Optimization of File Migration in Distributed Systems

Øivind Kure

Department of Industrial Engineering and Operation Research (IEOR),
University of California, Berkeley,
Berkeley, CA 94720

Abstract

This dissertation presents and evaluates several new algorithms that improve the performance of distributed file systems by migrating or copying files as necessary between system nodes. The results are based on analysis of file reference patterns from three commercial installations, modeling, and trace driven simulation.

The first part of the dissertation is an exploratory analysis of how shared user files are referenced. We find that although few files are shared, they are opened frequently, and account for a large fraction of the I/O traffic to user files. The reference pattern to shared files is not easily characterized, and varies widely among files. A batch Poisson process with geometric batch size is determined to be the most appropriate model.

Based on the exploratory analysis, we developed several algorithms for file migration and replication. The algorithms evaluated include those based on our file reference pattern analysis, as well as simple strategies such as static placement and movement on reference, and optimal look-ahead migration and placement. We found that only a few files should be migrated or replicated, but replication or migration can substantially reduce the network traffic (up to 63% for replication and 36% for migration, relative to static placement).

A policy based on a batch Poisson process with geometric batch size has the best performance when replication is not allowed. It uses as decision variables the fraction of a file accessed per open, the number of references from a user, and the number of changes in locality.

By replicating files, the network traffic can be reduced further compared to migration alone (up to 42%). Whether the additional copies should be invalidated or updated when the file is updated depends on the installation and the rules for placing users at nodes. The algorithms with the best performance use the average reference rate, the number of consecutive opens in update mode, and the time since the node started using the file as the decision variables. By comparing our realizable algorithms with optimal unrealizable algorithms, we show that it is unlikely that other migration or replication algorithms can achieve a substantially better performance.

Ackowledgement

I gratefully acknowledge the support and help of my advisor, Alan J. Smith. He kept my research on course and provided excellent advice and help. Most of all I acknowledge and thank my wife, Elin. Without her support, patience, and love I would never have started and could never have finished.

The material presented here is based on research supported in part by the National Science Foundation under grants CCR-8202591 and MIP-8713274, by NASA under consortium agreement NCA2-128, by the State of California under the MICRO program, and by the International Business Machines Corporation, Digital Equipment Corporation, Tandem, Hewlett-Packard, and Signetics.

Table of Contents

Chapter 1
Introduction

Summary

The topic of the thesis is file placement in distributed systems. The first chapter describes the characteristics of distributed systems and outlines the file placement problem and other similar problems. It also surveys related works and provides an overview of the thesis.

1.1. Introduction

This thesis investigates file placement in distributed systems. Such systems consist of several processors connected by a communication network. They range in size from a few physically adjacent processors to many geographically distant ones. Files can be stored at one or more locations in the system. Such a system is useful only if the processors can potentially access all files, regardless of where they are stored, or, even better, if the system has a unified transparent file system. Since there are several storage locations, the processors will have different access paths to the various files. A "distance" can be associated with each path, which may be measured in delay, dollars, response time, or wasted system resources. Ideally, files should be placed at or "near" processors where they are used.

The best placement is therefore a function of the reference pattern to the files, and our analysis is based on traces of file system activity. The scope of the study is limited to systems where users are permanently located at processors, and all jobs submitted by a user will run on the same processor. The research has three goals: 1) to discover the best strategy (static placement versus migration and replication), 2) to develop the best possible placement algorithms, and 3) to measure the quality of the proposed algorithms.

1.2. Background

1.2.1. The Architecture of Distributed Systems

This section is an overview of the distributed systems and their characteristics, particularly distributed file systems, since the goal of file placement is to increase their effectiveness. Distributed systems can roughly be be divided into three groups: closely coupled systems, loosely coupled systems, and wide area networks [Clark78].

In a closely coupled system, the processors share some or all of main memory. The system size is limited to a few meters; otherwise the access delay to memory would make them ineffective. The central issues for closely coupled systems are 1) effective sharing of main memory, 2) processor cooperation on the same task, and 3) cache consistency. File placement is not an issue for these systems since all processors share the I/O buffers and the disks. We will therefore not discuss closely coupled systems.

In loosely coupled systems, the various processors are connected by a high-bandwidth low-latency medium like Ethernet or token rings. The bandwidth of these media is a function of their size, so the length of the networks is limited to a few kilometers. Each processor is independent and has its own main memory. Processors may or may not have their own local disks. The Amoeba system [Tanenb85] is an example of one without local disks. It consists of several Sun workstations connected by an 10 Mbytes-per-second local

area network. The workstations are diskless, and all paging is done over the network. Specialized processors are dedicated to provide file service, data base service, print service, and communication service to the work stations. In addition, the Amoeba system contains a bank of processors that are dynamically allocated to different user tasks, like compiling.

The ITC file service [Satyna85][Morris86] is an example of a system where the workstations have their own disks. ITC offers a shared file system to the campus of Carnegie Mellon University (CMU). It was targeted for 5000 to 10000 personal computers (PC) or workstations, each with its own disk. The goals were to : 1) offer location transparency for the files, 2) allow user mobility, 3) provide security, and 4) design a system that would easily grow in size. To provide easy expansion, the system is organized hierarchally into several clusters. Each cluster consists of several workstations and a cluster server connected by a local area network. The server handles retrieval of shared files and access control. The clusters themselves are connected by a local area network to form a unified campus system. In a large network, the physical security of the network cannot be guaranteed. To guarantee integrity, all network traffic is encrypted. This makes paging over the network nearly impossible, and each workstation must have its own disk. The work stations or PCs can then also function outside the ITC system.

Another variant of loosely coupled systems is a cluster of several larger computers, like the VAX-cluster [Kronen86]. Large IBM installations are also configured in a similar way, with two or more processors loosely coupled through their channels. The cluster consists of a few powerful processors. They are connected to form a transparent system with a unified file system, which makes the cluster more flexible and easier to use than several independent computers. Processing is not in parallel, since each task is only allocated to one processor. In such a system it is no longer economical to provide servers dedicated to retrieval of files. These tasks can better be serviced by the processors themselves. Each processor has its own disks and provides its own file and database services. Hopefully, most file references will be to the processor's local disk, and interactions over the network can be avoided.

