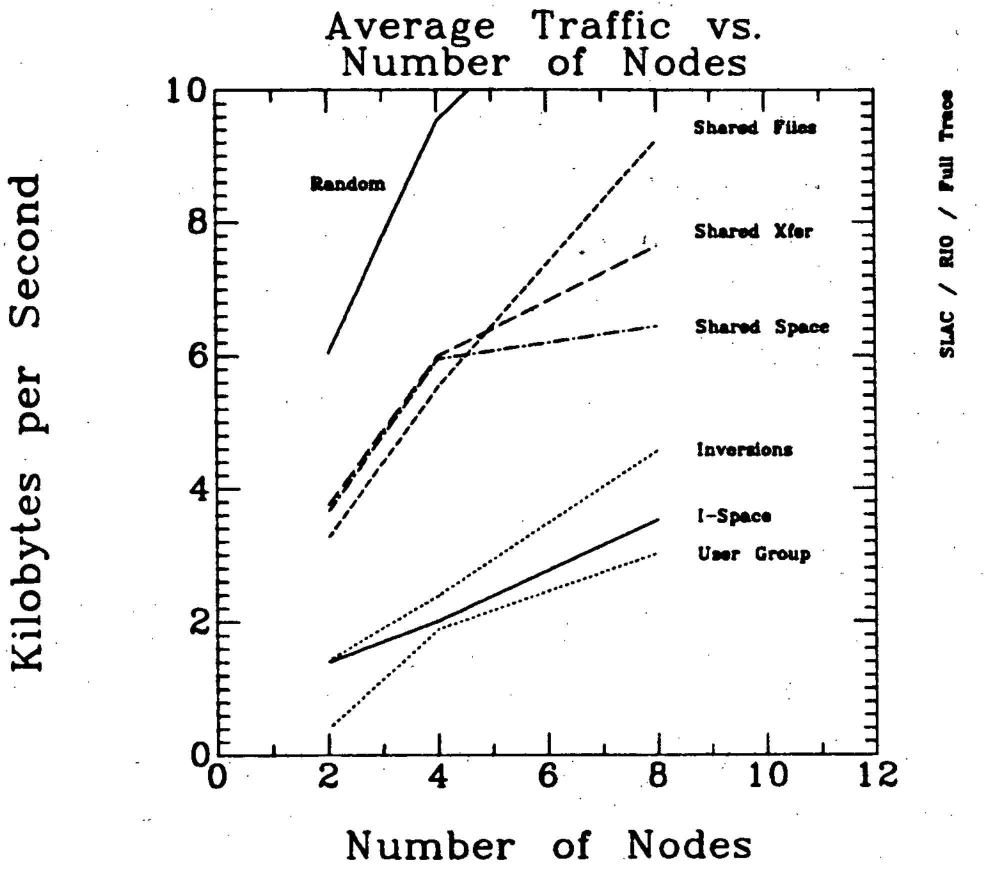

Fig 3.1. Average traffic as a function of the number of nodes in the system (SLAC trace). This traffic is generated by the RIO (Record I/O) migration algorithm, which keeps a single copy of the file at the creation node. Both partitioning and simulation use the entire trace.

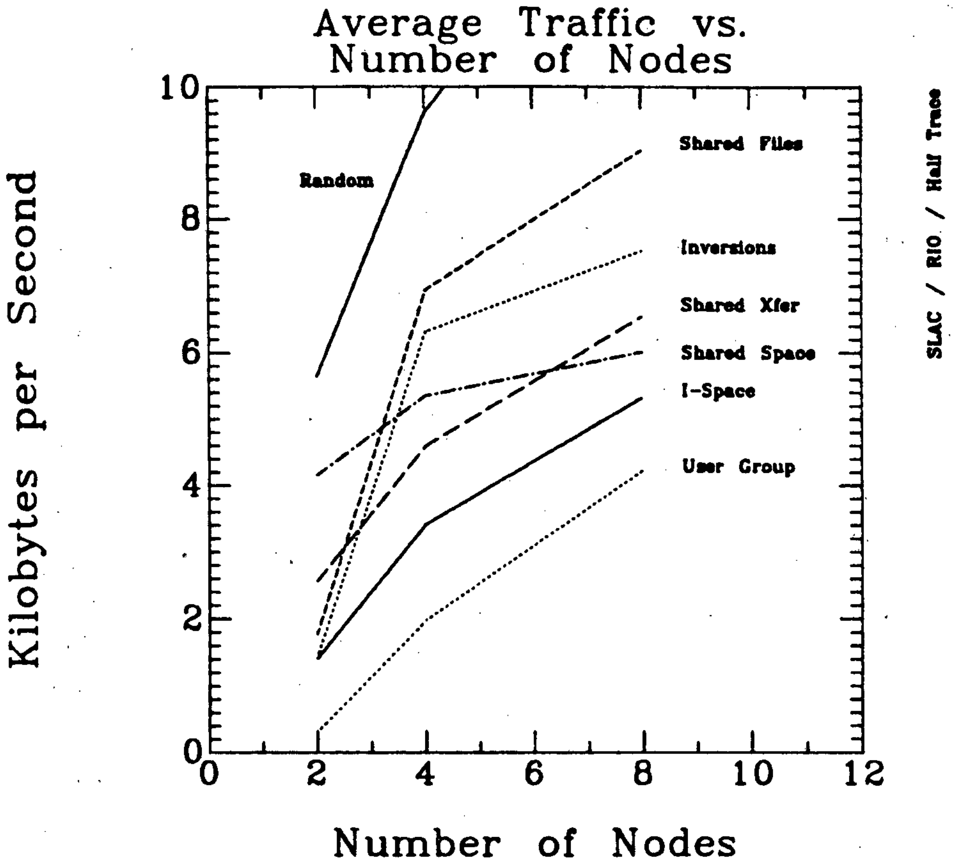

Fig 3.2. The same experiment of Fig 3.1, but partitioning is done on the first half of the data and it is tested on the second half.

Average values are useful to determine which partitioning procedure generates the least total traffic in a particular time period. However, worst case design rules are often used. In this case, we are interested in peak traffic values as generated by the various synthetic systems. Fig 3.3 shows peak hourly traffic for the interval [0,T], when the full trace is used for partitioning and testing. The ranking of the policies is almost the same. I-Space is better that the user group proximity, indicating a smoother demand on

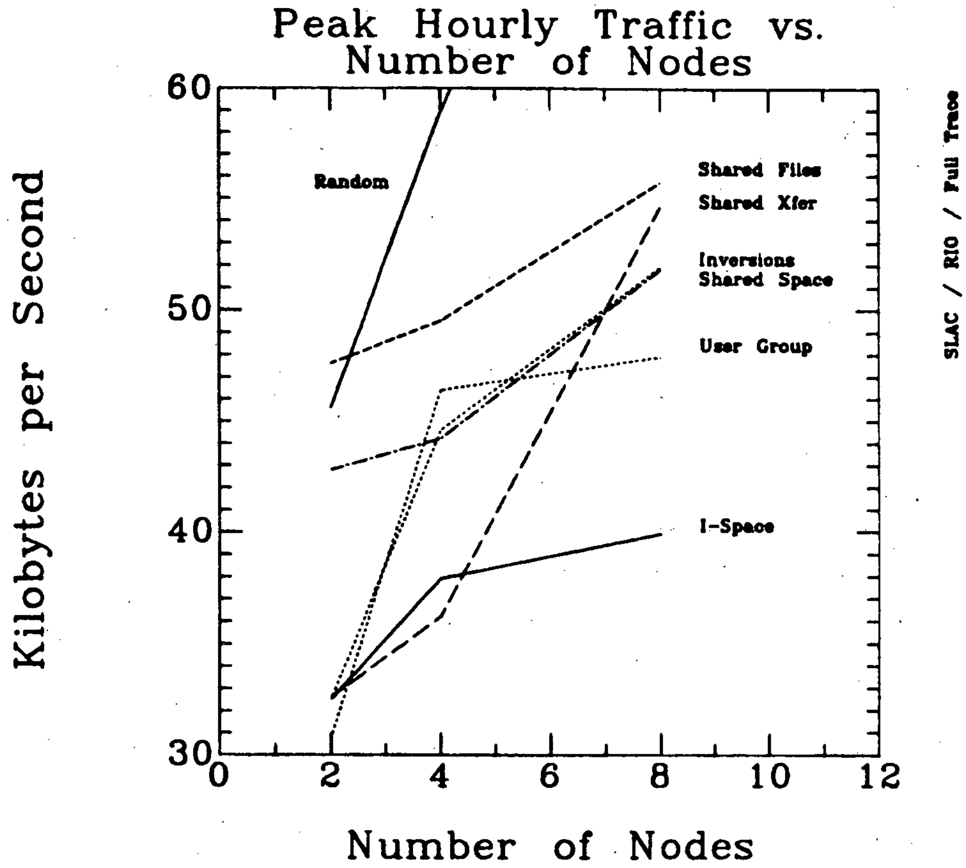

Fig 3.3. The maximum hourly traffic generated by RIO (Record I/O) for the SLAC, two week trace. The full trace is used in the experiment.

the communications system. The same measurements of Figure 3.3 are shown in Figure 3.4 when the first half of the trace is used for partitioning and the other half for testing.

To make sure that the results are not too sensitive to the migration algorithm used so far (RIO), we repeat most of the experiments using MRU to manage the files in the system. The experiments, again, are conducted both for the full trace(Figure 3.5) and for the second half only (Figure 3.6). Figure

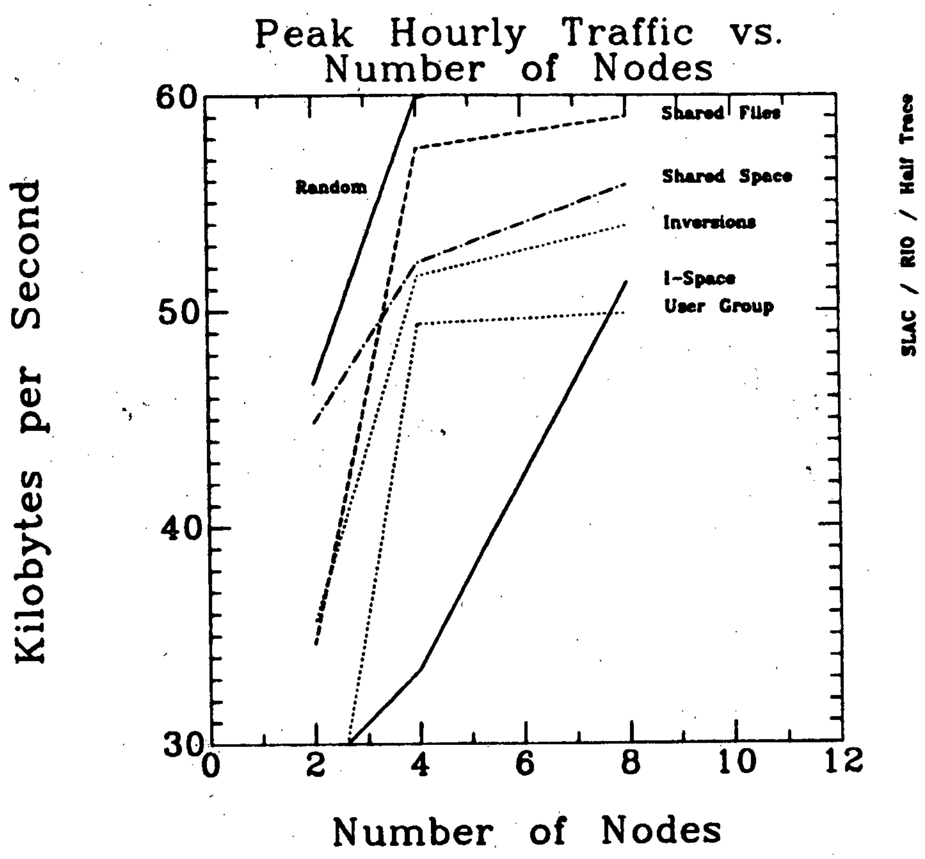

Fig 3.4. The same experiment of Fig. 3.3 using half of the trace for partitioning and the other half for testing.

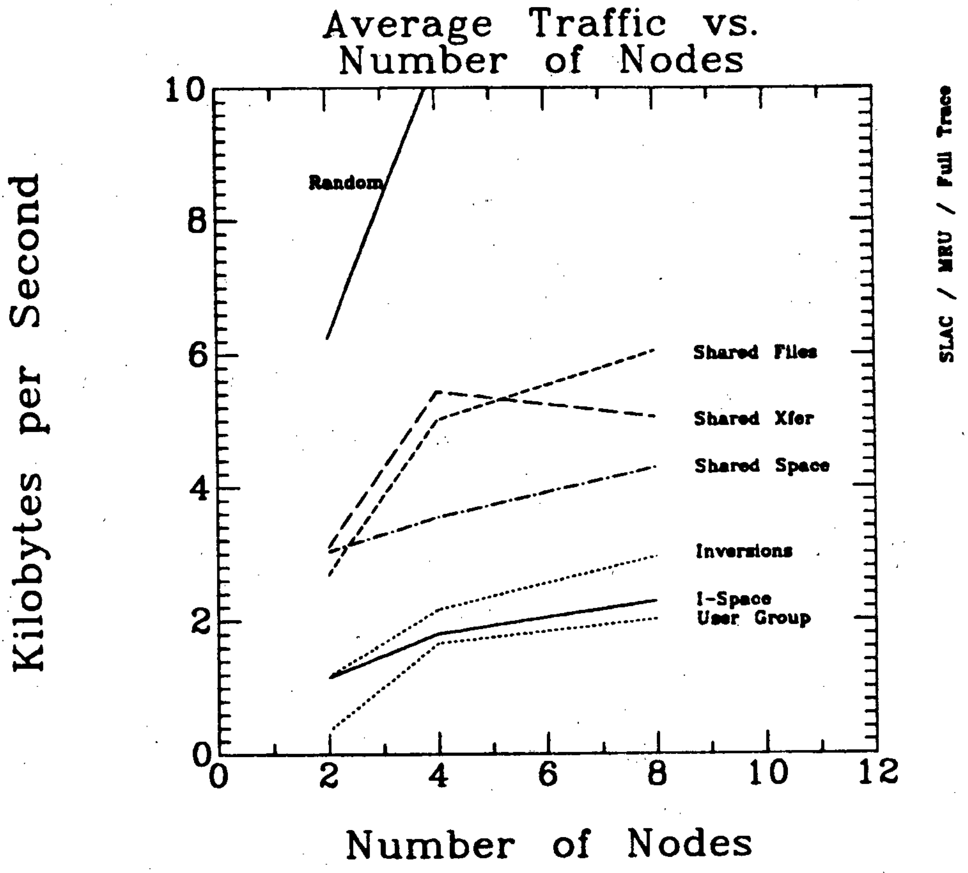

Fig 3.5. Average traffic versus space time when using the MRU (Most Recently Used) migration policy. Both partitioning and testing done on the full SLAC trace.

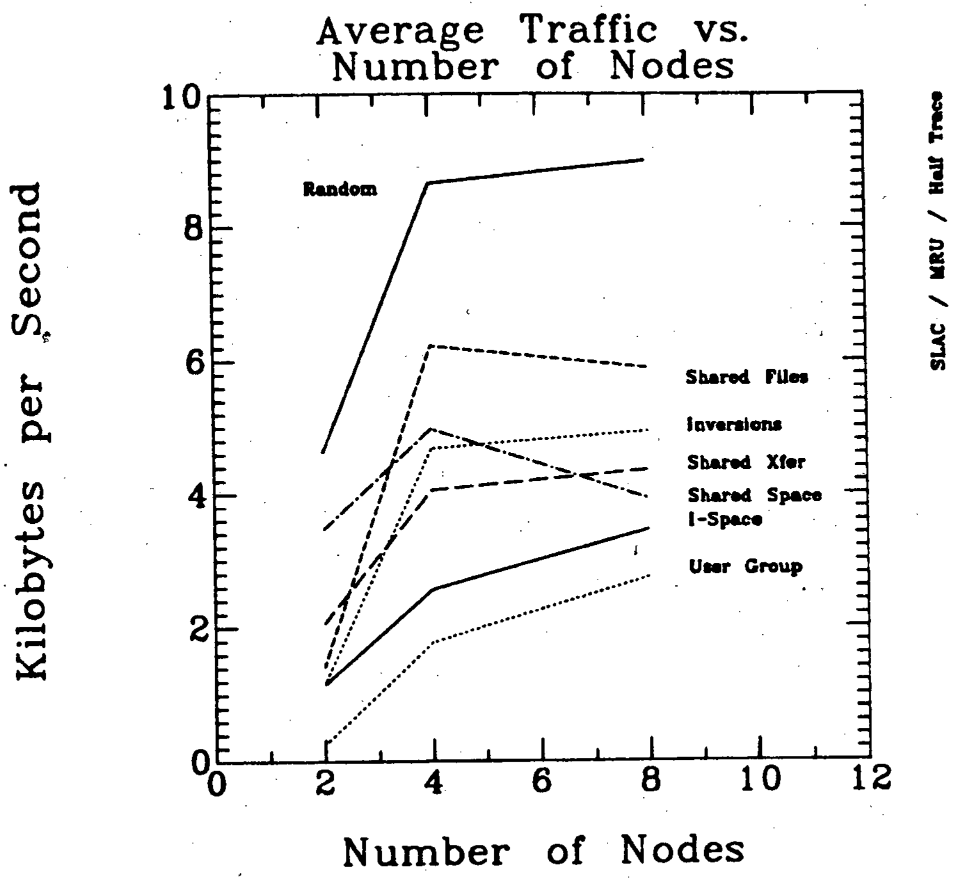

Fig 3.6. MRU policy on half of the SLAC trace.

3.7 shows the peak hourly traffic when using the MRU policy and the whole trace both in partitioning the system and to run the simulation.

These results confirm that User Group and Inversion Space are the two best criteria for partitioning the SLAC user community.