The final group of systems is the Wide Area Network (WAN). They are characterized by large geographic size, low bandwidth, and/or long delay. Arpanet is a typical example of a Wide Area Network. It connects a multitude of sites in the U.S. and Europe by leased lines or satellite links [McQuil77]. Some of the leased lines have a bandwidth as low as 56K bytes per second. Arpanet does not provide a unified and location-transparent file system, but it provides separate protocols for file transfers (ftp) and remote login (telnet). Due to the WAN's low bandwidth or high latency, few transparent systems have been built on top of these networks. One exception is an extended version of LOCUS, which is a distributed file system [Sheltz86].

Distributed systems are also classified according to the user interface [Tanenb85][Wupit83]. Tanenbaum uses two classifications, the distributed operating system and the network operating system [Tanenb85]. The classification differs in the transparency of the system. In a distributed operating system, the users are not aware of the distributed nature of the system; it appears as one virtual uniprocessor, where all objects are stored locally. The actual storage location is hidden from the users. Examples of distributed systems are: Cambridge Distributed Computing System [Needha82], Amoeba [Tanenb85], V kernel [Cherit83], and Eden project [Almes85].

In a network operating system the users are aware, often painfully, of the distributed nature of the system. Network operating systems do not offer a location-transparent unified system. The name space is not global, and it is the user's responsibility to locate various objects. All the network operating systems offer are procedures for accessing

remote processors or copying remote objects. A clear distinction is therefore made between local and remote objects. A good example of a network operating system is the ARPA network.

Various hybrids of these two distinct types are also possible. For example, the ITC file system [Satyna85] has a shared file system with transparent locations. However, this is the only service that is location transparent.

1.2.2. Why Distributed Systems

Compared to a centralized system, a distributed system can have more flexibility, lower cost, higher availability, and more computational power. It is, however, necessary to prefix any explanation with "can" or "may". There are many examples of distributed systems that, owing to poor design, have the opposite characteristics.

Flexibility is one of the major advantages of distributed systems. Within certain limits, processors and other servers can be added or removed. The system can therefore easily adapt itself to changes in user demand. A centralized system grows less gracefully; adding new users may result in either congestion or a costly CPU upgrade.

Distributed systems also offer an economic advantage. For many applications, it is cost effective to use several cheap but slow processors, compared to one fast but expensive one. Each task will run slower on the cheaper processors, but several tasks can be handled in parallel. The throughput of the two systems may therefore be comparable. As an example, expressed in dollars/cycle, it can be cheaper to compile n modules on n 1-MIPS processors than to compile them on one n-MIPS processor. It is also better economy to dedicate a fast processor to number crunching, and let the cheaper processors handle more mundane tasks like file editing and data communication.

A distributed system can also offer better availability than a centralized system, and can degrade more gracefully. In a distributed system, if 10% of the processors crash, either 10% of the users are denied access or the capacity of the system is decreased by 10%. In a centralized system, availability is binary. Either all users have access or none. Amoeba [Tanenb85] is an example of a system that degrades gracefully. The system has a pool of processors that are dynamically assigned to different user tasks. Any failure of these processors leaves the system functioning, but the total capacity will be reduced.

1.2.3. File Service

This section outlines the architecture of the file service, which is the process providing users with access to the file system. It is an important service, since almost all activities need to retrieve or store data. An extensive overview of the design issues for the file service are found in articles by Svoboda [Svobod82] and Sturgis [Sturgis80].

To the user there *should* be no conceptual difference between a centralized and a distributed file system. The network should function as an extension of the processor's own I/O channels, and hide the file location from the user. Tanenbaum makes the distinction between traditional file systems, like UNIX [Ritchi74], and robust file systems that offer a higher reliability for the data [Tanenb85]. A traditional file system seldom provides any guarantee for the consistency of the data when the processors crash. Tanenbaum divided the traditional file service into three parts: a disk service, a flat file service, and a directory service [Tanenb85]. The disk service deals with reading and writing disk blocks without any concern for file structure. The flat file service imposes a structure on the disk service. A user views the file as a sequence of bytes or records, and the flat file service maps this view onto the various disk blocks the file is stored on. The directory service maps an object name into an object, so the location of the various objects can be found.

Both centralized and distributed systems have this architecture. However, for distributed systems, the three parts must be built on top of a communication protocol. A communication path must be set up and maintained between the user referencing the file and the processor controlling the disk where the file is stored.

The traditional file system is often augmented by recoverable and serial transactions to make it more robust and reliable. This, however, increases the overhead of each file reference. A transaction is a user-defined collection of I/O operations that logically should be considered a unit. The classical example of a transaction is all I/O operations needed to transfer money between two accounts in a banking system. The amount is first subtracted from one account (a read and a write operation), and then added to the other account (a read and a write operation).

To provide reliable and consistent results, the opens from the various transactions must be synchronized so they do not interfere with each other. Only one transaction at a time can update an object (a file or a record). Otherwise, some of the updates to an object may be overwritten without being seen by any other transaction (the updates are lost). This means that data read in the beginning of the transaction A cannot be changed by another transaction until after transaction A has finished with the data.

Many systems, like a banking system, cannot tolerate lost updates, and the accesses from the transactions are synchronized and ordered serially. Synchronization can be achieved by many different methods [Bernst81][Mullen85], but for brevity we will only discuss the simple scheme of using access locks to the data [Bernst81]. Usually, two types of locks are used, multiple read and exclusive access update. The update lock can only be granted as long as no other transaction has a lock on the data, and no other locks can be granted while the update lock is held. If several different transactions need the same locks, there is a potential for deadlock where each transaction ends up with only a subset of the locks, blocking each other from completion. Any scheme based on access control through locks must therefore implement algorithms for detecting and solving deadlock situations [Chandy83].

In a distributed system, processors will crash independently of each other. In order to make the transactions reliable and robust, it is not sufficient to synchronize the transactions; they must also appear as atomic transactions, i.e., either all I/O operations in a transaction succeed or they all fail and leave the object (file or record) unchanged. Otherwise, if some of the I/O operations succeeded while the others failed, the result would be chaos.

A standard implementation of atomic transactions is to use a two phase commit protocol and stable storage [Bernst81]. Stable storage cannot be corrupted, and data stored there will survive a system crash. One way of implementing stable storage is to write the data sequentially to two different disk locations. A crash during the write operation will then only corrupt one copy of the data, and the other copy can be used for salvage.

With a two phase commit protocol, the intended changes to an object are written out to stable storage, without changing the original object. Once the intended changes are successfully stored, the transaction is ready to commit (the first phase). After the transaction is committed, the object themselves are updated (the second phase). Any failure of the system at this point will not affect the transaction, since the correct state of the objects involved can be reconstructed from the intended changes stored on the stable storage. Aborting the transaction before it is committed will not affect the objects, since they are not updated until after the transaction is committed. Only the intended changes will have to be discarded. With a two phase commit protocol combined with stable storage, the transactions will appear as atomic; regardless of the events in the system

either all updates in a transaction are successful, or none of them take effect.

Not all file services offer all of these services. The WFS [Swineh79] and XDFS [Mitche82] are two examples of file service that span the spectrum of possible combinations. WFS, which offers only the disk service, can only retrieve and store specific file pages. The files are identified by a 32 bit integer file identity, and access control is at the file level. Any additional services must be implemented by the users themselves. XDFS, a file service designed to support research on databases, provides byte level locking and atomic transactions.

1.2.4. Why is File Placement a Problem ?

We are mainly concerned with how file placement affects the performance of file service. As previously discussed, the file system hides the location of a file from the user, offering instead the abstraction that all files are stored at a local disk. This causes a performance penalty. Using a file stored at a remote location (a remote reference) is slower and wastes more system resources than using a file stored at a local disk. The penalty, which varies from system to system, consists of three parts:

There is the delay of the communication itself. The delay consists of two parts, the physical delay and the delay due to a limited bandwidth. Owing to geographic distance, any transmission will be delayed by a time equal to the distance divided by the propagation speed of the network (usually close to the speed of light). This will usually only be an issue for Wide Area Networks. For systems built on top of Local Area Networks (LAN), it can be ignored, since the distance between the processors is less than a few kilometers. The delay owing to the limited bandwidth is often substantially larger than the physical delay. The transmission delay is equal to file size / band width, and for networks with limited bandwidth (like Arpanet) may be of the order of seconds even for small files (100k). Again this is more of an issue for wide area networks than for local areas networks, which often have bandwidths from 3M bits to 10M bits per second.

There is, however, a substantial overhead associated with remote file reference. A communication path must be set up and maintained between the processor using the file (the requesting node) and the processor controlling the disk where the file is stored (the storage node). Setting up and maintaining a communication path is more costly in a wide area network owing to both a lower bandwidth and a higher error rate. The overhead is substantial even in local area networks using protocols that are tailored for remote operations. For example, in V-kernel, a distributed operating system for diskless workstations [Cherit83], a local page read from memory takes 1.31 msec, while a remote one takes 2.50 msec at the requesting node and 3.50 msec at the storage node [Cherit83]. A substantial part of this overhead is spent copying data through the different interfaces, both at the requesting node and the storage node.

Referencing a file remotely not only increases the response time for the user, but also affects all the other users. The CPU load of the file host increases, and the response time for all users placed at that node becomes larger. The chances for disk conflicts increase. Remote file references therefore reduce the independence of the users; the actions of one user are more likely to affect other users.

Since there is a performance cost of remote file references, the placement of files relative to where they will be used is an important performance problem. Ideally, files should be placed at or near the location where they are used, because is not only beneficial for the user opening the file, but also for the whole system.

Only files used by more than one location are difficult to place. One obvious solution is to create a copy of the file at all locations. This is only a feasible solution for files that

are never updated. If the file is updated, all copies must also be updated. Where and when to create a copy becomes a trade off between the expected benefit of local accesses and the cost of creating and maintaining a copy. File placement is therefore an important performance problem with no simple solution.

1.2.5. Related Performance Problem

File placement is only a problem because a remote reference is more costly than a local one. Similar issues occur repeatedly throughout the memory hierarchy. An overview of these issues is found in Smith's paper [Smith85b].

At the top of the memory hierarchy are the CPU registers followed by the CPU cache. The next level down is the main memory, followed by the disks, and then the tape archives. The top level has the shortest access delay, but also the smallest storage capacity. Not all data can be stored at the uppermost level, and the speed of most applications will be limited by the speed at which the data can be brought in from the underlying levels in the memory hierarchy.

Most solutions to this problem take advantage of the inherent locality [Denning72][Blevin76][Denning80] of references to data stored at the various levels of the memory hierarchy. Such locality has been shown at a memory level [Denning72] and at file level [Smith81b][Ouster85][Majuma86]. The locality is both temporal and spatial. Temporal locality means that an object accessed recently is likely to be accessed again. Spatial locality means that objects adjacent to the current one are likely to be used. The effect of the access delay can be limited by caching the most recently used data on the faster access medium. Most references will then find their target already in the cache, and there will be no need to wait while the data is retrieved. This is the motivation behind CPU caches, which keep a small amount of the most recently instructions and data in a fast memory located close to the CPU [Smith82].

The same idea can also be utilized for file service; each disk should keep some of the most recently used disk blocks in the memory of the controller [Smith85]. Owing to the locality of disk block references, some of the disk block requests can be serviced from the memory, and the slow access to the disk is avoided. In a distributed system, this idea can also be expanded to include the requesting node [Schroe85]. Some of the disk block requests can then be served from a local buffer at the requesting processor, avoiding most of the remote accesses.

Buffer caches at the requesting node raise the additional question of consistency [Thomps87]. If several nodes have used the same file, some of the blocks may exist concurrently in several buffer caches. Any updates to the file must not only be reflected in the version of the file stored on the disk, but also in all caches storing blocks of the file. The blocks must either be updated or erased. This is similar to the problem of maintaining consistency among copies of a file, and it is an important research problem. Other research areas are disk cache size [Smith85][Ouster85] and transfer block sizes [Lazows84]. The best block size is a function of spatial locality and the communication cost; increasing the block size becomes a trade off between wasting transfer capacity on data that will never be used, and decreasing response time by suppling data before it is requested.

The disk cache tries to minimize the delay between the main memory and disk. A similar problem, hierarchical file migration, tries to minimize the delay between disks and tape archives. Archives are slower, but have larger storage capacity. The goal is to minimize a weighted sum of the access delay and the storage cost by moving files between the tape storage and the disk as the access patterns change [Smith81][Smith81b][Smith82b]. The file placement problem is closely related to the

hierarchical file migration problem, since the latter can be viewed as a file placement problem with only two possible storage locations, near (at the disk) or remote (at the tape archive).