3.4.2.2. Hughes Aircraft Trace

The partitioning procedure has been repeated for the Hughes trace. There are a few differences in the results:

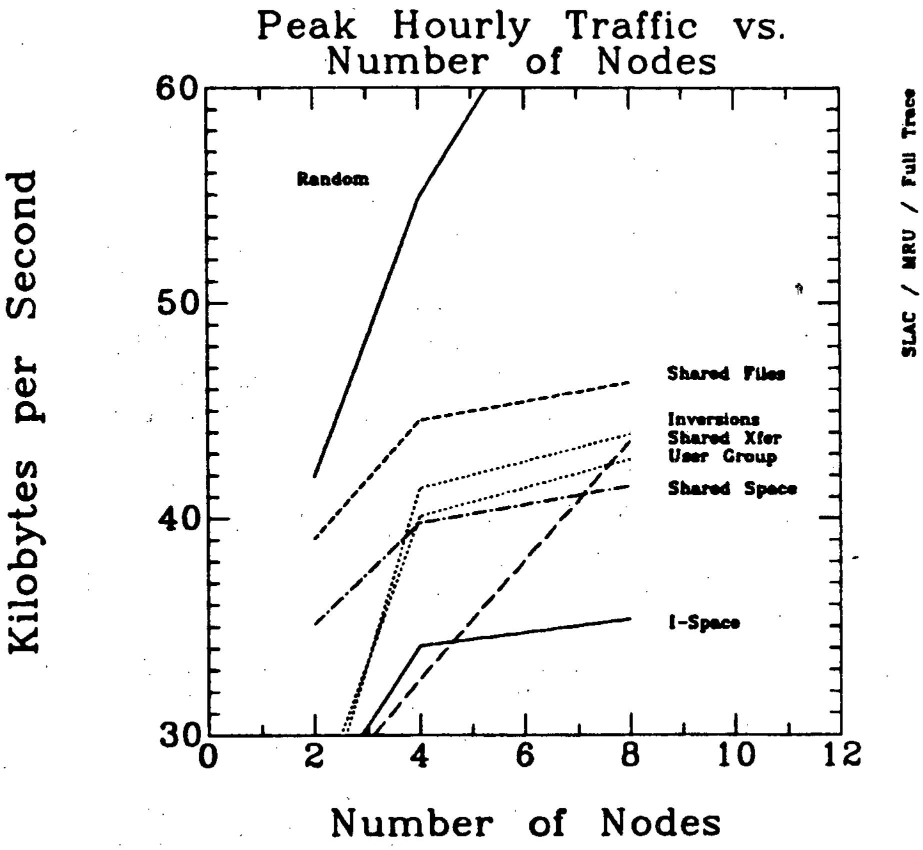

Fig 3.7. Maximum hourly traffic vs. number of nodes for the MRU migration policy on the full SLAC trace.

(1) The experiments based on partitioning on the first half of the trace and testing on the second half have not been done.

(2) The characteristics of the Hughes systems are such that most of the proximity measures achieve excellent partitions.

(3) The User Group measure has not been used because it is not possible to extract this information from the trace.

Figure 3.8 and Figure 3.9 correspond to the simulations using RIO. As we mentioned before, the Hughes systems can be very easily partitioned. Figure 3.8 shows the average traffic and Figure 3.9 the hourly peak traffic. In both cases, the entire trace is used to obtain the partition and to measure the traffic.

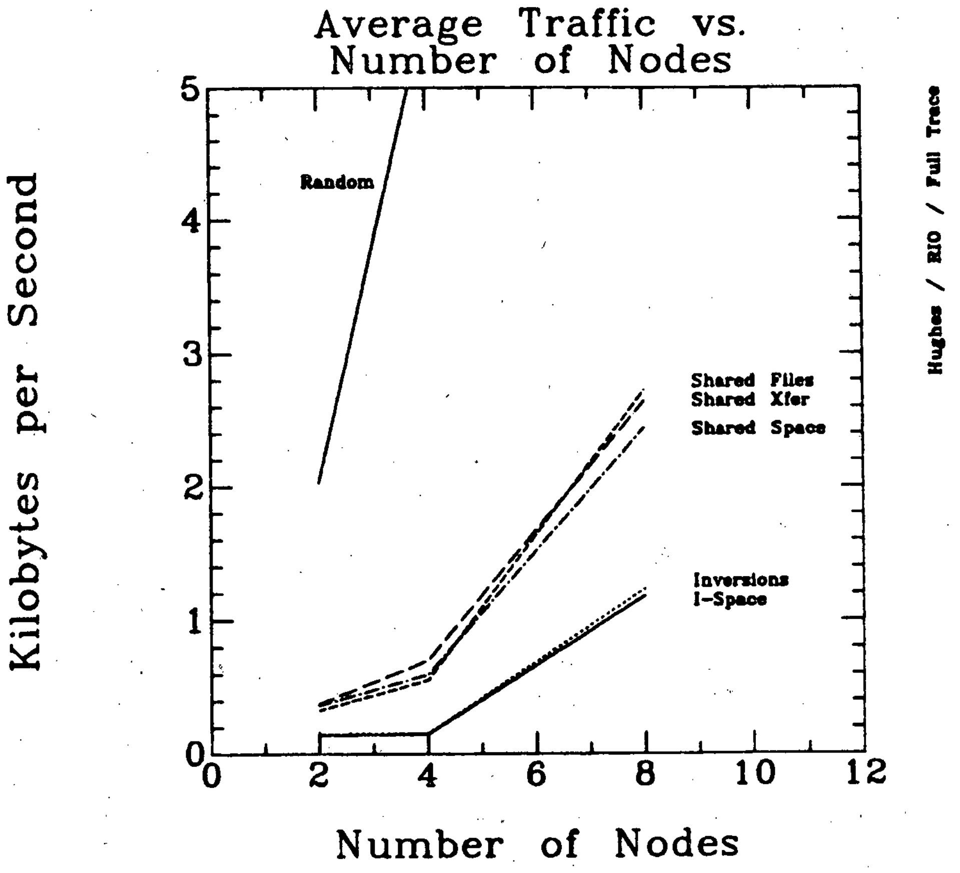

Fig 3.8. Average traffic as a function of the number of nodes in the system (Hughes trace). This traffic is generated by the RIO (Record I/O) migration algorithm, which keeps a single copy of the file at the creation node. Both partitioning and simulation use the entire trace.

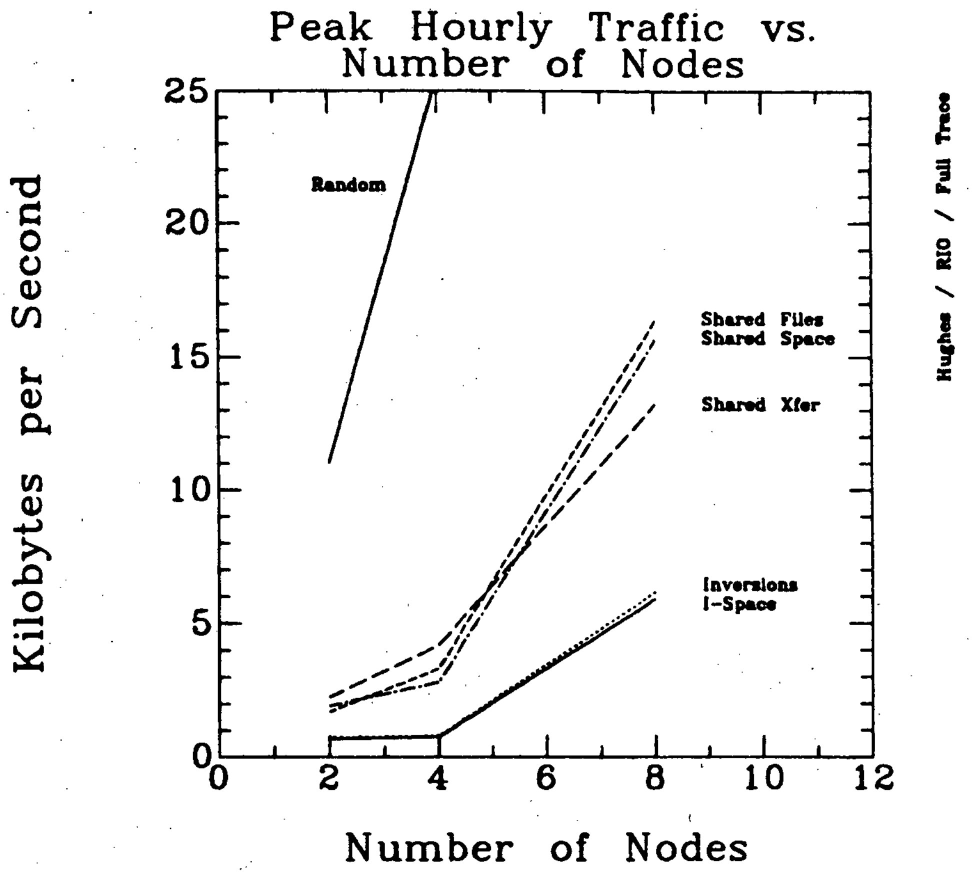

Fig 3.9. The maximum hourly traffic generated by RIO (Record I/O) for the Hughes, two week trace. The full trace is used in the experiment.

Figure 3.10 shows the traffic generated when the MRU policy is used with the Hughes trace.

3.5. Conclusion

In this chapter, we have developed a heuristic method for distributing centralized systems. The procedure begins by defining a measure of proximity between each pair of users. Using this proximity measure, the system is

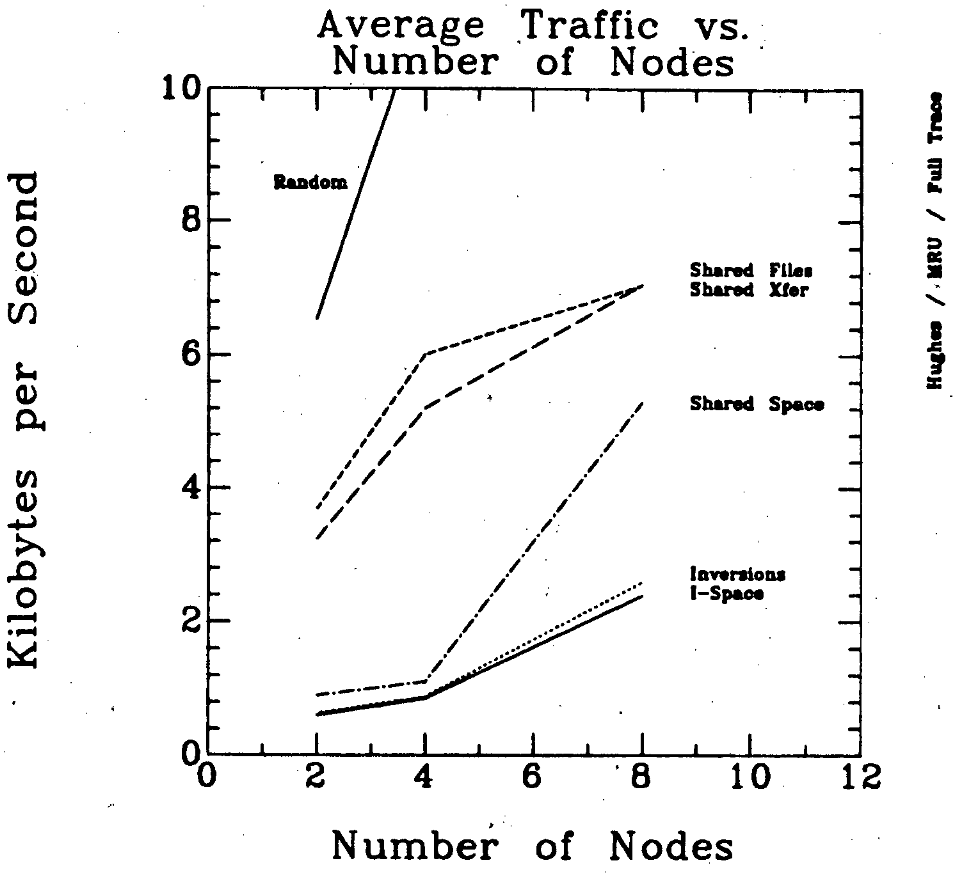

Fig 3.10. Average traffic versus space time when using the MRU (Most Recently Used) migration policy. Both partitioning and testing done on the full Hughes trace.

partitioned and the resulting classes of users are assigned to the nodes of a synthetic distributed system. The goodness of the distribution is then tested by running trace-driven simulations of the distributed system. Two sets of simulations have been performed, using two different file management policies. The results have been consistent.

We have performed the distribution procedure on two systems: SLAC and Hughes. The partitioning of the SLAC user community has produced subsets with a great amount of overlapping among them. We have also observed that the partition based on the user group of each user yields a very good partition in terms of generated traffic. Of the proximity matrices based on measurements of the data, the best two are the one based on number of inversions and the one based on inversion space. Because it is somewhat simpler to obtain and it does not rely on the size of the files provided by the trace, we choose the proximity measure based on the number of inversions for obtaining the distributed systems to be used in later chapters.

When we have applied the same methods to the user community of the Hughes system, we have obtained partitions that generate much less traffic among components. This is probably due to the fact that the user community is itself strongly partitioned into projects so that little sharing exists between projects. In any case, the proximity measure based on the number on inversions is again the one performing best and we choose it for obtaining the synthetic distributed computer systems that we need for the simulations of the next two chapters.

CHAPTER 4

SINGLE-COPY MIGRATION POLICIES

Summary

In this chapter we present a number of policies for migrating files in distributed computer systems. We only consider policies that maintain a single copy of each file in the system.

Based on a simple model of file sharing, we develop a migration policy that is optimal in the sense of minimizing the global network traffic. This policy is then compared to a number of heuristic algorithms using trace-driven simulations.

Introduction

Both centralized and distributed computer systems allow users to share information. In centralized systems, sharing is usually based on the access to some common storage area. This common area can be in main memory or on a secondary storage device. Access to shared memory can be inexpensive because the information need not be copied, in principle, from one user's space to the other user's space. A remapping of the physical storage region is all that is needed.

When dealing with distributed systems, sharing information always means transferring it from one system to another (we consider remote I/O's as transfers of data). Since transferring data can be an expensive operation and it can be done in many different ways, a number of decisions need to be made in order to implement sharing of information in a distributed system.

Let us assume that two users located at different nodes of a computer network want to share the information contained in a file. Should the file be permanently placed with one of the users? If so, with which one? What if there are more than two users? Should the file be moved back and forth between nodes as it is accessed? Should copies of the file be made and distributed to all users of the file? In this case, what should be done with the copies when one of the users updates the contents of the file? Or should the users be moved between systems? The decisions made about placement and transmission of shared files may have quite an impact on the overall performance of a distributed system. How to make these decisions is the subject of the remaining chapters of this thesis.

We will start by considering algorithms that manage only one copy of each file. These so-called single-copy policies are easier to implement and easier to model than the policies that create multiple copies of files. As we will show, single-copy policies do not provide, in general, the best results in terms of system performance. However, they are worth studying for a number of reasons: they can be implemented with existing file handling mechanisms, they are easier to optimize and they can outperform the multiple-copy policies under certain conditions, eg. heavy updating or expensive storage.

Section 1 is a brief review of the mechanisms that can be used for accessing remote files. Section 2 presents a simplified model of reference for shared files. The model is powerful enough to evaluate the class of policies that we consider in this chapter. Section 3 uses the model developed in the previous section to find an optimal policy for file migration under some rather general cost functions. Section 4 introduces sub-optimal policies for migration of shared files and uses the model of section 2 to predict their cost. Finally, section 5 uses trace-driven simulation to compare the behavior of all the policies in the synthetic distributed systems generated in the previous chapter.

4.1. Access to Remote Information

There are basically two ways for a user to access remote information:

(1) Run a task on the machine where the data is.

(2) Transfer the data to the user's local machine.

The first alternative can be implemented as remote login or remote execution. Remote login [Dav77, Day80, Tan81] provides interactive access to a remote machine. The user can run his programs in the remote machine and has access to all the data stored in that machine. This arrangement can result in low transmission requirements, especially if the result of the computation is a small amount of data that can be displayed on a terminal or printed on a hardcopy device. If the amount of data generated by the execution of the program is not so small or if the necessary data is actually resident in more than one machine, then remote login is not powerful enough. Some networks also provide remote execution [Hwa80]. This is a less interactive mechanism, that triggers execution of a task in the remote computer. It is usually combined with some transfer of data, as with remote mail programs. The topic of remote execution is an interesting one but beyond the scope of this research. It involves the study of load balancing [Bok79, Chu80, Mit79] and we believe that the issues of file placement and migration can be studied independently of process migration. Also, process migration is a less general approach, usually limited to homogeneous networks, while file transfer has been used in heterogeneous networks for quite some time.

From now on, we will assume that users have been assigned by a system administrator to specific machines and that all their tasks must run on their local machine. When a user needs to access a remote file, two mechanisms can be used:

(1) Transmit the records needed by the user, leaving the file at its current location.

(2) Move the entire file to the user's location.

The first method of access is known as *remote file access* or *remote open*. It has been implemented in a number of systems. Some of these systems have machines with almost no storage and all the files are located in a file server. The Cambridge Ring [Dio80] is a good example of such a system. In other cases, remote file access is the mechanism of choice for the access of remote information because it eliminates the problems of updating directories that refer to the files being moved [Pec81, Coa81]. As far as performance is concerned, remote file access has pros and cons. On the plus side, it does not generated unwanted traffic in that the only information that is transmitted has been explicitly requested by a user. Since the file is not moved, there is no need to update directory information. It has potential for optimization: In the frequent case of sequential access, special protocols can be used that do not require one request message per file record. Finally, records can be cached at the local machine, so that repeated accesses to the same record do not require extra transmissions. On the negative side, transmission can be inefficient if the records are too small. In the worst case, multiple messages may be needed for each file record that is to be transmitted, increasing the delay involved in remote access. Finally, remote access disregards the locality of user reference that may exist in a system. This can generate repeated remote opens from the same user to the same file, generating unneeded traffic.