File placement also affects the utilization of system resources. It is therefore affected by other strategies for maximizing the utilization of these resources, in particular strategies for load sharing.

The goal of load sharing is to maximize the CPU utilization [Zhou87]. With uneven user activity, some CPUs may be under-utilized while others are over-utilized. Load sharing tries to correct this imbalance by redirecting jobs to the under-utilized CPUs. A good load-sharing strategy should optimize total system performance, treat users fairly, and be failure resistant [Wang85]. Unfortunately, in a decentralized system no processor has full knowledge of the status of all other processors. A perfect redirection of the jobs is therefore impossible. Many different load sharing algorithms have been suggested. They differ in their architecture (centralized versus decentralized solutions), strategy (server initiated versus source initiated)[Wang85], the parameters used as load indicators, and balancing policy (deterministic routing [Lo84] versus probabilistic routing [Chou80]).

Load sharing and file placement are competing strategies. File placement utilizes the locality in users' file references by placing files close to where they are used. The underlying assumption is that users will remain at the same location, so that future references will come from the same location. Load sharing tends to break up the locality. Various jobs from the same user will be placed at different processors, destroying any geographical locality among the references. Load sharing can even out the load, but it will increase the cost of a file reference. Similarly, file placement can reduce the number of remote references, but at the cost of increasing the likelihood of uneven CPU loads.

1.3. Previous Work on File Placement

A large body of literature deals with file placement or database placement in distributed systems. Even though there are differences between database and file placement, they can be viewed as the same problem, and the current work makes no distinction between them. They differ mostly in underlying assumptions and some of their properties; databases are larger, have a smaller unit of reference (a record compared to a file), have a more constant reference pattern (on the database level), and are rarely erased. However, the approach used to solve the database placement problem is identical to the file placement problem, which is our research topic. This section presents a model for formulating the file placement problem, and it surveys related work.

1.3.1. System Model

File placement is a concern only when the processors in a distributed system have different paths to the various storage objects. The system can be modeled by a graph, with two types of nodes: processors and storage locations. The edges in the graph are the different paths between the processors and the storage locations. There is a cost associated with using each of these paths. It is not necessarily constant; for example, the response time depends on the actual load on the path at the time a file is referenced. The possible actions a user can take are: 1) moving a file to a new location, 2) creating a copy at a new location, 3) reading part or all of a file, and 4) updating part or all of a file, which implies that all copies of the file must be updated. These actions will not necessarily lead to the same cost for using a particular path.

The model may incorporate constraints on other system resources, such as constraints on storage capacity at a particular location, bandwidth constraints on the

communication links, or restrictions on where programs and data can be stored (either for security or reliability reasons). The file placement problem is thus described as finding the placement that will minimize the total cost of referencing the files subject to some constraints. Due to complexity, it is usually impossible to find the optimal solution. Instead the goal is to find a satisfactory or reasonable solution.

1.3.2. Survey of Previous Work

Only file placement and the related area of hierarchal file migration are surveyed. The review is short, since our approach is not related to most of the previous work in the area. Most of the previous work is based on incorrect assumptions, and the approaches used are not applicable under the correct assumptions.

The file placement problem is similar to many other allocation problems. Ramamoorthy and Wah showed that file placement is isomorphic to the single commodity warehouse location problem [Ramamo83]. A similar problem is how to allocate tasks to the processors in a distributed system. The processors communicate with each other along paths that may have different costs, and the goal is to find the task allocation that will minimize the total cost of inter-task communication and process time [Rao79][Price81][Lo84][Nikola86]. This leads to a problem formulation nearly identical to the one for file placement. Neither of these areas are included in the review, since the approaches used do not differ substantially from those found in the file placement literature.

There are basically two file placement strategies, static and dynamic. Static placement assumes files will not moved from their storage location. A more advanced strategy, dynamic placement, allows files to be moved as the reference pattern changes.

1.3.3. Static Placement

Dowdy et al. have given an extensive overview of static placement strategies [Dowdy82]. A similar overview was published by Wah [Wah84]. Dowdy et al. subdivided the static placement strategies into two groups based on the type of the object function. One group contained all formulations in which the cost of using a path is independent of the action of other users. The cost is typically measured in dollars. If the files can be replicated, even a simple formulation with node dependent cost and no constraints leads to an integer programming formulation [Casey72] [1]. It is therefore a N-P complete problem, and only for files shared by a reasonable number of users is it possible to find the optimal solution.

Two different lines of investigation are usually followed, either developing heuristics for solving the integer programming formulation [Fisher80] [Jenny82] or developing more accurate models. Chu added constraints on the storage available on each node, the availability, and the response time [Chu73]. This make the problem even less tractable from a computational perspective. Morgan [Morgan77] extended the model to account for the dependency between files and programs, while Dutta [Dutta82] proposed extensions to more accurately model the overhead cost of updating a file, which had not been done in

[1] The object function is:

$$\min \sum_i \left[\sum_{j \neq i} c_{ji}^u \lambda_j^u + c_i^o \right] x_i + \sum_i c_i^r \lambda_i^r (1 - x_i)$$

where λ is the reference rate, c_{ij}^u is the cost of updating node j when node i references the file, c_i^r is the cost of a remote open from node i, and x_i is 1 if node i has a copy of the file, otherwise 0.

previous formulations. This formulation was further extended by Ceri to include the difference in overhead between managing local and remote resources [Ceri82].

There is also a series of papers aimed at the design phase of the system [Mahmou76] [Chen80][Irani82][Laning83][Gavish86]. They considered the simultaneous problem of file placement and system design. The best file placement is a function of the topology and the capacity of the processors and the network, while the optimal topology and capacity assignment is a function of the workload for the system. The two problem must therefore be solved simultaneously.

Mahmoud used an object function that minimized the sum of the communication cost and the storage cost, subject to constraints on response time and availability [Mahmou76]. The resulting formulation is an integer programming formulation with non-linear constraints, and it must be solved by heuristic procedures. Mahmoud's heuristic used a feasible solution as a starting point. The search for a solution with lower cost was restricted to the set of feasible solutions that could be reached by either creating or erasing a copy. The heuristic will therefore only produce local optima.

The model was extended by Chen, who included the cost of the CPU capacity in the object function, while excluding the constraints on availability and response time [Chen80]. The formulation is then an integer programming problem that can be solved by branch-and-bound technique. Such an approach is only feasible for small problems, due to computational complexity. Gavish analyzed the model further and proposed heuristics based on Lagrangian relaxation [Gavish86].

Irani et al. combined the two previous formulations and suggested their own heuristics [Irani82]. The formulation was extended further by Laning, who included the effect of adaptive routing on the optimal placement [Laning83]. The latter extension is only of interest for store-and-forward networks, which typically are Wide area Networks.

The second group of static file placement formulations uses delay as the object function [Hugh73][Chen73][Kleinr76][Foster81][Woodsi86]. The best placement must be found for all files simultaneously, since the delay on a particular path depends on the number of users utilizing it. The various models have different assumptions about the topology of the system, but all of them assume files are referenced according to a Poisson process. The file assignment problem is first solved by finding the traffic flow that will minimize delay. The second stage is then to find a file placement that will result in the desired traffic flow. For general topologies the problem of matching file placement to a particular traffic flow is an integer programming problem.

1.3.3.1. Critique of the Work on Static File Placement

The major weakness of the static file placement approach is the assumptions used. All of them assume time-invariant reference rates. This is not a valid assumption, since, as we will show later, the reference rates are not constant over time. Static placement is therefore not the optimal solution, and it may not even be a good solution.

Time-invariant reference rates may be an adequate assumption in a system design formulation [Mahmou76][Chen80][Irani82][Laning83][Gavish86]. At design time, little is known about the detailed reference pattern, and the purpose of the formulation is to include the effects file placement and system design will have on each other. Whether the underlying assumptions for the reference rate are correct is less important.

Time-invariant reference rates may be a correct assumption for placement of databases. Most short-term changes in reference rates will not affect placement, since the size of the database is enormous compared to the amount of data accessed by each

transaction. Only long-term changes in the reference pattern can affect placement. All other changes can be neglected, since the cost of moving the database to take advantage of any short-term variation would overshadow the benefit. Such an argument is not valid for file placement. The reference patterns to files are less stable than those to databases, and the fraction of a file accessed per open is much larger. It may therefore be beneficial to move a file to take advantage of changes in the reference pattern.

Time-invariant rates are not an acceptable assumption for file placement in an existing system. The reference rates vary over time, and the placement scheme should take this into account. Yet, under time-invariant reference rates, any dynamic placement scheme is by default non-optimal. As long as the reference rates are assumed to remain constant, the best static placement will remain the best placement at any other time. Dynamic placement algorithms are by assumption non-optimal, and need not be considered. The result is that a large class of potentially interesting placement schemes are ignored based on an unverified assumption.

The static file placement approach also neglects the problem of incorporating new files and removing of old ones. As files are erased and created, the file placement must be recalculated. This implies that files must be moved to new locations. Such moves incur a cost that cannot be incorporated into a static placement formulation.

1.3.4. Dynamic Placement

With dynamic file placement strategies, files can be moved to different locations as reference patterns change. These strategies have not been researched as extensively as static placement strategies. Since the various dynamic approaches have few common features , each one is critiqued as it is presented. What has been done can be divided into two approaches: a dynamic programming approach and a heuristic approach.

Segal's work belongs in the first category [Segal76][Segal79]. He assumed that nodes referenced files according to a Poisson process, with time varying rates that changed according to an underlying Markov chain. We can think of the states in this chain as representing levels of user interest, such as active and passive. With time variant rates, static placement is no longer the optimal strategy. Instead, the files should be moved around as the users' interest in them changes. Segal found the optimal strategy by solving a stochastic dynamic programming problem.

Segal's work is interesting, and his model of how users reference files is plausible. Unfortunately, his approach is difficult to implement. He assumed the transition probabilities of the underlying chain are known. Yet, in any practical implementation, the transition probabilities and the rates must be estimated. These probabilities can only be estimated by observing the changes in the reference rates, and they are therefore difficult to estimate. Even though there are procedures for doing so, they are not easily implemented on any large scale file system.

Levin and Morgan suggested a simpler approach [Levin76]. They assumed reference rates that changed deterministically over time, i.e., when the rates would change and their new values were known for the whole time horizon of the problem. This leads to a dynamic programming formulation. To reduce the computational complexity, they used a branch-and-bound technique that limited the search tree. The weakness with their work is again the assumption used. The future reference rates are seldom known deterministicly. Instead, only the stochastic properties of future reference rates may be known, and Morgan and Levin's approach is therefore of less interest for file placement.

Three heuristic studies are reviewed here, an analytical study [Ruan87], a simulation study [Dowdy84], and study based on trace driven simulation [Porcar82]. Ruan and

Tischy [Ruan87] analyzed some heuristics for replication of files in a distributed environment. They investigated remote access, replication before a file is used, and demand replication where the file is replicated to the user referencing it. In their model, files are opened according to a Markov chain where the transition probabilities are independent of who the current user is. The reference model is not validated, but it is a plausible one. In many aspects it is similar to the one we propose. We have, however, chosen to analyze several other replication schemes, since we do not believe demand replication is the best possible strategy. Unfortunately, the analytical approach cannot be extended to these replication schemes.

Dowdy et al. used a simulation model as the main tool [Dowdy84], but they also compared the results to analytical solutions. The system considered is a LAN with one file server and several work stations, each with its own disk. The decisions they analyze are when to move a file to the work station and where to place it after the work station has finished with the file. The decision rules are threshold rules, where a file is only moved to a work station that has referenced it more than n times in the last t time units. All work stations have identical stochastic reference patterns to a file. This is an unrealistic assumption, since users most likely will have different reference pattern to the same file. We found this to be true for the traces in our analysis. A large set of parameters is used to formulate the model, and the underlying assumptions are not clearly stated, but most likely they include geometric distributions of the number of blocks accessed per open and for the file size. Neither of these assumptions is validated. In addition, it is difficult to extrapolate the results to other systems, since the model relies on a large set of parameters.

Porcar used a different approach [Porcar82]. The basis for his analysis is traces of file system activity. The reference patterns were first characterized, and then several heuristic file placement strategies were proposed. He analyzed demand migration, several demand replication schemes, and a migration scheme based on a two-state Markov chain model, whose states were the primary location and all other locations. Finally, the performance of the various schemes was tested through trace-driven simulation. Part of his work also explored different schemes for placing users at the various processors.