The other way of moving the data to the user is to move the file permanently to the node where it is needed. This action is called a *file transfer*. File transfer protocols are available in most computer networks [Gie78, Hui81, Hwa80]. The automatic transfer of files can be a rather involved operation depending on the way that network directories are organized. On the other hand, file transfer is a potentially efficient mechanism for various reasons. First, only one message from the requesting site is needed to transmit the entire file. This is in contrast with remote file access, where one message may be needed for each file record. Secondly, transfers of large amounts of information tend to use the available bandwidth more efficiently, by eliminating packet fragmentation, for example. Finally, if the user is going to access the file repeatedly, only one file transfer is needed, as opposed to multiple remote accesses.

In the remaining sections of this chapter, we will investigate how the mechanisms of remote open and file transfer can be combined in order to optimize the operating cost of a distributed system.

4.2. Model of File Sharing

A real, genuine model of file sharing would be very complex and would include many parameters, like relations among files, relations among users, characteristics of the individual files (size, life, organization). A more reasonable description of the general model, oriented towards our studies of file migration algorithms, could include the following items:

(1) The order in which different users access a file.

(2) The distribution of the interreference times.

(3) The distribution of the fraction of the file accessed per open.

(4) The probability that an open results in the file being updated.

(5) The correlations between the previous parameters. For example, interreference times may be shorter when the same user opens a file twice in a row.

(6) The relationships between different files.

Even this description is too complex and the models considered will be limited in certain ways. For example, we will not consider relationships among different files even though, in most systems, files are often used in groups. Furthermore, we are making the assumption that, in a single-copy environment, the cost of a remote write is the same as the cost of a remote read. This is true in terms of volume of data transmitted since a single copy has to be update. It is only an approximation in terms of delay because remote reads have to incur a round trip network delay (pre-fetching can help reducing this delay) while remote writes can be implemented so that the issuing program does not have to wait at all. As a result, the sharing process can be modeled by considering only two aspects: the order of reference by users and the fraction of file accessed per open. Such a model can be used to estimate the cost of all the policies that we consider in this chapter. This applies to the costs related to traffic and delay. The storage costs in the single-copy case are constant if the unit storage costs are the same on all the nodes.

4.2.1. Full Markov Model

Figure 4.1 shows a typical pattern of access to a file that is shared by twelve users. Each line represents one user and each horizontal dash a four-hour time interval. Digits stand for the number of times that the file was opened during that one hour period (a dash means zero opens). Two characteristics of the access pattern are worth noting:

```
read     --1-----------------------------------------------------------------------------

read     ---356----343-14732--722---437-1194112-2-6--2111-----21---------------2-11263---

read     ----21------------------55------------------------------------------------------

read     --1------------------------1-----------------------------------------------15---

read     -----------3---112----1122---111-2-1-1---1----72---1--1-----------1--2---1--2---

read     -------11--111211---1--1-1-------11--------------413--241--12--11-1-----1-12----
update   -------22----3-------------------22--------------------2------------------------

read     ---------341--11-----------------22-3--313---232-----------------4---22----11---

read     ----------2-----11---12---------------------------------------------------------

read     ----------62---11----1112--1321---------------6---2738---31----32-----2---5-----

read     -----------3----1-------------73--31---1--1----------------3-1--411---1--1------

read     -------------------------1--------------------------------------------22--------

read     ------------------------------------1-----1---------11----1------11-------------
```

Fig 4.1. Pattern of access to a typical shared file. Each row corresponds to one user. In a row, each dash stands for a period of four hours during which the file was not opened by the user. Digits indicate the number of opens during a given period.

(1) When a file is opened by a given user, it is very likely that the same user will be the next to open the file.

(2) In many of the observed cases, one single user accounts for a large percentage of all accesses to the file. It is not uncommon for this user to be the only one that updates the file. We will call this user the *owner* or the ***high frequency user*** of the file.

In trying to model the observed behavior of shared files, an LRU model [Rau77] would correctly capture aspect (1). Actually, a full LRU model may be an overkill, because the system does not seem to have any memory of the order in which users access the file besides the current user. A simpler model, with a single parameter α, could be used. This model would assume that the probability of a user opening a file twice in a row is α. The probability of any other user opening the file instead is $1 - \alpha$. Both the LRUM and this one-parameter model keep track of the order in which the file is being opened by its users. However, they do not keep track of the names of the users. Consequently, these models cannot accommodate characteristic (2). In order to do that, the model has to remember who is the owner of the file.

A Markov Chain model can handle both requirements by remembering in its states both the name of the high frequency user and the current user of the file. This is the model that we have adopted. Some extra care is needed in the assignment of users to nodes, though.

In the first place, we consider the network nodes, rather than individual users, as states of the model. For the purpose of this optimization, references to a file originating from a cluster of users are equivalent; that is, it does not matter which user makes the reference because the open is local or remote regardless of who is the originator within the node.

A second consideration is the assignment of nodes to the states of the Markov Chain. Our initial goal was to assign all the owners (highest frequency users) to state 1, the second highest frequency users to state 2, and so on. This would require two passes over the trace: one to compute the frequency of access in order to assign the users to the states of the chain and a second pass to estimate the parameters of the chain. Fortunately, we have observed that in more than 90% of the cases (this figure was obtained by sampling 50 files at random), the first user that accesses the file is also the highest frequency user. There are two possible explanations for this experimental evidence. First, if the first access to the file corresponds to its creation, then the user is the real owner of the file and, as we mentioned before, this tends to be a high frequency user. Second, whether or not the first access is the file creation, the probability of observing a high frequency user is obviously higher than that of observing a user that seldom references the file. In the rest of this chapter, we will use interchangeably the concepts of owner of the file, first user and user assigned to state 1 of the Markov chain.

From the previous observations, we have decided on the following assignment of users to state: for each file, the first node that references the file is mapped into state 1 of the chain, the second node into state 2, and so on.

One last decision to be made is how to estimate the parameters of the Markov model. This could be done independently for each file, for some groups of files or for the whole file population. Estimating the parameters for independent files is made difficult by the small number of references to each file. Files could be aggregated by file type, file size, or by the number of users of each file.

We have estimated the parameters of the model by aggregating all shared files in the system. Figure 4.2 shows the state diagram of the Markov chain for the synthetic distributed system with four nodes obtained from the SLAC trace.

The model of file reference is not complete if we do not specify the fraction of file referenced per open. For this, we choose a very simple model. We make the assumption that the fraction of the file used is independent of the

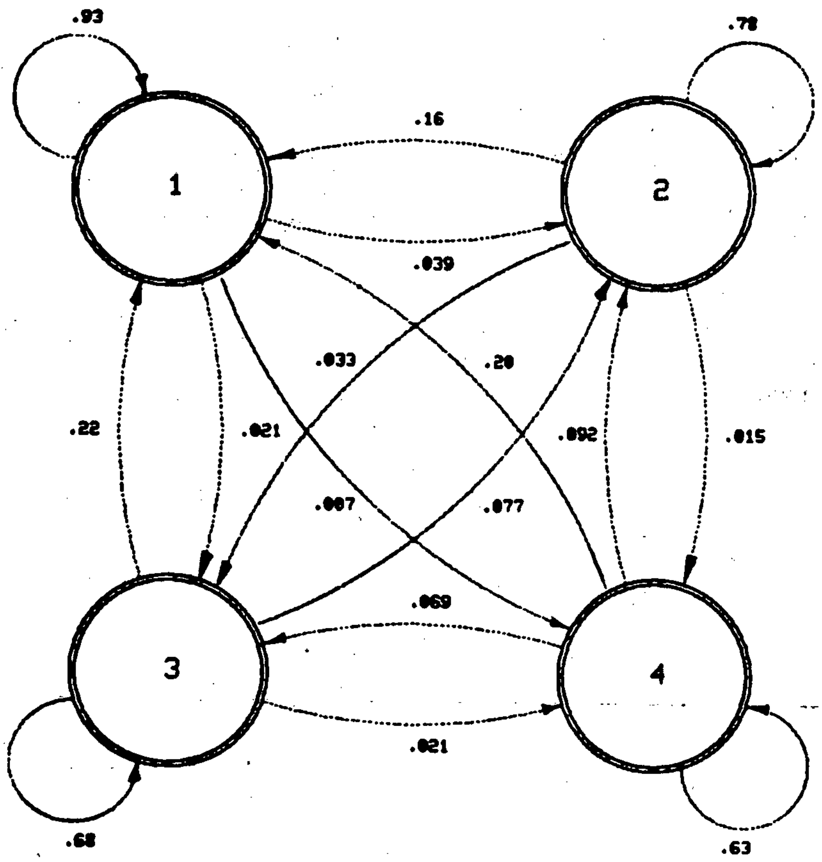

Fig 4.2. An aggregate model of file sharing for the 4-node system obtained from the SLAC trace.

user that is accessing the file. Furthermore, we assume that the fraction of file used per open is independent of the file or any of its characteristics, like size or age (these two assumptions are made for modeling convenience). The distribution of the fraction of file used per open can be estimated directly from the data as an empirical distribution. As a matter of fact, only the first moment of this distribution is needed for our optimization model.

To conclude this section, we review the main aspects of our model of file reference.

(1) All files are assumed independent and belonging to a single population.

(2) The order in which users reference files is modeled as a Markov chain where each state corresponds to a node in the computer network. The states are assigned to the nodes so that state 1 corresponds to the node that creates the file or references it in the first place.

(3) The fraction of the file that is accessed during each open is considered independent of the other characteristics of the file (size, age), of the user (node) referencing the file and of the fraction of the file accessed during previous opens.

4.2.2. Reduced Model

The full Markov model that we just introduced is useful for a number of purposes, like predicting the cost of certain migration algorithms. The problem with this model is that it has $O(N^2)$ parameters, where N is the number of nodes. Beyond four or five nodes, the number of parameters becomes too big.

Even if the parameters could be estimated, the problem of finding the optimal one-copy policy becomes intractable because it requires solving a system of 2 N^2 linear equations in 2 N^2 variables. Doing this symbolically is impossible except for the smallest values of N.

This prompted us to look more carefully at the model of Figure 4.2. Figure 4.3 shows again the same model obtained from the same data. However, this time, the edges are not labeled with the transition probabilities as it is customary in Markov chains. Rather, they are labeled with the stationary transition probabilities, that is, the product of the transition probabilities by the state stationary probabilities. These numbers indicate the long-term probability that the transitions represented by the edges occur.

It is clear from looking at the figure that state 1 is in a class by itself. In the first place, the probability of reentering state 1 is extremely high, compared to the other transitions in the model. In the second place the probabilities of the transitions from state 1 to the other states are also higher than the probabilities of the transitions among the other states. This suggests that a model with only two states, one state equivalent to state 1 and one state equivalent to the remaining states, would capture most of the properties of the full model at a smaller cost. This reduced model should yield the same stationary probabilities than the full model and the same probability of reentry into state 1. The parameters of the reduced model can be derived, if needed, from those of the full Markov model. The reduced model has only two states and hence two parameters no matter what the number of nodes is.

Let N be the number of states of the full model. Let p_{ij} be the transition probabilities of the full model and p'_{ij} those of the reduced model (p'_{12} and p'_{21} are the two parameters of the model). Likewise, let π_i be the stationary state probabilities of the full model and π'_i those of the reduced model. The two conditions to be satisfied are then:

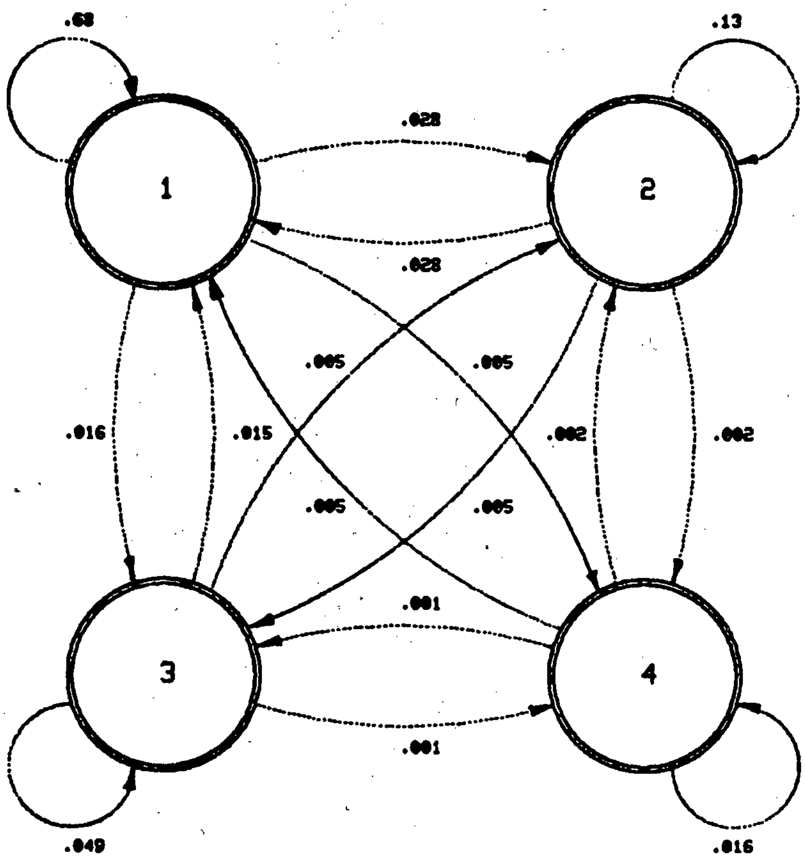

Fig 4.3. Stationary transition probabilities. Same data as Figure 4.2.

$$\pi'_1 = \pi_1 \tag{4.1a}$$

and

$$p'_{11} = p_{11} \tag{4.1b}$$

The first parameter, p'_{12}, is obtained from eq. (4.1b):

$$p'_{12} = 1 - p'_{11} = 1 - p_{11} \tag{4.2}$$

To derive the second parameter, p'_{12}, we first recall that, for a 2-state Markov chain, the stationary probabilities can be obtained in closed form:

$$\pi'_1 = \frac{p'_{21}}{p'_{21} + p'_{12}} \tag{4.3a}$$

$$\pi'_2 = \frac{p'_{12}}{p'_{12} + p'_{21}} \tag{4.3b}$$

From (4.3a) we obtain:

$$p'_{21} = \frac{\pi'_1}{1 - \pi'_1} p'_{12}$$

By using (4.1a) and (4.2) we get to the final result:

$$p'_{21} = \frac{\pi_1}{1 - \pi_1}(1 - p_{11})$$

4.3. Optimal Long Term Solution

Given the reduced model of file sharing presented in section 4.2, we can now derive an optimal policy for the placement and migration of a file in a distributed system. A policy is a mapping between the state of a system and

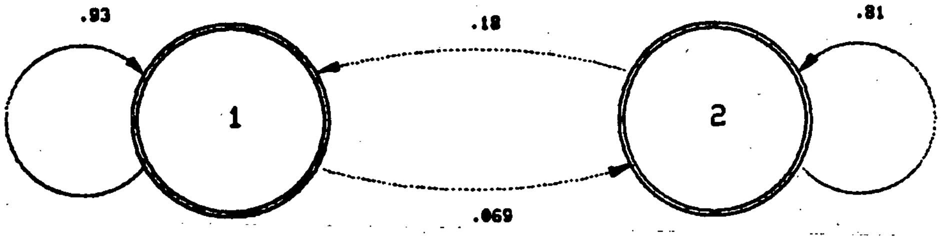

Fig 4.4. Transition probabilities of the reduced model. p_{12} and p_{21} are obtained from the full model.

a set of actions. The state is not an intrinsic property of real systems. It is, rather, a modeling decision. In the first place, the number of state variables can vary widely, depending on the required degree of accuracy. Secondly, the state of a system may involve only a few current variables or the whole history (past, present and future) of the system. For the model of file sharing, the state consists exclusively of the name of the current user, or, more precisely, of the node where the current user resides.

Actions are decisions that (probabilistically) may alter the state of the system. In the framework of the Markov model, this means that when the system enters a state, the probabilities governing departure from the state are not fixed, but rather may be selected from a set of alternatives, depending on the action taken. If there are costs involved in the change of state, these costs may also depend on the action taken.

We now turn to the specific problem of finding the optimal migration policy for the model of file sharing that we described in section 4.2. We start by defining the states of our model. The state of a file is determined by its current user (the users that has opened the file most recently) and by its current location. In our reduced model, the current users may be the "owner" (state 1) or any other user (state 2). Independently of who the current user is, the file may be physically stored in the owner's node or in one of the other nodes. In other words, a user can read and write either from a local copy or from a remote copy. The model remembers both the current user and the current location of the file. The set of states is $S = \left\{ S_1, S_2, S_3, S_4 \right\}$. Figure 4.5 shows the expanded model.