Porcar's analysis is mostly restricted to demand type algorithms, where the file is moved or replicated each time a different location starts using it. Our approach is in many aspect similar to the one Porcar used. However, his work lacks an analysis of the stochastic properties. We therefore disagree on the appropriate model for how files are referenced. Naturally, the algorithms we investigate are different. A broader range of algorithms are included in our analysis, in particular algorithms that adapt themselves to the observed reference patterns.

1.3.5. Hierarchical File Migration

Hierarchical file migration can be viewed as a file placement problem with only two locations, either near (at the disk) or remote (at the tape archive)[Smith81b][Smith82b][Stritt77][Revell75]. The strategies used in the hierarchical file migration are therefore too limited for file placement. However, the approach used to investigate hierarchical file migration is interesting, since the problems have many common aspects. Research dealing mainly with implementation experience [Lawrie82] is not reviewed.

The approach in the current work is in many aspects similar to the one used by Smith [Smith78][Smith81b][Smith82b]. He used actual work loads as the basis for the analysis and to validate the performance of the various heuristics he proposed. The traces

Smith used had low granularity; he could only determine whether a file had been referenced on a particular day. The reference patterns were first analyzed to characterize the stochastic properties of the reference process [Smith81b]. These characteristics were then translated into reasonable heuristics [Smith82b]. The heuristics differ in the characteristics they are based on: the most elaborate ones are based on the empirical distribution for the time between references, while others use only the expected time between references. The latter are easier to implement, but they had a worse performance. Finally, Smith used trace driven simulation to measure the performance of the algorithms he proposed. To measure the quality of the heuristics, he used the optimal migration scheme given full knowledge of future reference (the optimal look-ahead algorithm).

1.4. Approach

This section contains an overview of the approach used in this thesis. It outlines the research goals, the underlying system model, the method used, and the assumptions we made.

1.4.1. Research Goals

The overall goal is to explore the best possible file placement strategies in a distributed system. In order to reach this goal several questions must be considered. One such question concerns the form of the best strategy. Most previous work assumes static placement is the best strategy. This is based on the invalid assumption that reference rates are time invariant. (This will be shown in the next two chapters.) However, file reference patterns change over time, and a static file placement may not be the best possible strategy. Instead, the performance may improve if files are allowed to migrate to new locations. One of the subgoals is therefore to investigate the advantage migration offers over static placement. The research for this subgoal has two parts. One is to measure the advantage of migration. The second is to design dynamic placement schemes that will work.

Performance may further be improved by permitting the replication of files at several storage locations. This further broadens out placement options. For example, migration can be viewed as replicating a file at a new location and erasing the old copy. The investigation of replication schemes has the same goals as the one for migration schemes: first to measure the advantage replication offers over migration or static placement, then to develop schemes that actually improve the performance.

The disadvantage of replication is that any updates to the file must be transferred to all copies of the file. The copies must contain the same version of the file, which means that updates must be sent to all copies. An important research question is then how to maintain consistency. The choices are either to update all copies, erase all additional copies, or update some copies and erase the remaining ones.

To summarize, the overall research goal is to develop good placement schemes for static file placement, file migration, and file replication. This includes finding a reasonable characterization of the file reference process, good decision variables, and reasonable estimation schemes for the various parameters. Using the term "reasonable" implies that the quality of the proposed algorithms must also be analyzed. To reach the overall research goal it is necessary to analyze the difference between the three placement strategies, i.e., static placement, placement allowing migration, and placement allowing replication.

1.4.2. System Model

This section outlines the system model used as the background for our problem definition. The system considered here is loosely coupled, where processors have their own local disks. Processors are connected by a high-bandwidth, low-latency network like Ethernet or token ring, and they share a unified file system. The files in the system are partitioned onto the disks of the various processors.

Each file can be associated with a *file host*, which is the processor controlling the disk where the file is stored. An open is characterized by a two-element vector, where the elements are the identity of the processor issuing the open and the identity of the file host. Opens to a file at the processor's own disk are called local opens, while opens to files stored at other processors' local disks are called remote opens. The two systems mentioned previously, the ITC file system [Satyan85] and a cluster of powerful processors, fit this description.

There is no conceptual difference between a remote and a local reference. In a remote reference, the network and the file host operate as an extension of the processor's own I/O channels. Remote accesses are costly because they waste system resources. A remote access implies opening and managing a communication path with a substantial overhead both for the processor itself and the file host. Even though the delay is insignificant compared to the delay of disk retrieval, the waste of CPU resources cannot be ignored. A remote access is also not "fair", since the file host is delayed by retrieving the file from its disk, and by managing its part of the network transfer. The added I/O load also increases the congestion on the file host.

1.4.2.1. Measure of Performance

Many different measures can be used to evaluate the performance of file placement algorithms. Among these are user response time, system throughput, number of packets sent, number of bytes sent, cost, storage space used, or file server activity. Which one to choose depends on the scope of the analysis and the features of the system that is analyzed. For example, in a system designed to serve interactive database transactions the delay will typically be the important performance measure, while the throughput may be a more interesting measure for batch oriented systems.

For our purpose, the network traffic measured in bytes is the most appropriate performance measure; it is a system independent measure, and it is a direct measure of the events that cause performance degradation. As Cherition [Cherit83] pointed out, most of the delay of a remote operation is due to copying data through various interfaces. This wastes CPU time in proportion to the number of bytes copied.

Network traffic is also independent of any system parameters like network bandwidth and processor speed. The results of the analysis are therefore not restricted to a particular system configuration. More important, the measure is independent of the workload on the CPUs and the network. Measures like delay or throughput depend on the workload for the CPU and the communication network. These metrics are therefore useless unless the workload of the CPUs and the communication network is specified. This makes delay or throughput unsuitable for our purpose, since the goal is to analyze file placement independent of specific systems and workload assumptions.

Any effect from the timing of the various file system actions cannot be accounted for when network traffic is used as a performance measure. For example, moving n files simultaneously has the same performance as moving the same n files sequentially. In systems using delay or throughput as measures, this may not be true;
n transfers in parallel can cause a longer delay than n sequentially and well spaced

transfers. Since network traffic does not account for the timing of the various actions, the effect on the performance of dependencies between reference patterns to different files cannot be measured with this metric. The placement of a file will then be independent of the placement of all other files. In spite of this disadvantage, we use the network traffic as the performance measure, since it is system and work-load independent. Using the network traffic counted in bytes as a performance measure also partially accounts for the effect any overhead for each data block transferred over the network; the sum of such an overhead for a remote open or file transfer will be proportional with the number of blocks transferred, and therefore also approximately proportional with the number of bytes transferred. This is true only for remote opens and file transfers consisting of many blocks sent over the network. The performance measure does not include the overhead of setting up a remote open or a file transfer, since these types of overhead are system dependent. However, we measure several performance parameters for the various algorithms (for example the number of remote opens and file transfers), so the algorithms' performance can be judged by different measures.

1.4.3. Problem Formulation

The goal of the investigation is to develop placement schemes that will minimize the total cost of using the files in the system. Relative to each processor, a file can have only two positions, either near (at the processor's own disk) or remote (at another processor's disk). For a file shared by n nodes, each with its own disk, there are 2^n-1 possible placements for the up to n possible copies of the file that can exist. The cost of using a file at a remote location is the same regardless where it is stored, since we are using network traffic to measure performance. The cost of a local reference is 0.0, since no network traffic is created.

The distributed system considered here can formally be modeled by a fully connected graph. Each node represents a processor with a local disk. The edges, which all have the same cost, represent the connections between the processors. With the metric we have chosen, the cost is proportional to the number of bytes transferred over the network. The goal is to find the file placement with the lowest total cost for a particular file reference pattern.

1.4.4. Methodology

The approach to the file placement problem is similar to the ones used by Smith [Smith81b][Smith78] and Porcar [Porcar82]. We believe actual workloads must be the basis for any research on file placement, since the best file placement is a direct function of how files are used.

Using actual workloads has some disadvantages. A workload is only a snapshot of the activity at a particular installation, and it is therefore not necessarily representative of a typical workload. In addition, the results of the analysis are often difficult to extrapolate to systems other than those the workloads were obtained from.

The alternative is to base the investigation on a plausible model of user behavior. If the model is representative, the results of the analysis can be more general. Examples of this approach is found in work by Segal [Segal76], Bassioun [Bassio84], Dandamundi [Dandam85], and Mincer-Daszkieviez [Mincer84]. However, models are often chosen for their simplicity with little regard for whether they are representative of the actual process.

The current analysis is based on actual workloads, but to achieve generality, workloads from three different environments are used. These workloads were used to derive models of how files actually are used. The models are simple but plausible, and they

reproduce the observed behavior. These models are then used to formulate different placement strategies.

1.4.5. Assumptions

We assume there are no storage or bandwidth constraints. The placement of each file can therefore be evaluated independently. Further, we assume that each user is permanently placed at a particular node, and that all jobs from a user will run on the same processor. The goal is to develop the best possible placement. This can only be done by taking advantage of the geographical locality among the references to a file. To do so, a user must remain in the same location. Only when the benefit of keeping users at a fixed location has been investigated is it possible to judge the advantage of moving users around to achieve load sharing.

In the system simulated, each user is permanently allocated to a processor (a node) in the system. All jobs originating from a particular user are routed to and run on the same processor. Different allocation schemes for the users are analyzed, and they are discussed in more detail in Chapters 4 and 5.

The scope of the thesis is restricted to user files, and system files are excluded. This is done for two reasons. The reference pattern to user files and to system files are different, so the placement schemes are different. In addition, our approach is only valid for user files. The traces are obtained from systems with centralized file systems. The traces are, however, representative for the reference pattern to user files in a distributed system, since the reference pattern to a centralized and a transparent distributed system will be the same. This is not true for system files, since the architectures of the centralized and the distributed systems are different.

Only files accessed by at least two users during the file's lifetime are included in the simulation. Files accessed by only one user need not generate any network traffic when stored at the node where the user is residing. This is a simplification, since files accessed by only one user during a trace may have been created by a different user before this trace started. In the simulation, all accesses to these files will appear as local, while in reality they may either have been remote or caused the file to be moved. In the traces most of the files referenced by a single user were created and scratched during the trace, so the error of excluding them should be minimal.

1.4.6. Overview

The first step is an exploratory analysis of the file reference pattern found at three different installations. The goal is to find the features that can be exploited by a placement algorithm. The traces are therefore analyzed by size, sharing pattern, and content. This exploratory analysis is discussed in Chapter 2. The same chapter also contains a detailed description of the traces themselves. We find that only a few files are shared, but the shared files represent a disproportional fraction of the I/O traffic to user files (up to 47%). The shared files do not have a uniform behavior. They vary in size, number of opens per file and user, number of users sharing a file, and the time between opens. No single set of fixed parameters common to all files can therefore adequately describe how shared files are referenced.

Chapter 3 investigates the stochastic properties of a user's file reference process. The stochastic properties are used to develop appropriate models of the reference process. Several models are discussed in order to find the most appropriate one, that is, the simplest one that has most or all of the observed properties. This model is used in Chapters 4 and 5 to develop the "best" possible file migration and file replication algorithms. We

find that the time between opens from a user to a particular file can be modeled by a renewal process with a high coefficient of variation. The number of consecutive opens from the same user to a file can be modeled by a geometric distribution. A batch Poisson model with geometric batch size is the most appropriate model. In this model, users issue batches of opens to file, and the time between opens has an exponential distribution.

Chapters 4 and 5 discuss different file placement strategies. Chapter 4 contains an analysis of strategies for file systems that do not allow replication of files, while Chapter 5 analyzes strategies for file systems that allow replication.

The research emphasizes strategies that adapt themselves to the reference patterns observed at the various installations. The same approach is used in both Chapter 4 and 5. The strategies are formulated based on the discussion of the various models in Chapter 3. The performance of these strategies is then tested with trace driven simulation. To serve as reference points several simpler strategies were also simulated, usually either static or demand driven strategies.

This approach, however, cannot measure the quality of the algorithms. The placement algorithms are based on simple models. Therefore algorithms with better performance may exist. The interesting question is, how much better? To help answer this question, we calculated the optimal look-ahead strategy. The optimal look-ahead is the best possible placement if the future reference pattern is known. The performance of look-ahead strategy is a lower bound for any placement algorithm, since it is based on full knowledge of future references. The difference between this bound and the performance of the various strategies is then a measure of the quality of the algorithms. The optimal look-ahead strategies also provide a measure of the advantage of migration over static placement, and replication over migration. The look-ahead algorithms are therefore important parts of Chapter 4 and 5.