In each of these states, two actions can be taken : M (Move the file in the event of a remote open) and D (Do not move the file and perform a remote

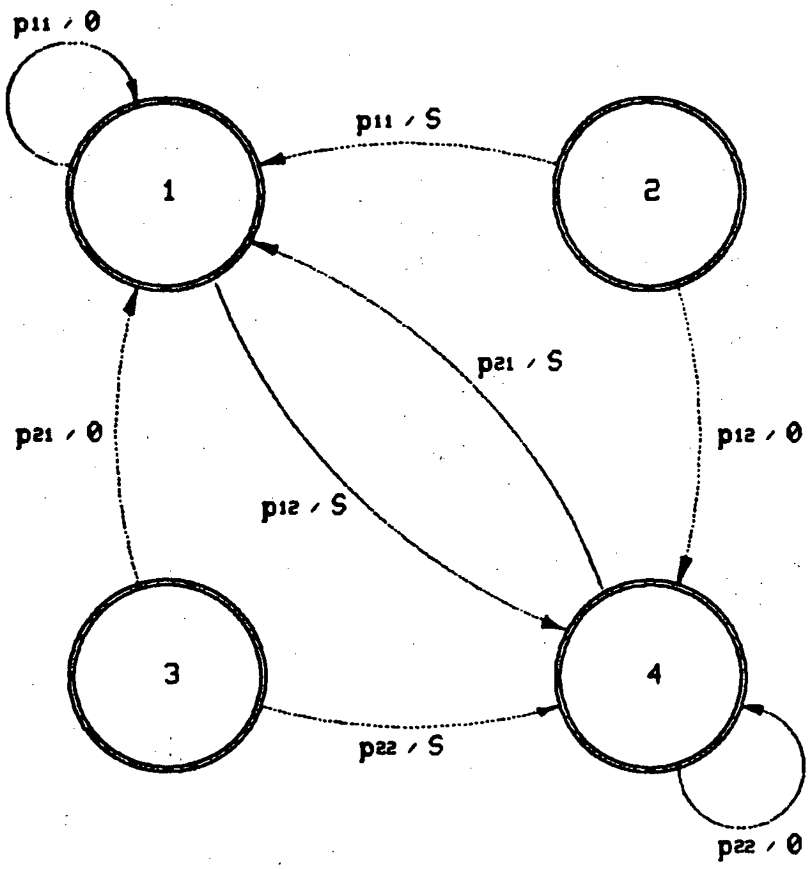

Fig 4.5. Markov model of reference and storage for shared files. The Markov chain results from choosing the "Move" action in all states.
State 1: file at node 1, current user is 1.
State 2: file at node 2, current user is 1.
State 3: file at node 1, current user is 2.
State 4: file at node 2, current user is 2.
The labels on the state transitions are the probability of the transition and the cost of making that transition.

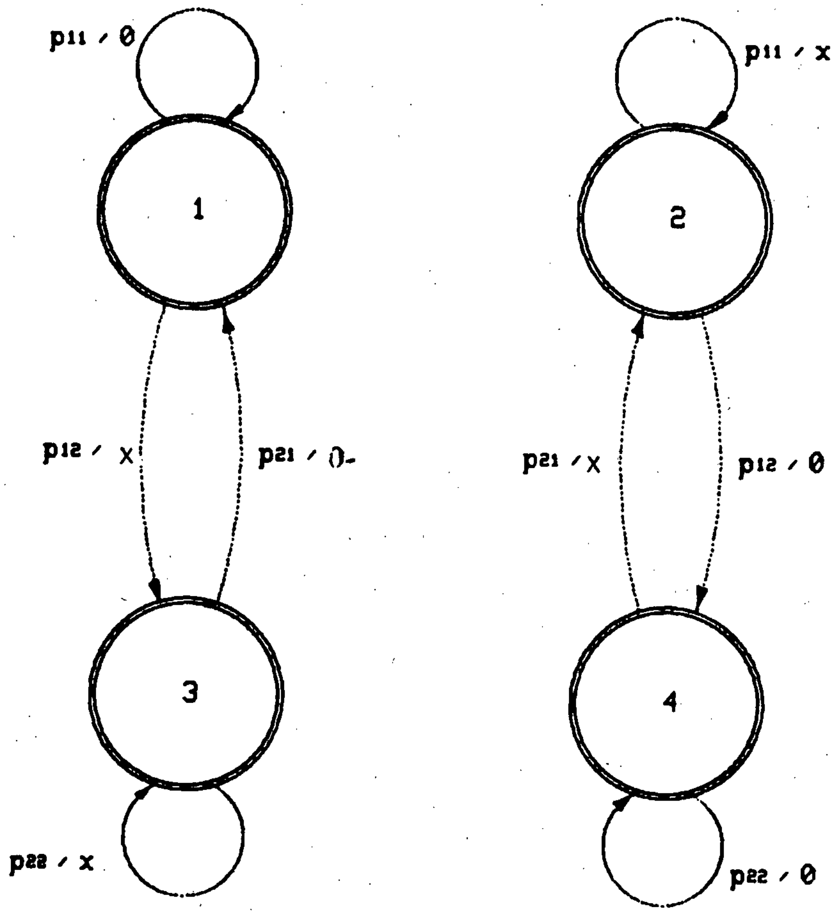

Fig 4.6. Markov model of reference and storage for shared files. The Markov chain results from choosing the "Do Not Move" action in all states.

access). Let $A = \left\{ M,D \right\}$ be the set of actions. M is the action than uses a transfer protocol to move the file to the location of the user that needs it. The cost of this action may be zero if the file is local or it may be non-zero and proportional to the size of the file, if the file is not local. D is the action that does not move the file. The cost of accessing the file is zero if the file and the user are in the same node, and the cost is proportional to the amount of information needed from the file if it is in a remote location.

We will also consider the cost of storing the file. This cost is proportional to the size of the file and to the average length of time that the file remains in a particular node. More specifically, the costs are as follows:

S: Average cost of transferring the file between nodes one and two.

X: Average cost of a remote open (amount transfer during an open).

S_1: Storage cost per unit of storage per unit of time at node 1.

S_2: Storage cost per unit of storage per unit of time at node 2.

τ: Average interopen time.

The model and the optimization procedure can handle different storage costs for the two nodes and different average costs for remote opens. Even though we carry these different costs through the whole optimization procedure, we do not use them in our simulations. In the simulations we assume a common storage unit price and a single average cost for the remote opens. It should be noted that the optimization procedure only requires that the costs associated with the transitions be stationary.

A Markov model where the assignment of probabilities depends on a set of actions is called a *Markov Decision Process* [Ber76, Ros70]. Such a process can be defined as a series of transition matrices, T^*, and cost matrices, C^*, one for each element of A, the set of actions. Upon entering a state, an action is chosen (maybe according to a pre-defined strategy) and the selection of this action impacts the probabilities for leaving the state and the costs of the various alternatives. In our case, we have two actions, M and D and hence two sets of matrices T^M, C^M and T^D, C^D.

The first set corresponds to the Move action. It describes the transition probabilities and the costs if the action Move is chosen in all states.

$$T^M = \begin{bmatrix} p_{11} & 0 & 0 & p_{12} \\ p_{11} & 0 & 0 & p_{12} \\ p_{21} & 0 & 0 & p_{22} \\ p_{21} & 0 & 0 & p_{22} \end{bmatrix}$$

$$C^M = \begin{bmatrix} 0 & 0 & 0 & S \\ S & 0 & 0 & 0 \\ 0 & 0 & 0 & S \\ S & 0 & 0 & 0 \end{bmatrix}$$

These two matrices correspond to the diagram of Figure 4.5.

The second set of matrices correspond to the Do Not Move action:

$$T^D = \begin{bmatrix} p_{11} & 0 & p_{12} & 0 \\ 0 & p_{11} & 0 & p_{12} \\ p_{21} & 0 & p_{22} & 0 \\ 0 & p_{21} & 0 & p_{22} \end{bmatrix}$$

$$C^D = \begin{bmatrix} 0 & 0 & X & 0 \\ 0 & X & 0 & 0 \\ 0 & 0 & X & 0 \\ 0 & X & 0 & 0 \end{bmatrix}$$

These two matrices correspond to the diagram of Figure 4.6.

The elements of C^* are the costs incurred in making transitions between states ie. at the time of opening the file. In addition, files incur a cost $S_1.\tau$ when they are stored at node 1 between opens and a cost $S_2.\tau$ when they are stored at node 2.

Given the transition probabilities and the costs for each alternative, we want to find an optimal policy (a set of actions) that minimizes the average cost of operating the system over an infinite horizon. The techniques for finding this optimal policy are well known [How60, How71]. It can be shown that the optimal policy is a stationary policy under very general conditions. With the help of an algebraic manipulation program †, we have been able to derive the optimal stationary policies for this problem in closed form. The method used has been the policy iteration algorithm described in [How60].

A policy is a vector of actions, one for each state of the model. In this case, the optimal policy is stationary. Therefore, it is a vector of actions that do not depend on the time. Since the policy iteration has been performed in closed form, we are able to derive parametric optimal policies, based on the parameters of the problem (transition probabilities and costs). In the case at hand, we obtain three different policies: P^1, P^2 and P^3 depending on the relationships among X, S, p_{12}, p_{21}, S_1, S_2 and τ. For each optimal policy P, the policy iteration method also yields G, the average cost of operation over the infinite horizon.

$$P^1 = \begin{bmatrix} M \\ M \\ M \\ M \end{bmatrix} \text{ if } \begin{cases} X > 2\,S\,p_{21} + (S_2 - S_1)\tau \\ X > 2\,S\,p_{12} + (S_1 - S_2)\tau \end{cases}$$

$$G^1 = \frac{2\,S\,p_{12}\,p_{21}}{(p_{12} + p_{21})\,\tau} + \frac{p_{12}S_2 + p_{21}S_1}{p_{12} + p_{21}}$$

The first policy, P^1, tells us to always move the file in case of a remote open. Intuitively, we see that P^1 is optimal when the average amount transferred per open, X, is relatively large, compared to the size of the file.

$$P^2 = \begin{bmatrix} D \\ M \\ D \\ M \end{bmatrix} \text{ if } \begin{cases} X \le 2\,S\,p_{21} + (S_2 - S_1)\tau \\ p_{21} > p_{12} \end{cases}$$

$$G^2 = \frac{X\,p_{12}}{(p_{12} + p_{21})\,\tau} + S_1$$

The second optimal policy applies when $p_{21} > p_{12}$, i.e., when the file has a probability of being in state 1 (of the reduced model) greater than that of being in state 2. This is true for the systems that we have observed. In this

† Vaxima, a descendant of Macsyma.

case, P^2 is optimal if the amount transferred per open is relatively small. The policy tells us to move the file away from state 2 (non owner) in case of a remote open, but to keep the file in state 1 (owner), even in the case of a remote open. Therefore, if the file starts in node 1 (as can be expected), the optimal policy in this case is to store it permanently there.

$$P^3 = \begin{bmatrix} M \\ D \\ M \\ D \end{bmatrix} \text{ if } \begin{cases} X \le 2\, S\, p_{12} + (S_1 - S_2)\tau \\ p_{21} < p_{12} \end{cases}$$

$$G^3 = \frac{X\, p_{21}}{(p_{12} + p_{21})\,\tau} + S_2$$

P^3 applies when $p_{21} < p_{12}$ and it is the symmetrical of P^2.

The migration algorithm that chooses and implements the optimal policy will be called AVOPT. It is not a realizable algorithm because it requires the knowledge of the averages of the file sizes and file transfers over an infinite period. A realizable version of AVOPT would use running estimates of all the parameters and it would recompute the optimal policy at each remote open. This adaptive policy is only optimal during the periods where the parameters remain more or less constant. We will call this policy DYNOPT and we will test it, together with AVOPT, using trace-driven simulations.

4.4. Suboptimal Policies

The previous section has left open the problem of estimating the parameters of the model. We will go back to this in the last section of the chapter. But first, we will present three non-optimal policies that do not require the estimation of parameters in order to work.

4.4.1. Remote I/O

The first of the three policies is *Remote I/O* or *RIO* for short. This is a simple policy that stores the file permanently with the user that created it. All accesses from remote users are handled through remote file access and the file is never moved. The policy can be defined in the full Markov model as

$$P^{RIO} = \begin{bmatrix} D \\ D \\ D \\ D \end{bmatrix}$$

It must be noted that all the actions besides the one for state 1 are irrelevant. Since the file is placed, upon creation, in state 1 and the action for state 1 is Do Not Move, none of the other states will ever be visited, and the actions will not be used.

In the calculation of the costs for the policies and in the trace-driven simulations that we conduct in the next section, we will assume that the storage costs are identical at all nodes and equal to S_1. The average cost of the policy, calculated from the full model, is

$$G^{RIO} = \frac{(1 - \pi_1)}{\tau} X + S_1$$

Substituting π_1 from (4.3a), we get:

$$G^{RIO} = \frac{X\, p_{12}}{(p_{12} + p_{21})\, \tau} + S_1$$

So G^{RIO} is equal to G^2. This comes to no surprise because P^{RIO} is actually the same policy as P^2 when the model is started in state 1.

4.4.2. Optimal Remote I/O

RIO has the assumption built in it that most of the accesses to the file will come from users located at the node of its owner. It would be interesting to see how good an assumption this is. One way of conducting the test is to implement a non-realizable policy, that actually places each file at the node of greatest activity. We call this policy *Optimal Remote I/O, OPRIO* for short.

One other reason for introducing this policy and measuring its behavior is that OPRIO is the policy that is implicitly used in many of the papers that solve the file assignment problem by mathematical programming methods. Using this policy in simulations will allow us to compare the best one-copy policy attainable by a fixed assignment of files to some of our more dynamic policies.

In terms of actions, this policy is

$$P^{OPRIO} = \begin{bmatrix} D \\ D \\ D \\ D \end{bmatrix}$$

The difference with P^{RIO} is, of course, in the initial location of the file. Conceptually, two passes over the data are needed in order to implement this policy: one to find the heaviest user for each file and one to measure the cost of the policy. OPRIO could be approximated by using an exponential weighting estimator of the frequencies and by moving the file to the node with the highest frequency of use.

4.4.3. Most Recently Used

This policy moves the file to the node of the current user at every open. It is called the *Most Recently Used (MRU)* policy, because the file is always located at the node where it has been most recently opened. In terms of actions, it can be defined as:

$$P^{MRU} = \begin{bmatrix} M \\ M \\ M \\ M \end{bmatrix}$$

The cost obtained from the full Markov model is:

$$G^{MRU} = \frac{1 - \sum_{i=1}^{N} p_{ii} \Pi_i}{T} S + S_1$$

4.5. Simulation Results

Trace-driven simulations have been run in order to compare the performances of the policies that we have introduced. We measure their performance based on two variables:

(1) The number of remote opens originated.

(2) The average traffic generated.

Before we show the results obtained in the simulations, we must explain certain implementation details for some of the policies. In the definition of RIO, for example, we stated that the file is placed with the user that creates it. In the simulation, however, we do not see the creation of all the files and in some cases it is not possible to determine the owner of the file. In these cases, we have placed the file with the user that has first opened the file in the span of the trace.

Some explanation is needed for AVOPT, the average optimal policy and DYNOPT, its adaptive version. Let us recall that the optimal policy was derived from the two-state, reduced model of file reference. The reduced model does not contain any transitions between two non-owner users. Consequently, the optimal policy does not provide an action for such a transition. In our implementation of AVOPT and DYNOPT, in the event of a transition between non-owners, we transfer the file. It is very likely that the next transition will be to the same user or to the owner of the file. In that case, the optimal policy can be applied again. The transitions between non-owners could be handled in several other ways. For example, we could do what the optimal policy says when there is a transition between non-owner and owner. Alternatively, we could do what the policy specifies whenever there is a transition between non-owner and owner.

Finally, in the case of DYNOPT, we must describe how the parameters of the model are estimated. The model has three types of parameters, the transition probabilities, the file sizes and the amount of information accessed per open. For each one of these parameters we have chosen a different estimation procedure.

The transition probabilities are estimated as an unweighted average of the last five transitions out of a particular state for each file (only two parameters, p_{12} and p_{21}, need to be estimated). The exploratory data analysis shows that 70% of the shared files are opened more than ten times. In the period of time before the estimates can be obtained, the files are managed using MRU.

We use the current size of the file as the estimate for the long term average. This requires some explanation. It has been observed in the trace that the size of a file rarely changes during its entire life. Furthermore, most of the changes in size are increases. Under these circumstances, it makes sense to forget past information about the size of the files.

The amount of information accessed per open shows much more variability that the size of the file. Our estimate is an unweighted average of the observed values since the last change in file size.

Finally, we must say what are the costs that we consider in obtaining the optimal policies AVOPT and DYNOPT. One assumption that we make in all the cases is that the storage costs are the same at all the nodes of the computer network. This in turn means that storage costs for all single-copy policies are the same and they can be disregarded in the optimization procedure. The only cost left is the cost of transmission. Both in the file transfers and in the remote accesses, we have made the assumption that the transmission costs are proportional to the amount of information (bytes) that is transmitted.

Based on all these assumptions, we have run trace-driven simulations for the five policies (RIO, OPRIO, AVOPT, MRU, DYNOPT) and have measured both the number of remote opens and the average traffic generated by each one of the policies.

4.5.1. SLAC Trace

The first set of figures corresponds to the SLAC trace.

It is not surprising that MRU is the policy which generates the smallest number of remote opens. After all, the Markov model of file sharing shows a very strong user locality in the referencing process and MRU is the policy that adapts the fastest to the changes in locality. As for the amount of traffic generated, it is normal that the optimal policies come ahead of MRU, since the optimization was done in terms of average traffic. We should also explain why the two optimal policies produce such close results and why in some cases the adaptive policy is even better than AVOPT, even though it is not backed up by the theoretical optimality result. The main reason is that AVOPT uses parameters for the Markov model that are estimated over the entire population of files, while DYNOPT uses estimates of the parameters on

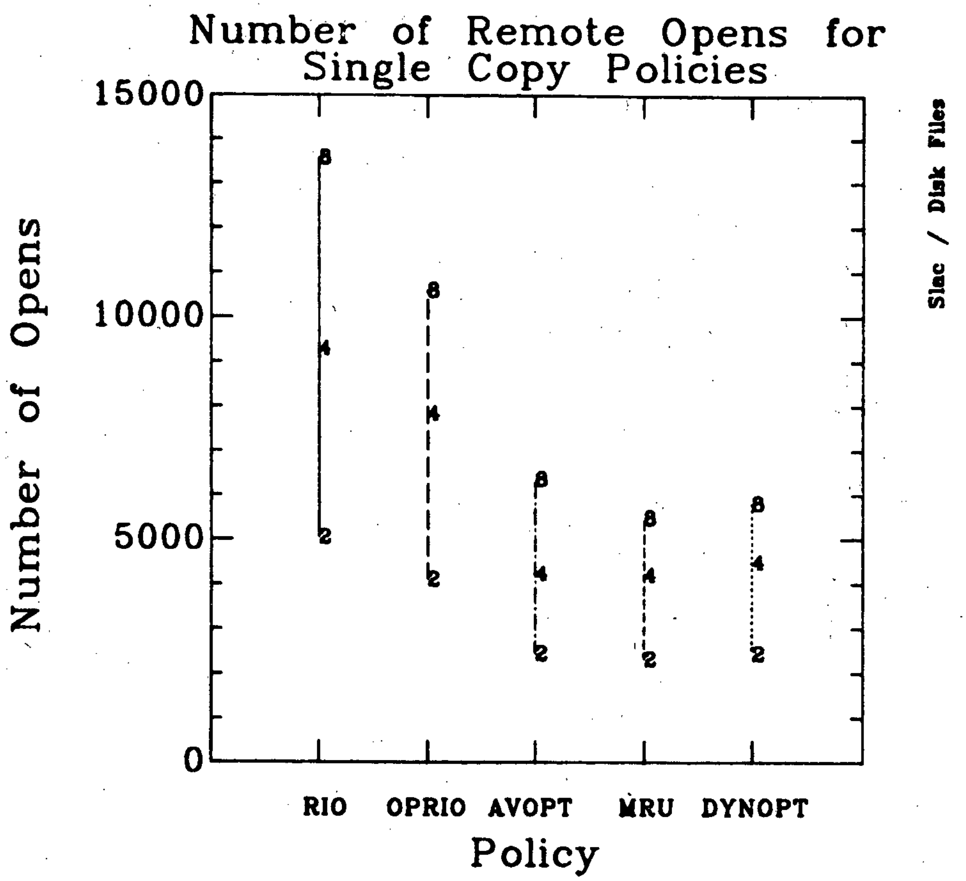

Fig 4.7. A comparison of the number of remote opens for the one-copy policies in three different synthetic distributed systems with 2, 4 and 8 nodes. SLAC trace.

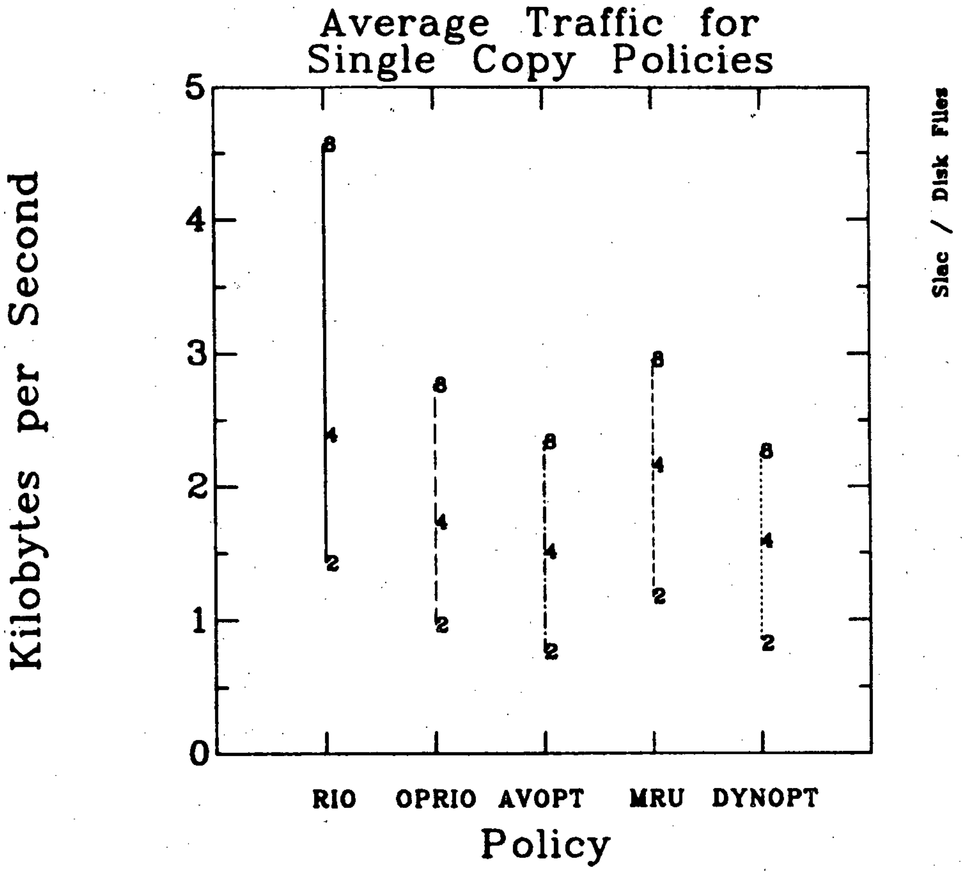

Fig 4.8. A comparison of the traffic generated by the five one-copy policies in three different synthetic distributed systems with 2, 4 and 8 nodes. SLAC trace.

a per file basis. This implies that the decisions made by DYNOPT are closer to the behavior of each particular file.

4.5.2. Hughes Aircraft Trace

When the Hughes system was partitioned in chapter 3 we made the remark that the components were almost completely disconnected in terms of the files that they share. Therefore, the number of remote opens and the traffic generated by the shared files should be very low. This is actually the case. Figure 4.9 shows the number of remote opens generated by the five policies that we study. The number of remote opens is about half of that generated by the SLAC installation even though the Hughes system has a higher number of opens per hour. As was the case in the SLAC simulation, the smallest number of remote opens is generated by MRU.

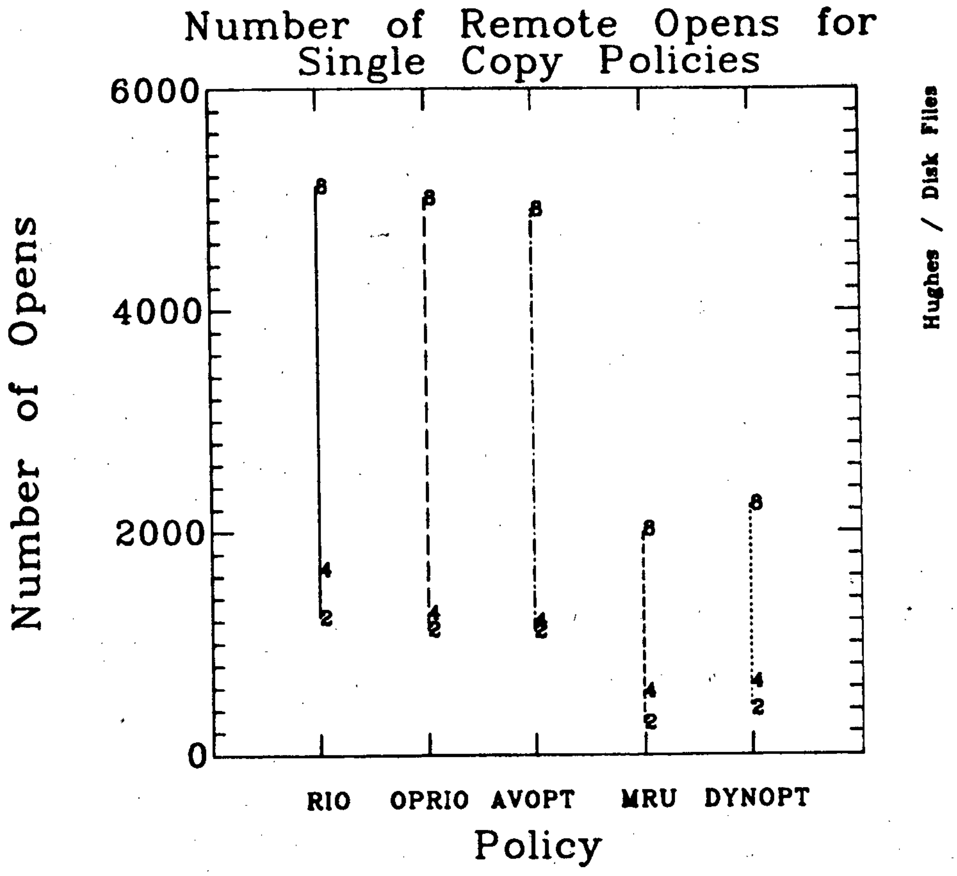

Fig 4.9. A comparison of the number of remote opens for the one-copy policies in three different synthetic distributed systems with 2, 4 and 8 nodes. Hughes trace.

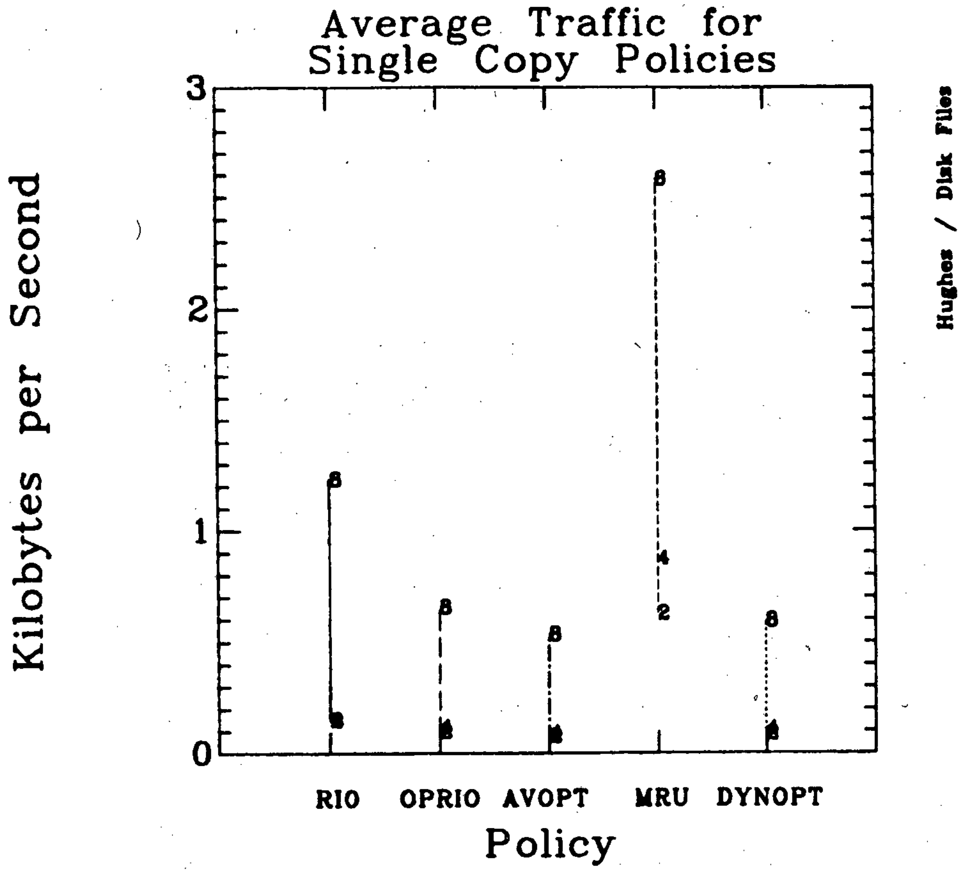

Fig 4.10. A comparison of the traffic generated by the five one-copy policies in three different synthetic distributed systems with 2, 4 and 8 nodes. Hughes trace.

The measurement of traffic (Figure 4.10) produces some surprising results in the traffic generated by MRU. After closer examination, it appears that a few very large files that are used by more of one node are responsible for the traffic. The performance of RIO, for example, is better because the large files are not moved. Again, the main reason for these results is that there are very few remote opens and the relative performance of the policies are dominated by the large sizes of a few files.

4.6. Conclusions

In this chapter, we have developed a model of file sharing and from this model, we have derived an optimal one-copy policy (AVOPT) for the migration of files in distributed systems. Improvements in the order of 50% in generated transmission traffic with respect to the most static policy are achieved with an adaptive, implementable policy (DYNOPT). The cost model for this policy is very flexible. Our optimizations and simulations have been focused on average traffic but any other stationary cost measures could be handled. For example, it would be as easy to introduce costs related to the delay in obtaining remote data or a composite of traffic, delay and storage space.

One policy that has performed better than expected is OPRIO, the optimal static assignment of files. An adaptive version of OPRIO, that keeps the file in the most active node over a period of time, would be worth investigating.

CHAPTER 5

MULTIPLE-COPY MIGRATION POLICIES

Summary

This chapter presents a simplified mode of operation for a distributed computer system that maintains multiple copies of files. Based on this mode of operation, we introduce a family of file migration policies: the Space-Time Update-Rate Working Set policies.

Trace-driven simulations are used to evaluate the performance of these policies. The simulations suggest that the best policies (in terms of network traffic generated) are those that penalize copies that are not being locally referenced but are being updated through the network in order to maintain their consistency.

Introduction

In Chapter 4 we presented various single-copy policies for placing and migrating files in distributed systems. Some of the policies allocated files to nodes of the network using very simple heuristics. Others were much more complicated and used a dynamic assignment based on adaptive estimation of various parameters. One of the main conclusions of that study is that the most complex policies achieve, at best, a fifty percent reduction in network traffic over the simplest policy.

Further inspection of the patterns of access presented in chapter 4 reveals the existence of *extended localities*. Extended localities occur when a (generally small) number of users access a file over a short period of time in a random order. This situation generates unavoidable traffic as users located at different nodes access the file almost concurrently. Still, the existence of some locality should be exploited. One way of doing so is to provide each active member of the extended locality with a copy of the file.

Keeping multiple copies of files in a system can be an expensive mode of operation. First of all, copies take up physical storage space. Secondly, the structure of the directories becomes more complex and looking for files may be slower, increasing user delay. Finally, maintaining the copies of the file in a consistent state may require a fair amount of update traffic if the file is updated frequently. Our goal in this chapter is to see whether the decrease in the number of file transfers can offset these costs.

In Section I we will propose a mode of operation for file systems maintaining multiple copies of files. The placement and migration policies are introduced in Section II. Finally, Section III contains the results of the trace-driven simulations that have been conducted to evaluate the policies.

5.1. Management of Multiple Copies of Files

It is not the goal of this research to devise new concurrency control algorithms for distributed file systems. There is a large body of literature on the subject [Min79, Bad78, Lin79, Sto78, Tho79] and two recent surveys [Ber81, Kol81], have covered thoroughly this area of research. However, it must be pointed out that most of the effort has been directed to the mode of operation of DDBMS's (Distributed DataBase Management Systems), not distributed file systems. The distinction is important because the concern with concurrency control and with crash recovery in DDBMS's has produced updating algorithms that are very expensive in terms of overhead [Bad81, Rie79, Gar79]. This overhead is so big under general assumptions that some researchers have concluded that it is not worth replicating data in distributed databases unless availability is an overriding factor [Gar81]. We believe that the operating requirements of distributed file systems are substantially different from those of DDBMS's and that the question of file replication should be investigated.

It must be noted, before we make any more statements about distributed file systems, that we are not talking about transaction-oriented systems like DFS [Stu80] or the Cambridge File Server [Dio80] but rather about the distributed versions of "classical" file systems provided by operating systems like IBM/OS or UNIX. These file systems basically provide a read operation, a write operation and some primitive internal locking mechanism to preserve the internal consistency of the file and the associated directories. Under these conditions, we can describe the reasons why the operation of this type of file system is considerably simpler than that of a DDBMS.

In the first place, crash recovery is usually not handled directly by the file system. File systems and device malfunctions are usually corrected by restoring information from periodic dumps. In any case, the update algorithms do not have to be concerned with possible crashes. Secondly, concurrency control in file systems is in a much more primitive state than in database management systems (many operating systems do not provide any concurrency control at all). Usually, it is not a feature of file systems to provide synchronization primitives as a side-effect of the read and write operations. Rather, applications programs use other Interprocess Communication (IPC) mechanisms to control concurrency.

We conclude from these remarks that a very simple mechanism with write-locks is sufficient in most cases to maintain consistent copies of files in a computer network. We will now describe this mode of operation in more detail.

5.1.1. Mode of Operation

The main purpose of this section is to present a plausible mode of operation in order to estimate the costs involved in the management of multiple copies of files. An actual implementation would have to take into account many more details, including race conditions and deadlock detection and avoidance. We will not discuss these details nor the directory mechanisms necessary to search files and copies of files in a network. We first describe the data structures needed and then present the way in which the major file access operations work.

5.1.1.1. Master Copy

For each file, there is at least one copy, the *master copy*, in the system at all times. This copy may be statically assigned to a node of the network or it may logically move from node to node. In our implementation, the master copy is the most recently accessed copy. The master copy is never deleted.

5.1.1.2. Write Lock

The master copy has a write-lock associated with it. In order to write to the file, a user must obtain the write lock from the master copy. If it succeeds, it owns the lock and the local copy becomes the current master copy. If the master copy is being updated, the lock cannot be obtained and the user has to wait until the master copy is closed and the lock released.

5.1.1.3. Read Locks

All copies of the file have read locks. The read lock of a copy is set only when updates that have taken place at a remote master copy are being transmitted to the copy. This is the only time when copies are locked for reading.

5.1.1.4. Reading the File

When a user wants to read the file and there is a copy at the local node, this copy is used, as long as it not read-locked. If it is read-locked, meaning that the copy is being brought up-to-date with the master copy, the user must wait. Since the length of time that the copy is read-locked is short (updates are sent in one batch) we make users wait instead of allowing remote access to a non-read-locked copy. If there is not a copy at the local node, a fresh copy is brought over the network from any copy that is not read-locked. This is done even if the file as a whole is write-locked.

5.1.1.5. Updating the File

When a user wants to update the file, he must make sure that the file is not being updated by another user. This is done by checking whether the master copy is write-locked. If it is, the user has to wait until the lock is released. If it is not, the user obtains the lock, and the user proceeds with updating the file. If there is no local copy, one is brought in from another node. This transfer can be avoided if the user is overwriting the file rather than updating it. When the updates are done and the copy is closed, the write lock is released and the updates are distributed to the other copies of the file.

5.1.1.6. Distributing the Updates

When the master copy has been updated, the remaining copies have to be brought up to date. This is done by read-locking all the other copies (this is the only time that the read locks are used) and transmitting all updates in a batch. In order to read-lock a copy, it must not be opened. If it is, the updating has to wait until the copy is not being referenced.

The mode of operation that we have just described ensures that copies of the file are always available for reading except during the periods when updates are being distributed. In particular, users can read their (slightly outdated) local copies when the master copy is being actively updated. Furthermore, opening a file for reading does not involve any network operation if there is a local copy of the file. This is important because more than eighty percent of all opens are read-only opens and because the policies that we are going to define achieve hit ratios on the order of 0.9 and higher. Therefore, we want to make reads to local copies inexpensive even if that increases the cost of writing into the file when there is no local copy (writing to the master copy does not involve any communications either).

If an application is sensitive to the on-going updating of a file at a different node, then a high level inter-process communication protocol should be used for synchronization purposes.

One last aspect of our mode of operation is that all the copies of the file are maintained up-to-date between opens. This is in contrast with the updating algorithm used in WFS [Gif79]. In WFS, there are stale copies of the file in the system. The file system uses a weighted voting mechanism and file version numbers to decide what are the copies that contain current information. Since we are more concerned with the performance aspects of the algorithms than with the reliability issues, we will not consider maintaining stale copies of the files in the system. Stale copies are costly in terms of storage and users incur extra delay because of the voting when they need to use the file.

5.1.2. Cost Model

In our cost model we will be making most of the assumptions that we made in the study of single-copy policies about storage and transmission costs. Namely, storage and transmission costs will be considered proportional to the size of the file.

We will consider three main costs in the operation of the file system:

(1) C_t: Cost of creating a new copy and transferring it to a node.

(2) C_s: Storage cost.

(3) C_u: Cost of maintaining a copy up-to-date when the file is being updated at remote nodes.

The cost of creating a new copy of the file is, for our purposes, equal to the cost of transferring the file from the node having the master copy, let us say, to the node that needs the new copy. This is a traffic cost and is equal to:

$$C_t = S \times c_t$$

where

S: Size of the file.

c_t: Communication cost per unit of information.

The storage cost will be considered to be proportional to the size of the file and also to the length of time the copy spends at the node:

$$C_s = S \times c_s \times t$$

where

S: Size of the file.

c_s Cost of storing one unit of storage per unit time.

t Time during which the file is stored at the node.

To calculate the update cost, we make the assumption that the traffic generated by the locking messages is negligible. These messages are only needed at open and close time and have a small, constant size. Furthermore, only fifteen percent of all opens (for shared files) are update opens and these are the only opens requiring locking messages. Consequently, the cost of updating copies of files is the traffic cost of transmitting the changes to the remote copies. As we mentioned before, these updates can be transmitted all at once when the active copy is closed after updating.

As we described in section 5.1.1, every time that the master copy is updated, all the other copies are updated. Lets U_r be the average rate of information (bytes per second) written to a file (or to its master copy, since there may not be concurrent updates to other copies) by all its users. The updates to the master copy are free in terms of network traffic or delay because the master copy is local to the user updating the file. Eventually, the updates have to be transmitted to all the existing copies of the file. Over the long run, each copy will incur a cost:

$$C_u = U_r \times c_u \times t$$

where:

U_r: Average rate of update.

c_u: Unit cost of update traffic. This cost can be equal to c_t in many cases. We make the distinction, though, because this traffic can be considered as low priority traffic and could possibly benefit from lower communication rates.

t: Time that the copy remains at the node.

5.2. The Migration Policies

The mode of operation that we have outlined in the last section somewhat constraints the type of policies that are available to us. All the policies that we will study have the following characteristics:

(1) They are demand policies.

(2) They use file transfer as opposed to remote I/O. This constraint could be relaxed in later studies. In particular, a combination of the optimal policies of Chapter 4 and of multiple-copy policies should be investigated.

(3) They are variable space policies.

(4) They treat files independently.

We are limiting this study to demand policies. In this environment, a demand policy is one that transfers files only when a user wants to open a file and there is no local copy available. In particular, we do not consider moving files when other related files are being moved. Related files are files that are frequently used together by a same program, for example. Policies that initiate the transfer of files before they are accessed by the users called pre-fetching policies. This type of policy incurs two types of transmission traffic: demand traffic and pre-fetching traffic. It is usually expected that some of the pre-fetching traffic will offset some of the demand traffic and a substantial part of the delay costs.

Our policies do not generate pre-fetching traffic but they create update traffic in addition of the demand traffic. As a matter of fact, we will present results showing that a considerable amount of traffic can be generated by updates being transmitted to all the copies of a file in the system.

There are at least two ways of maintaining the master copy of each file in the system. One way is to assign the master copy statically to a node in the network. Another way would be to make the most recently used copy play the role of the master copy. For our simulation studies, we have selected the second approach.

Our migration policies are an extension of the Space-Time Working Set policy [Den78, Smi81b]. Space-Time Working Set (STWS) removes any file for which the space-time product is greater than a certain parameter C:

$$S \times t \times c_s > C$$

Note that this policy has only one parameter: $\frac{C}{c_s}$ that can be interpreted as the ratio of fetching cost to storage cost and that the fetching cost is considered constant (independent of the size of the file). This is in accordance with the characteristics of many secondary and tertiary storage devices, for which the time necessary to access a file is mostly spent in the movement of mechanical parts. The actual transfer times of the files constitute such a small portion of the whole process that it is usually disregarded.

Our policies are Space-Time Update-Rate Working Set (STURWS) policies. They remove any copy for which:

$$S \times t \times c_s + Ur \times t \times c_u > C \qquad 5.1$$

The implicit assumption is that the file that has accumulated the largest retention cost is likely to incur the largest retention cost to the next reference and hence removed. STURWS policies have two parameters:

(1) $\frac{C}{c_s}$, a ratio of fetching cost to storage cost.

(2) $\frac{c_u}{c_s}$, a ratio of update communications cost to storage cost.

In our case, unlike in the classical STWS policy, we will consider the fetching cost proportional to the size of the file, as we said in section 5.1.2.

This general policy generates two subclasses of policies, depending on how U_r is computed.

(1) U_r can be computed as an long term average of the update activity.

(2) U_r can be measured since the last reference.

Of all the possible policies that can be derived from the general STURWS, we have chosen four in order to run the trace-driven simulations. We now describe each one of these policies.

5.2.1. Mean Update Rate (MUR)

This policy uses an overall average of the update rate (including future activity) as an estimate of U_r in equation (5.1). Consequently, it is not a realizable policy. We test this policy anyway in order to compare it to the policies that actually measure the amount of update transmission received by the copy since the last reference.

5.2.2. Working Set (WS)

This policy is obtained by setting $c_u = 0$ in (5.1). The rule for removing copies becomes:

$$S \times t \times c_s > C$$

and if we assume that C is proportional to the size of the file, we obtain:

$$S \times t \times c_s > c_t \times S$$

where S is the size of the file, c_s is the cost of storing one unit of storage per unit time, c_t is the communication cost per unit of information and t is the time since the last reference. Therefore, this policy removes any file that has gone unreferenced for a time t such that:

$$t > \frac{c_t}{c_s}$$

Since $\frac{c_t}{c_s}$ is a constant for all files, this policy is Working Set (WS). By further varying the ratio $\frac{c_t}{c_s}$, we can obtain working set policies operating at different transfer rates and with different storage requirements. At one end, for $\frac{c_t}{c_s} = 0$, the working set policy behaves like MRU, the single-copy policy presented in the last chapter. At the other end, for $\frac{c_t}{c_s} = \infty$, the working set policy degenerates into a policy that maintains all the copies of the files ever created in the system in a consistent state.

5.2.3. Space-Time Update Working Set (STUWS)

This policy uses a real measurement of $U_r \times t$ since the last reference rather than an average of U_r. As was mentioned earlier, our policies have two parameters, $\frac{c_t}{c_s}$ and $\frac{c_u}{c_s}$. For our simulations we have chosen the two parameters such that $c_u = c_t$. One interpretation of this equality is that we are using the same priority (hence the same cost) to transfer files on demand and to distribute updates from the master copy.

5.2.4. Delete On Update (DOU)

A conceptually different policy can be obtained by setting $c_u = \infty$ in eq. (5.1). This policy will remove any copy (other than the master copy) that is about to receive any updates. In the absence of updates, the policy behaves like a pure Working Set policy. In the limit, when $\frac{c_t}{c_s} \rightarrow \infty$, the policy keeps read-only copies indefinitely but flushes them as soon as they have to be updated.

Before showing the results of the trace-driven simulations that have been run to test the four policies that we have described in the last four subsections, we shall make a comment about the small effect that the size of the files seems to have on these policies. This may seem surprising to the reader who is familiar with the generalizations of the working set policy to variable size objects. For example, in the classical case of file migration between disk and tape, the size of the file plays an important role. This is because, while the cost of storage is always considered proportional to the size, the cost of transferring the file from tape to disk is usually considered a constant, due to the large delay involved. This explains why Space-Time Working Set policies tend to eliminate large files from active storage while retaining the smaller ones. In our case, both the storage cost and the transfer costs are proportional to the size of the file and hence, the size of the file becomes an irrelevant factor in determining which copies must remain and which must be deleted.

5.3. Experimental Results

In order to compare the policies that we have defined, trace-driven simulations have been conducted on the same synthetic systems that were used to test the single-copy policies in Chapter 4. For each policy we have measured the average storage space needed and the average traffic generated. In order to estimate the average delay experienced by the users in accessing their files, we have measured the number of opens that result in a file transfer and the amount of information transmitted during these transfers. From these measurements and the characteristics of the network (network delay, network bandwidth, operating system overhead), a measure of delay can be obtained.

5.3.1. SLAC

The first measure used to compare the policies is the number of remote opens that they generate. As a first approximation, the delay experienced by the users of the file system can be considered to be proportional to the number of remote opens. Figure 5.1 shows the performance of all the policies when used in three synthetic systems, with two, four, and eight nodes. For comparison, the performance point of MRU (Most Recently Used, single-copy policy) is plotted for each system. All the policies behave very similarly to pure Working set as far as the number of remote opens is concerned. Table I shows that, for the WS policy, almost a thirty percent reduction in demand traffic can be obtained by maintaining copies of files alive for one hour after they have been referenced. The penalty in storage space for doing this is almost negligible (under two percent). By retaining the copies 4 hours after the last reference, we can achieve a fifty percent reduction in the demand traffic. The rate of return in reduction of demand traffic and number of remote opens decreases substantially after the twenty four hour mark. From then on large increases in storage space have a small payoff in remote opens saved and finally, when all the copies are kept for periods of time in the order of weeks, the demand traffic is one order of magnitude less than the traffic incurred by the single copy policy MRU.

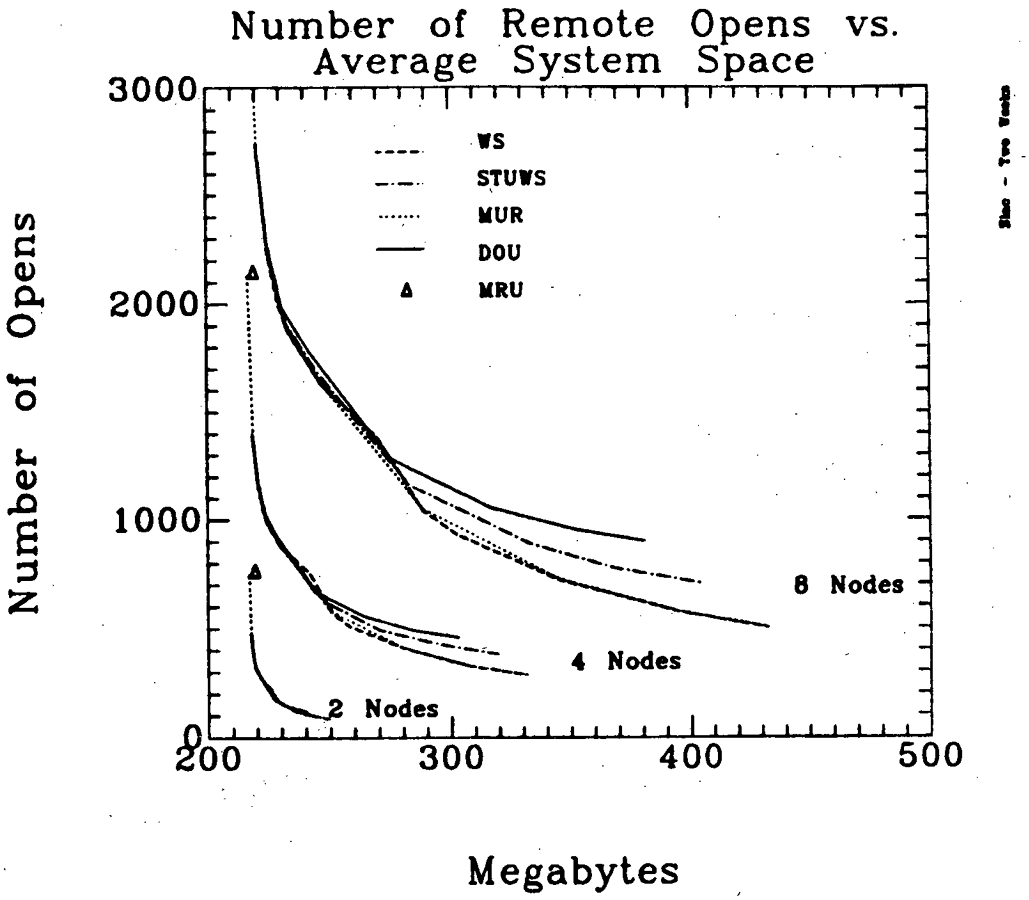

Fig 5.1. Number of remote opens vs. average system space for the SLAC trace. Each curve corresponds to one value of the update traffic cost. The points on the curve are obtained by varying the demand traffic cost with respect to the storage cost.

Table I. Retention Times for the Working Set Policy (SLAC)

Retention Time (Hours)	Avrg. System Space (Megabytes)	Avrg. Total Traffic (Kbytes/Sec.)	Avrg. Demand Traffic (Kbytes/Sec.)
0.	216.7	3.25	3.25
1.	220.6	2.64	2.31
2.	225.1	2.43	1.87
4.	229.1	2.27	1.63
6.	232.8	2.25	1.51
8.	246.1	2.40	1.26
16.	269.7	2.34	1.00
32.	288.5	2.23	0.724
64.	302.5	2.22	0.648
128.	345.2	2.29	0.500
256.	395.1	2.41	0.386
512.	432.8	2.52	0.338

Figure 5.2 shows the demand traffic generated by the transfer of remote files at open time. The demand traffic follows very closely the number of remote opens that are incurred by the system. Again, the differences between policies that handle update traffic in different ways is very small. It must be noted, though, that the policies that do not charge copies for their update traffic perform slightly better, in terms of demand traffic. This is not surprising since copies are kept alive longer, in the average, when the cost of updating them is disregarded.

Figure 5.3 shows the average total traffic generated by the operation of the migration policies. Here the policies that take into account the update traffic are clearly superior to the ones that do not. A good point of reference in Figure 5.3 is the curve for the case where the cost of updates is considered infinite. This has the effect of deleting any copy that is going to be updated. This policy is called "delete on update" in the Figure 5.3. Since this policy incurs no update traffic, its total traffic is equal to the demand traffic generated and hence is identical to the corresponding curve in Figure 5.2. Compared to this policy, the policy that makes the costs of demand and update

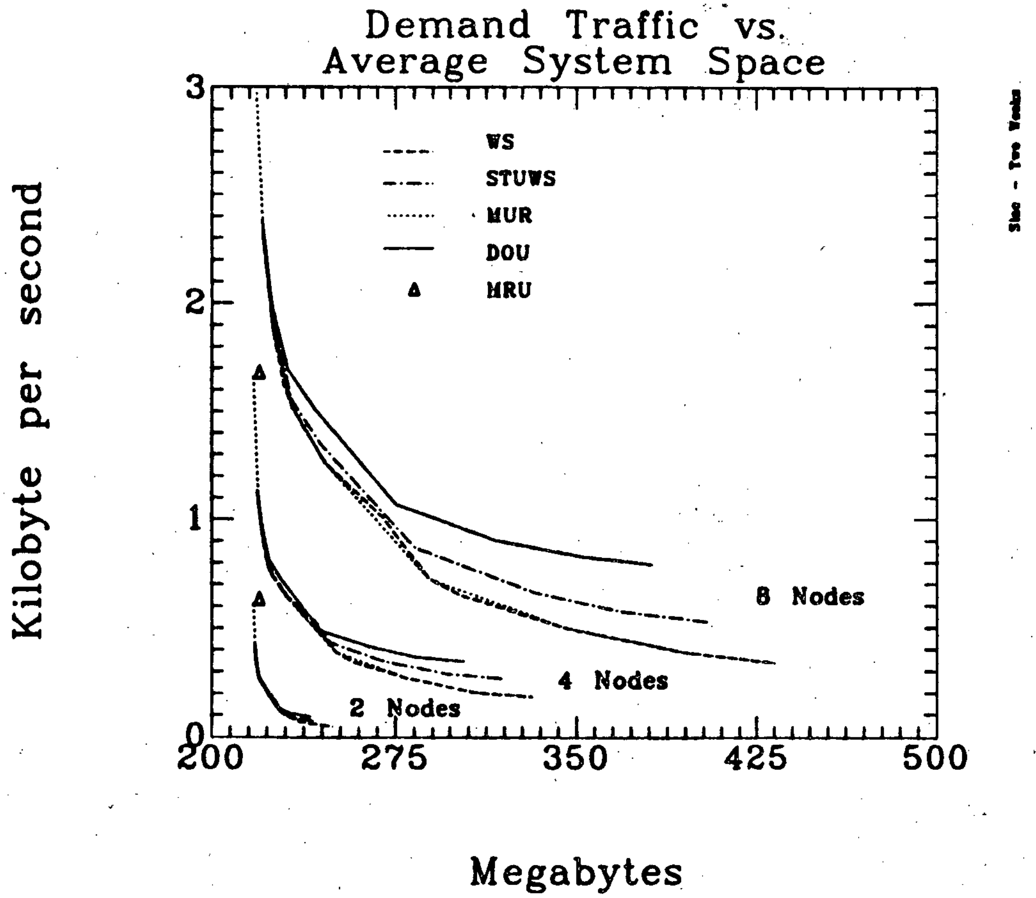

Fig 5.2. Demand traffic vs. average system space for the SLAC trace. Each curve corresponds to one value of the update traffic cost. The points on the curve are obtained by varying the demand traffic cost with respect to the storage cost. This in turn determines the maximum time that a copy remains alive after its last reference.

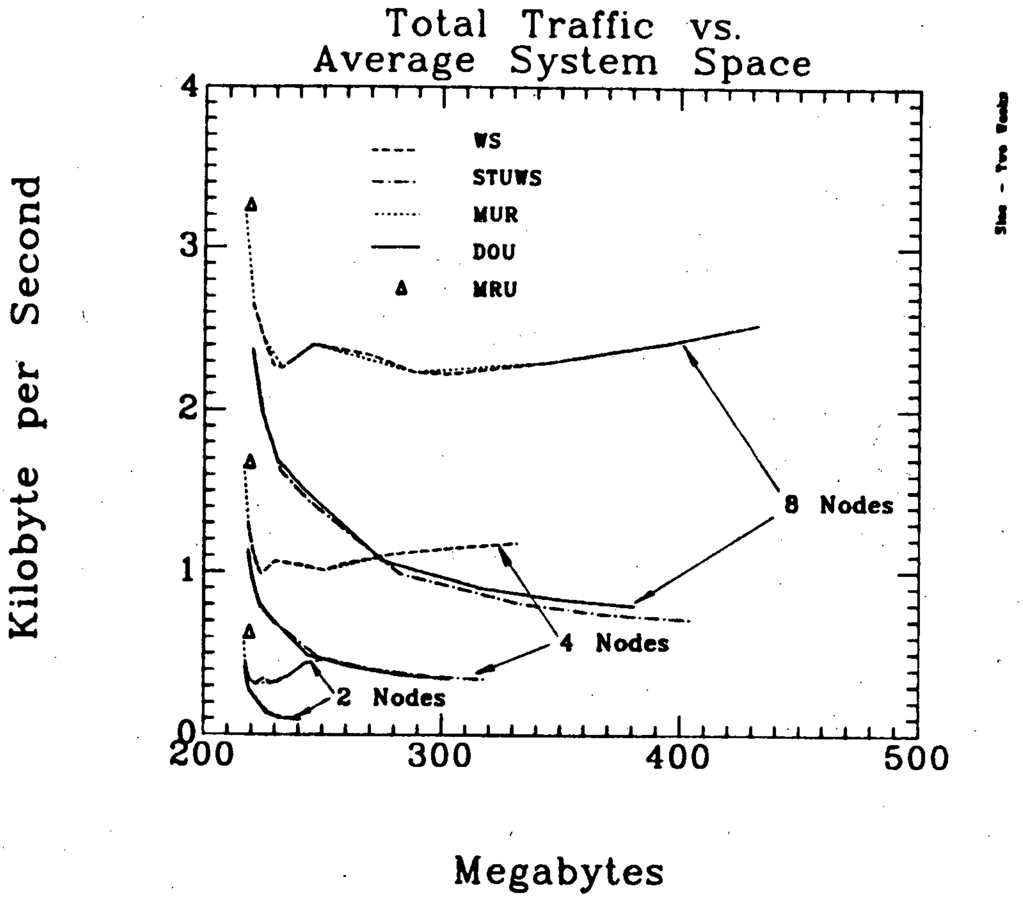

Fig 5.3. Total traffic vs. average system space for the SLAC trace. Each curve corresponds to one value of the update traffic cost. The points on the curve are obtained by varying the demand traffic cost with respect to the storage cost. This in turn determines the maximum time that a copy remains alive after its last reference.

traffic equal performs almost identically. The performance of the policies that either do not consider the update traffic or only consider its mean value is much worse. Basically, the update traffic increases almost linearly with the time the copy is kept alive, and this generates very large amounts of traffic. The difference turns out to be really small between policies that assign different costs to update traffic. Making a decision about the type of policy that should be chosen still depends on the relative costs of update and demand traffic. In several distributed systems, the total bandwidth of the system is much larger than the demand traffic and yet the delay is still too high (this is particularly true of satellite networks). In these cases, it may be advantageous to sustain large volumes of update traffic in order to reduce delay by twenty or thirty percent. The operating point on the curves of update traffic vs. demand traffic is ultimately chosen by the user according to his performance goals.

5.3.2. Hughes Aircraft

The same experiments have been conducted for the Hughes trace. Figures 5.4 and 5.5 show the demand traffic and the total traffic generated by the policies. Table II gives the retention times for the WS policy. The results are essentially identical to those obtained for the SLAC system.

5.4. Conclusions

This chapter has presented a simplified but realistic mode of operation for a distributed file system that handles replicated data. Based on this

Table II. Retention Times for the Working Set Policy (Hughes)

Retention Time (Hours)	Avrg. System Space (Megabytes)	Avrg. Total Traffic (Kbytes/Sec.)	Avrg. Demand Traffic (Kbytes/Sec.)
0.	276.8	2.58	2.58
1.	279.3	2.08	2.01
2.	282.4	1.98	1.48
4.	297.8	1.84	0.97
8.	320.1	1.75	0.72
16.	334.7	1.83	0.62
32.	378.5	1.87	0.48
64.	392.2	1.89	0.41
128.	404.1	1.90	0.37
256.	430.8	1.92	0.32

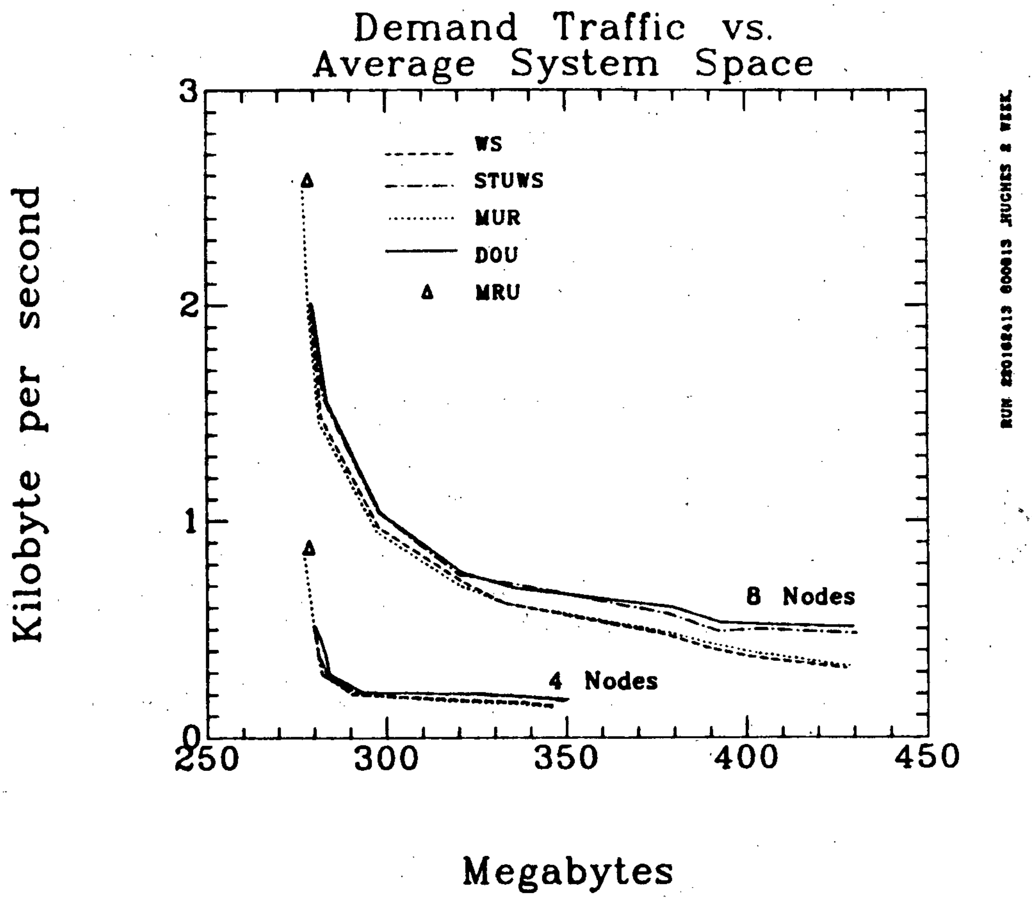

Fig 5.4. Demand traffic vs. average system space for the Hughes trace. Each curve corresponds to one value of the update traffic cost. The points on the curve are obtained by varying the demand traffic cost with respect to the storage cost. remains alive after its last reference.

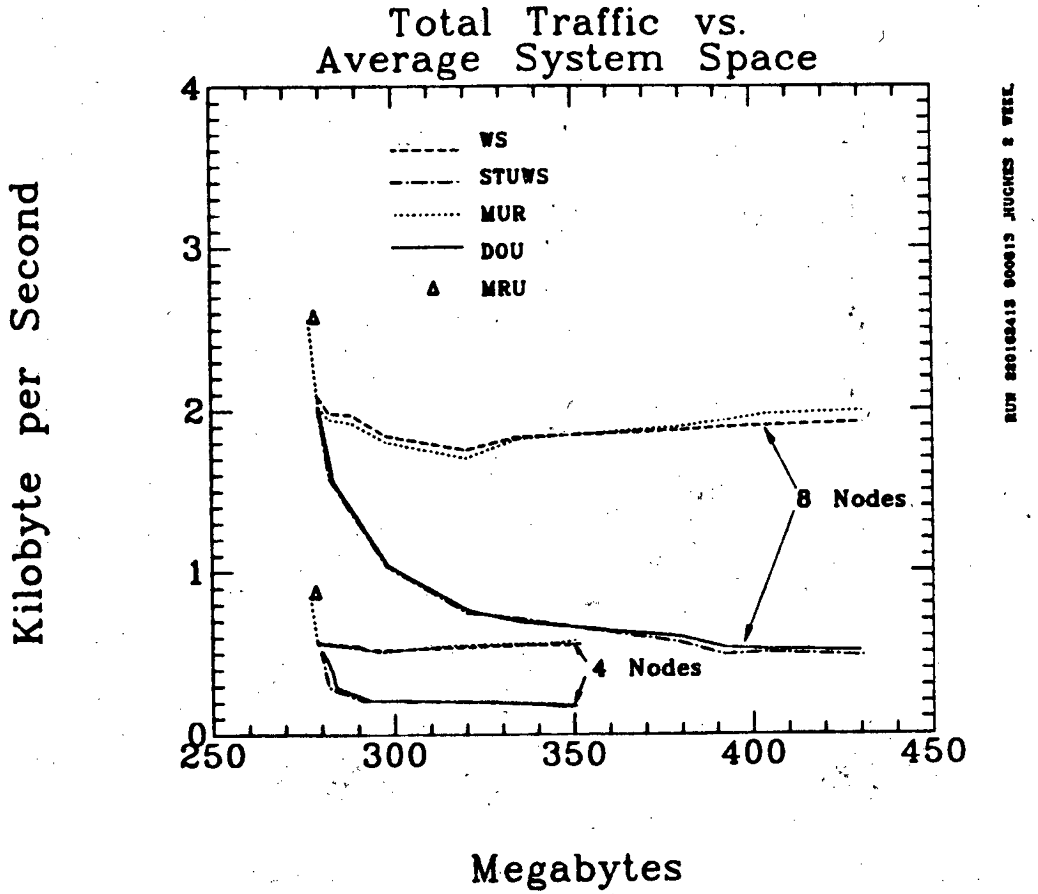

Fig 5.5. Total traffic vs. average system space for the Hughes trace. Each curve corresponds to one value of the update traffic cost. The points on the curve are obtained by varying the demand traffic cost with respect to the storage cost.

mode of operation, the costs of operating the file system have been shown. Our policies are Space-Time Update-Rate Working Set policies. That means that copies that exceed a retention cost based on storage costs and update traffic costs are removed. Using trace-driven simulations, we have shown the operating curves of four distinct policies. Most of the policies achieve a reduction of an order of magnitude in demand traffic and number of remote opens compared to the single-copy policies.

CHAPTER 6

CONCLUSION

6.1. Summary

The management of global (shared) files can have a big impact on the performance of distributed computer systems. In this dissertation, we have devoted most of our effort to the study of the file referencing process and to the development of algorithms to place and migrate files in distributed computer systems.

The topic of assigning files to the nodes of computer networks has received considerable attention in the past. Unfortunately, most of the work in this area has concentrated on the optimization of distributed file systems based on extremely simple models of the workload. Our study is based on traces of activity collected from a number of real systems. This allowed us to analyze the workload and to adapt our algorithms to it.

In Chapter 3 we obtained synthetic distributed systems from the real centralized systems that generated the traces. In particular, we used the SLAC and the Hughes systems. This partitioning of the users of a large centralized computer system into a number of smaller user communities was based on their utilization of shared files. We found that the SLAC system produced user subsets with a higher degree of overlapping (in terms of shared files) than the Hughes Aircraft installation.

Chapters 4 and 5 introduced the file migration policies. First the single copy policies were presented. In this context, we developed a migration policy that is optimal in the sense of minimizing the average network traffic. The existence of extended localities in the use of shared files suggested the use of multiple-copy policies in order to reduce the network traffic even further. We studied a number of policies based on the Working Set model. The best performing policies in terms of generated traffic are those that closely monitor the update traffic and that react to it by deleting the copies that are too expensive to maintain up to date.

6.1. Directions for Future Research

In this final section of the dissertation, we present a list of topics for research that have been suggested by our work.

6.1.1. Extension to Other Systems

The research we have described in this dissertation can be extended in many directions. One obvious direction is to study of the performance of our algorithms in different environments. In particular, the two systems that we have analyzed can be described as scientific centers containing essentially the same hardware and the same file system. It would be interesting to see whether our results carry over to other types of workloads and to other types of operating systems, especially those that encourage sharing of files more than does IBM/OS.

The type of system on which this study should be repeated, though, is a real distributed system. Back in 1977, when the traces that we have used were obtained, there were not many distributed systems in operation and those that existed were too new to be traceable. Today, there are thousands of distributed computer systems in operation, and some of them have sophisticated measurement tools. It would be interesting to see what sorts of workloads are supported by these systems and how our migration algorithms would fare in this environment.

6.1.2. Database Management Systems

There is a great deal of interest in Distributed Database Management Systems. Our results cannot be extended a priori to DDBMS's for two main reasons:

(1) The workload of a DBMS is intrinsically different from that of a file system. The total volume of information is very large and yet users typically access a small fraction of the whole database.

(2) The mode of operation in many DDBMS's is based on the paradigm of transactions. In order to preserve the consistency of the transaction and the integrity of the database, intricate controls and locking mechanisms are used. The result, in a distributed environment, is that a considerable amount of control traffic is needed to maintain the operation of the system. This traffic is such a large portion of the total communications activity that most of our assumptions would have to be revised.

It would be interesting to repeat our experiments based on traces of database activity and under a more appropriate mode of operation, including a two-phase commit protocol and taking into consideration the traffic generated by the locks.

6.1.3. Related Files

Our study has not considered any relations between files. Yet, it is rather intuitive that in any computer installation files are not used independently but rather in well defined groups. This knowledge should be used to improve the performance of distributed file systems. In particular, the delay experienced by the users of one of these groups of files could be drastically reduced by migrating the entire group when one of the members of the group has to be migrated. This hypothesis has to be tested experimentally.

6.1.4. Constrained Design Problem

Our approach to the problem of the assignment of files to the nodes of a computer network has been to assume that there are no constraints in storage space or communications bandwidth, and to measure the needs for these resources in an environment without queueing delays. In some situations, the resources are allocated based on other considerations. It would be interesting to see how our algorithms work in an environment where big bursts of traffic cannot be accommodated.

6.1.5. Multiple-copy Policies

In the realm of single-copy policies, the optimal static assignments should be investigated further.

Our coverage of the variable space, multiple-copy policies has been rather incomplete. In particular, we have not investigated policies that use estimates of interreference times in order to calculate the times copies should be allowed to stay at a given node. Given the positive results obtained by Smith [Smi81b] in his study of hierarchical file migration, such a study should be undertaken.

BIBLIOGRAPHY

[Alc76] A. Alcouffe and G. Muratet, "Optimum Location of Plants," *Management Science* **23**(3) pp. 267-274 (Nov. 1976).

[Art81a] E. Arthurs and B. W. Stuck, "A Theoretical Performance Analysis of Polling and Carrier Sense Collision Detection Communication Systems," *Computer Communication Review* **11**(4) pp. 156-162 (Oct. 1981). Proc. Seventh Data Com. Symp.

[Art81b] H. Pat Artis, *Predicting the Behavior of Secondary Storage Management Systems for IBM Computer Systems*, Bell Laboratories, Piscataway, New Jersey (1981).

[Bad78] D. Z. Badal and G. J. Popek, "A Proposal for Distributed Concurrency Control for Partially Redundant Distributed Data Bases Systems," *Proc. 3rd Berkeley Workshop on Distr. Data Management and Comp. Networks*, pp. 273-288 (August 1978).

[Bad81] D.Z. Badal, "Concurrency Control Overhead or Closer Look at Blockin vs. Nonblocking Concurrency Control Mechanisms," *Proc. 5th Berkeley Workshop on Distr. Data Management and Comp. Networks*, pp. 85-104 (February 1981).

[Bel76] Geneva G. Belford, "Optimization problems in Distributed Data Management," *Proc. Third International Conference on Computer Communication*, pp. 297-301 (3-6 August 1976).

[Ber81] Philip A. Bernstein and Nathan Goodman, "Concurrency Control in Distributed Database Systems," *Comptng. Surveys* **13**(2) pp. 185-222 (June 1981).

[Ber76] Dimitri P. Bertsekas, *Dynamic Programming and Stochastic Control*, Academic Press, Inc., New York (1976).

[Bog] David R. Boggs, John F. Shoch, Edward A. Taft, and Robert M. Metcalfe, *Pup: An Internetwork Architecture*.

[Bok79] S. H. Bokhari, "Dual Processor Scheduling with Dynamic Reassignment," *IEEE Trans. Soft. Eng.* **SE-5**(4) pp. 341-348 (July 1979).

[Buc79] B. P. Buckles and D. M. Hardin, "Partitioning and Allocation of Logical Resources in a Distributed Computing Environment," pp. 247-276 in *Distributed System Design*, ed. D. F. Palmer, (Oct. 1979).

[Bux81] W. Bux, "Local-Area Subnetworks: A Performance Comparison," *IEEE Trans. Comm.* **COM-29**(10) pp. 1465-1473 (October 1981).

[Cas72] R. G. Casey, "Allocation of copies of a file in an information network," *Proc. Spring Joint Comptr. Conf.*, pp. 617-625 AFIPS Press (1972).

[Cas73] R. G. Casey, "Design of Tree Networks for Distributed Data," *Proc. AFIPS*, (1973).

[Cha77] G. A. Champine, "Six Approaches to Distributed Data Bases," *Datamation*, (May 1977).

[Cha76] K. M. Chandy and J. E. Hewes, "Optimal file allocation in Distributed Systems," *Proc. Int. Symp. Computer Performance Modeling Measurement and Evaluation*, pp. 10-13 (March 1976).

[Chu76] Wesley W. Chu, "Performance of File Directory Systems for Data Bases in Star and Distributed Networks," *Proc. NCC*, AFIPS Press (1976).

[Chu80] Wesley W. Chu, Leslie J. Holloway, Min-Tsung Lan, and Kemal Efe, "Task Allocation in Distributed Data Processing," *Computer* **13**(11) pp. 57-69 (November 1980).

[Coa81] K. E. Coates, D. L. Dvorak, and R. M. Watts, "An Overview of BLN: A Bell Laboratories Computing Network," *Computer Communication Review* **11**(4) pp. 224-229 (Oct. 1981). Proc. Seventh Data Com. Symp.

[Cof73] E. G. Coffman and P.J. Denning, *Operating Systems Theory*, Prentice-Hall, Enlewood Cliff, New Jersey (1973).

[Cof80a] E. G. Coffman, Jr., Erol Gelenbe, and Roger C. Wood, "Optimal Replication of Parallel-Read, Sequential-Write Systems," *Performance Eval. Rev.* **9**(2) pp. 209-216 (Summer 1980). Proceedings of Performance 80

[Cof80b] E. G. Coffman, Jr., Erol Gelenbe, and B. Plateau, "Optimization of the Number of Copies in a Distributed Data Base," *Performance Eval. Rev.* **9**(2) pp. 257-263 (Summer 1980). Proceedings of Performance 80

[Cof81] E.G. Coffman, Jr., H.O. Pollak, E. Gelenbe, and R.C. Wood, "An analysis of Parallel-Read Sequential-Write Systems," *Performance Evaluation* **1**(1) pp. 62-69 (January 1981).

[Coo63] L. Cooper, "Location-Allocation Problems," *Operations Research* **11**(3) pp. 331-343 (1963).

[Coo64] L. Cooper, "Heuristic Models for Location-Allocation Problems," *SIAM Rev.* **6**(1)(1964).

[Dav77] J. Davidson, W. Hathaway, J. Postel, N. Mimno, R. Thomas, and D. Walden, "The Arpanet Telnet Protocol: Its Purposes, Principles, Implementation, and Impact on Host Operating System Design," *Proc. 5th Data Comm. Symp.*, (1977).

[Dav79] D. W. Davies, D. Barber, W. L. Price, and C. M. Solomonides, *Computer Networks and Their Protocols*, John Wiley, New York (1979).

[Day80] John D. Day, "Terminal Protocols," *IEEE Trans. Commun.* **28**(4) pp. 585-593 (April 1980).

[Den78] Peter J. Denning and Donald R. Slutz, "Generalized Working Sets for Segment Reference Strings," *Comm. ACM* **21**(9) pp. 750-759 (September 1978).

[Dio80] Jeremy Dion, "The Cambridge File Server," *Operating Syst. Rev.* **14**(4) pp. 26-35 (Oct. 1980).

[Dur78] Gary Durbin, Todd Kinney, Peter Lamasney, Edward Newman, and Edward Syrett, "Guideline on Major Job Accounting Systems: The System Management Facilities (SMF) for IBM Systems under OS/MVT," NBS Special Publication 500-40, U.S. Department of Commerce (October 1978).

[Esw74] K. P. Eswaran, "Placement of records in a file and file allocation in a Computer network," *Proc. IFIPS*, pp. 304-307 (1974).

[Eth81] The Ethernet, "The Ethernet. A Local Area Network. Data Link Layer and Physical Layer Specifications," *Computer Communication Review* 11(3) pp. 20-66 (July 1981).

[Faj73] Roger Fajman and John Borgelt, "WYLBUR: An Interactive Text Editing and Remote Data Entry System," *Comm. ACM* **16**(5) pp. 314-322

(May 1973).

[Fis80] Marshall L. Fisher and Dorit S. Hochbaum, "Database Location in Computer Networks," *J. ACM* **27**(4) pp. 718-735 (October 1980).

[Fol80] H. C. Folts, "Procedures for Circuit-Switched Service in Synchronous Public Data Networks," *IEEE Trans. Comm.* **COM-28**(4) pp. 489-496 (April 1980).

[For62] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, N. J. (1962).

[Gar79] Hector Garcia-Molina, *Performance of Update Algorithms for Replicated Data in a Distributed Database*, Dept. of Computer Science, Stanford University (June 1979). Ph.D. Dissertation

[Gar81] Hector Garcia-Molina, "The Cost of Data Replication," *Computer Communication Review* **11**(4) pp. 193-198 (Oct. 1981). Proc. Seventh Data Com. Symp.

[Gie78] M. Gien, "A File Transfer Protocol (FTP)," *Computer Networks* **2** pp. 312-319 (Spt. 1978).

[Gif79] David K. Gifford, "Weighted Voting for Replicated Data," *Proc. Seventh sumposium on Operating Systems Principles*, pp. 150-162 (Dec. 1979).

[Gop81] Gita Gopal and J. W. Wong, "Delay Analysis of Broadcast Routing in Packet-Switching Networks," *IEEE Trans. Comput.* **C-30**(12) pp. 915-922 (Dec. 1981).

[Gra77] Enrique Grapa and Geneva G. Belford, "Some Theorems to Aid in Solving the File Allocation Problem," *Comm. ACM* **20**(11)(November 1977).

[Gre80] P. E. Green, "An Introduction to Network Architectures and Protocols," *IEEE Trans. Commun.* **28**(4) pp. 413-424 (April 1980).

[Har75] John A. Hartigan, *Clustering Algorithms*, John Wiley & Sons, Inc., New York (1975).

[Hay81] Jeremiah F. Hayes, "Local Distribution in Computer Communications," *IEEE Communication Mag.* **19**(2) pp. 6-15 (March 1981).

[Hob80] Verlin L. Hoberecht, "SNA Function Management," *IEEE Trans. Commun.* **28**(4) pp. 594-603 (April 1980).

[Hol73] E. Holler, "Files in Computer Networks," *First European Workshop on Computer Networks*, pp. 381-396 (April 1973).

[How60] Ronald A. Howard, *Dynamic Programming and Markov Processes*, Technology Press and John Wiley & Sons, Inc., New York (1960).

[How71] Ronald A. Howard, *Dynamic Probabilistic Systems. Volume II: Semi-Markov and Decision Processes*, John Wiley & Sons, Inc., New York (1971).

[Hui81] C. Huitema and I. Valet, "An Experiment on High Speed File Transfer Using Satellite Links," *Computer Communication Review* **11**(4) pp. 254-257 (Oct. 1981). Proc. Seventh Data Com. Symp.

[Hwa80] K. Hwang, B. W. Wah, and F. A. Briggs, *A Hardwired Network of UNIX Computer Systems*. Oct. 25, 1980.

[IBM73] IBM, "OS SMF," GC28-6712-7, IBM (April 1973). Eigth Edition

[IBM78] IBM, "OS/VS2 MVS System Programming Library: System Management Facilities (SMF)," GN28-2903, IBM (May 5, 1978).

[Kol81] Walter H. Kolher, "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems," *Comptng. Surveys*

13(2) pp. 149-184 (June 1981).

[Lam81a]

Simon S. Lam and Y. Luke Lien, "Modeling and Analysis of Flow Controlled Packet Switching Networks," *Computer Communication Review* **11**(4) pp. 98-107 (Oct. 1981). Proc. Seventh Data Com. Symp.

[Lam81b]

S. S. Lam and Y. C. L. Lien, "Congestion Control of Packet Communication Networks by Input Buffer Limits - A Simulation Study," *IEEE Trans. on Comp.* **C-30**(10) pp. 733-742 (October 1981).

[Lee77] Robert P. Lee and Richard R. Muntz, "On the Task Assignmnet Problem for Computer Networks," *Proc. Tenth Hawaii International Conference on System Sciences*, (1977).

[Lev78] K. Dan Levin and Howard Lee Morgan, "A Dynamic Optimization Model for Distributed Databases," *Operations Research* **26**(5) pp. 824-835 (Sept.-Oct. 1978).

[Lin79] W. K. Lin, "Concurrency Control in a Multiple Copy Distributed Database System," *Proc. 4th Berkeley Workshop on Distr. Data Management and Comp. Networks*, pp. 207-220 (August 1979).

[Lud81] G. W. Luderer, H. Che, J. P. Haggerty, P. A. Kirslis, and W. T. Marshall, "A Distributed UNIX System Based on a Virtual Circuit Switch," *Operating Syst. Rev.* **15**(5) pp. 160-168 (Dec. 1981). Proceedings Eight Symposium on Operating Systems Principles

[Lun77] Allen W. Luniewski, *File Allocation in a Distributed System*, M.I.T. Laboratory for Computer Science (December 19, 1977).

[Lun78] Allen W. Luniewski, *Some Results on File Allocation in a Local Network*, M.I.T. Laboratory for Computer Science (March 22, 1978).

[Mah76] Samy Mahmoud and J. S. Riordon, "Optimal Allocation of Resources in Distributed Information Networks," *ACM Trans. Database Systs.* **1**(1) pp. 66-78 (March 76).

[Mar81] Madhav Marathe and Sujit Kumar, "Analytical Models for an Ethernet-Like Local Area Network Link," *Performance Evaluation Review* **10**(3) pp. 205-215 (Fall 1981).

[McQ74] J. M. McQuillan, "Adaptive Routing Algotithms for Distributed Computer Networks," BBN Report No. 2831 (May 1974). (Ph. D. Thesis, Harvard University)

[McQ77] J. M. McQuillan and D. C. Walden, "The ARPA network Design Decisions," *Computer Networks* **1** pp. 243-289 (Aug. 1977).

[McQ78] J. M. McQuillan and Vinton G. Cerf, *A Practical View of Computer Communications Protocols*, The Institute of Electrical and Electronics Engineers, Inc. (1978). Library of Congress No. 78-61492

[Met76] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Comm. ACM* **19** pp. 395-404 (July 1976).

[Min79] T. Minoura, "A New Concurrency Control Algorithm for Distributed Database System," *Proc. 4th Berkeley Workshop on Distr. Data Management and Comp. Networks*, pp. 221-236 (August 1979).

[Mit79] I. Mitrani and K.C. Sevcik, "Evaluating the Trade-off Between Centralized and Distributed Computing," *Systems*, pp. 520-528 , Huntsville, Alabama(Oct. 1979).

[Mor77] Howard L. Morgan and K. Dan Levin, "Optimal Program and Data Locations in Computer Networks," *Comm. ACM* **20**(5) pp. 315-322 (May 1977).

[Pec81] Michael A. Pechura, "Microcomputers as Remote Nodes of a Distributed System," *Comm. ACM* **24**(11) pp. 734-738 (Nov. 1981).

[Ram79] C.V. Ramamoorthy and B. W. Wah, "Data Management in Distributed Data Bases," *Proc. NCC,* pp. 1011-1024 AFIPS Press (1979).

[Rao79] Gururaj S. Rao, Harold S. Stone, and T. C. Hu, "Assignment of Tasks in a Distributed Processor System with Limited Memory," *IEEE Trans. Comptrs.* **C-28**(4)(April 1979).

[Rau77] Ramakrishna Rau, "Properties and Applications of the Least-Recently-Used Stack Model," Technical Report No. 139, Digital Systems Laboratory, Stanford University, Stanford (May 1977).

[Raz80] Rami R. Razouk and Gerald Estrin, "Modeling and Verification of Communication Protocols in SARA: The X.21 Interface," *IEEE Trans. Comptrs.* **C-29**(12) pp. 1038-1051 (December 1980).

[Ric80] James Richardson, *Creating and Using Reduced Tapes.* August 19, 1980.

[Rie79] David R. Ries, *The Effects of Concurrency Control on Database Management System Performance,* Computer Science Dept., University of Californi, Berkeley (April 1979). Ph.D. Dissertation

[Ros73] Lawrence L. Rose and Malcolm H. Gotterer, "A Theory of Dynamic File Management in a Multilevel Store," *International Journal of Computer and Information Sciences* **2**(4) pp. 249-256 (1973).

[Ros75] Lawrence L. Rose and Malcolm H. Gotterer, "An Analysis of File Movement under Dynamic File Management Strategies," *BIT*, pp. 304-313 (1975).

[Ros70] S. M. Ross, *Applied Probability Models with Optimization Applications*, Holden-Day, San Francisco (1970).

[Seg76] A. Segall, "Dynamic File Assignment in a Computer Network," *IEEE Trans. Automatic Control* **AC-21**(2)(April 1976).

[Seg79] A. Segall, "Dynamic File Assignment in a Computer Network-Part II: Decentralized Control," *IEEE Trans. Automatic Control* **AC-24**(5)(October 1979).

[Sho80] John R. Shoch and Jon A. Hupp, "Measured Performance of an Ethernet Local Network," *Comm. ACM* **23**(12) pp. 711-721 (December 1980).

[Smi81a] Alan Jay Smith, "Analysis of Long Term File Reference Patterns for Application to File Migration Algorithms," *IEEE Trans. Soft. Eng.* **7**(4) pp. 403-417 (July 1981).

[Smi81b] Alan Jay Smith, "Long Term File Migration : Development and Evaluation of Algorithms," *CACM* **24**(8) pp. 512-532 (August 1981).

[Sto78] Michael Stonebraker, "Concurrency Control of Multiple Copies of Data in Distributed INGRES," *Proc. 3rd Berkeley Workshop on Distr. Data Management and Comp. Networks*, pp. 235-258 (August 1978).

[Str77] Edward P. Stritter, "File Migration," STAN-CS-77-594 (January, 1977). Ph. D. Dissertation

[Stu80] H. Sturgis, J. Mitchell, and J. Israel, "Issues in the Design and Use of a Distributed File System," *Operating Syst. Rev.* **14**(3) pp. 55-79

(July, 1980).

[Tan81] Andrew S. Tanenbaum, *Computer Networks*, Prentice Hall, Inc., Englewood Cliffs, N.J. (1981).

[Ten81] Richard L. Tenney, Gilbert Falk, and Douglas H. Hunt, "Impact of Satellite Technology on Transport Flow Control," *Computer Communication Review* **11**(4) pp. 248-253 (Oct. 1981). Proc. Seventh Data Com. Symp.

[Tho79] R. Thomas, "A Solution to the Concurrency Control Problem for Multiple Copy Databases," *ACM TODS* **4**(2)(June 1979).

[Thu79] Kenneth J. Thurber and Harvey A. Freeman, "A Bibliography of Local Computer Network Architectures," *Computer Architecture News* **7**(5) pp. 22-27 (February 1979).

[Tob78] Fouad A. Tobagi, Mario Gerla, Richard W. Peebles, and Eric G. Manning, "Modeling and Measurement Techniques in Packet Communications Networks," *Proc. IEEE* **66**(11) pp. 1423-1445 (Nov. 1978).

[Tri80] Kishor S. Trivedi, Robert A. Wagner, and Timothy M. Sigmon, "Optimal Selection of CPU Speed, Device Capabilities, and File Assignments," *J. ACM* **27**(3) pp. 457-473 (July 1980).

[Uns81] Mehmet S. Unsoy and Theresa A. Shanahan, "X.75 Internetworking of Datapac and Telenet," *Computer Communication Review* **11**(4) pp. 232-239 (Oct. 1981). Proc. Seventh Data Com. Symp.

[Vin80] Ilse Vinsin, Calvin Ross, and Ed Russell, *ASP User Guide*, SLAC Computing Services (SCS), Menlo Park, California (24 March 1980).

[Wah79] B. W. Wah, *Data Management in Distributed Data Bases*, University of California, Berkeley (1979). Ph.D. Dissertation

[Wec80] Stu Wecker, "DNA: the Digital Network Architecture," *IEEE Trans. Commun.* **COM-28**(4) pp. 510-526 (April 1980).

[Whi70] Kevin M. Whitney, *Optimal design of message processing and communication systems*, The University of Michigan (1970). Ph. D. Dissertation

[Wil80] Maurice V. Wilkes and Roger M. Needham, "The Cambridge Model Distributed System," *Operating Syst. Rev.* **14**(1) pp. 21-29 (January 1980).

[Won78] J. W. Wong, "Queueing Network Modeling of Computer Communication Networks," *Comptng. Surveys* **10**(3) pp. 343-352 (September 1978).

[Zad71] L. A. Zadeh, "Similarity Relations and Fuzzy Orderings," *Information Sciences*, (3) pp. 177-200 (1971).

[Zim80] H. Zimmermann, "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," *IEEE Trans. Commun.* **28**(4) pp. 425-432 (April 1980).

TECHNICAL INFORMATION DEPARTMENT
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720