In Chapter 4 we conclude that the migration algorithm based on a batch Poisson model with geometric batch size has the best performance. The decision variables used by this algorithm are the fraction of a file accessed per open, the number of references from a user, and the number of changes in locality. In the last section of the chapter, we justify why it is unlikely that other algorithms can achieve a substantially better performance.

Chapter 5 quantifies the additional advantage of replication compared to migration alone. However, for sequential files, migration has the same performance as replication. Whether the additional copies should be invalidated or updated when the file is updated depends on the installation and the rules for placing users at nodes. The algorithms with the best performance use the average reference rate, the number of consecutive opens in update mode, and the time since the node started using the file as the decision variables. The last section of the chapter shows that these algorithms have a good performance with little room for improvement.

As discussed previously, our approach is in many aspect similar to the one Porcar used. However, his work lacks an analysis of the stochastic properties. We therefore disagree on the appropriate model for how files are referenced. Naturally, the algorithms we investigate are different. A broader range of algorithms are included in our analysis, in particular algorithms that adapt themselves to the observed reference patterns. Porcar's analysis is mostly restricted to demand type algorithms, where the file is moved or replicated each time a different location starts using them. The use of the look-ahead algorithms as an analysis tool is also different from Porcar's approach.

Chapter 2
Exploratory Analysis of File Use

Summary

This chapter is an analysis of the file reference patterns to user files at three commercial installations. The results are later used to construct file placement algorithms for user files in a distributed environment. The focus is on those aspects of the reference patterns that are most important for file placement. Based on our problem definition, only files referenced by more than one user during their lifetime (a shared file) present a placement problem. Even though the users may not reference the file concurrently, some of the references from different users may have to be done over the network.

Although only few user files are shared, they represent a disproportional larger fraction of the total I/O traffic to disk files (up to 47% of all bytes transferred). How they are managed will therefore affect the system performance.

The shared files do not have a uniform behavior. They vary in size, number of references per file, number of users accessing them, and the time between references. No single set of fixed parameters common for all files and users can therefore adequately describe how shared files are referenced. The next chapter will therefore investigate appropriate stochastic models for reference patterns to user files.

2.1. Introduction

This chapter is an exploratory analysis of file use at three large commercial installations. The analysis is the basis for an investigation of different file placement / migration schemes for a distributed file system. It therefore concentrates on those aspects of file use that are important for determining where a file should be placed. Some of these are file size, file lifetime, number of users accessing a file, number and mode of references to a file, fraction of a file accessed per open, the time between opens to a file, and the order users will access a file. The results from this analysis are later used to construct file placement / migration schemes that improve system performance.

The exploratory analysis emphasizes shared files, since these are the only ones that potentially can present a placement problem. A file is shared if at least two different users have referenced it sometime during the trace. If these users are located at different processors, some of the file references will result in network traffic. The file will then be remote to at least one of the users. The reference must then either be done over the network or the file can be moved or copied to the processor accessing it. Regardless, the result will be network traffic that may reduce the performance of the system. Files accessed by only one user do not represent a placement problem. They are best placed at the processor where the user is located, and they will therefore not result in any network traffic.

The first part of the chapter describes the traces and the installations they were obtained from. These installations were traced over a period ranging from 7 to 30 days. The next section provides a general overview of the file system activity. Even though shared files represent only a fraction of all files, they represent a substantial part of all the I/O traffic to disk files (up to 47%). The last part is a detailed analysis of shared files performed across file organization, file size, and sharing behavior.

The reference pattern to shared files is best characterized by the lack of uniformity among the various measures; there are large variations in the number of references per file, number of users accessing them, the time between opens, and the file size. The typical user will open a particular shared file less than 10 times during the trace period. It is therefore unlikely that parameters can be estimated accurately before a placement decision is made. For two of the traces there is a tendency for larger files (> 1M bytes) to be opened more frequently than small files. This is similar to what other researchers have observed [Smith81b][Floyd86]. However for the last installation the tendency is the opposite; small files (< 200K bytes) are referenced more frequently than larger ones. There are also indications that the order users will access a file is not random. Instead, the references are clustered with a longer interval between clusters from different users.

In the text we have used the terms "reference", "access", and "open" interchangeably. Generally, a reference refers to all I/O activity bracketed by the open and the close call.

2.2. Description of the Traces

The installations being studied are Hughes Aircraft Company, Stanford Linear Accelerator Center, and Amdahl Corporation. The traces were gathered in 1977 and 1978. The three installations represent different computing environments. Even though the technology has changed since the traces were gathered, we think they are representative of file reference patterns to user files at large installations. The transaction volume and the speed of the computers have increased, but this should not have significantly affected the reference pattern to the individual user files. Unfortunately, changes in the workloads are not reflected in the traces. All the traced installations use either IBM or IBM compatible equipment, and the results of the analysis may therefore not be valid for file systems with a different organization.

Commercial IBM environments were chosen as the background for the analysis, since user files are shared more frequently in these installations. The alternative was a UNIX trace from an academic installation, but these installations typically have few shared user files [Thomps87]. One study of the UNIX system at University of Rochester found only 2062 changes in locality (different user referencing a file) among all user files referenced during a 7 day interval [Floyd86]. At such an installation, file placement is not a problem, since so few files are shared.

2.2.1. The Installations

The three installations represent different computing environments, which is reflected in different use of the file system. The next three paragraphs contain a detailed description of the hardware and the software used at the various installations when the traces were obtained. For brevity IBM acronyms are used without further explanation; the particular instances of hardware and software that are used are of little interest for the remaining analysis.

The Stanford Linear Accelerator (SLAC) is representative for a scientific workload. At the time of the trace, the installation had two IBM 370/168 (3 and 5M bytes) and an IBM 360/91 (2M bytes) processor. The processors were loosely linked (by ASP version 3.1)[Visin80], and they shared access to almost all disks. One of the 370 machines was used as a front end. It handled spooling, time-sharing, and a large fraction of the batch workload. The two other processors were dedicated to batch jobs. The two 370s ran SVS (single virtual system), while the 360 ran MVT. For job entry and editing SLAC's own system, WYLBUR [Fajman73], was used. Only the front-end was traced, but, since

WYLBUR ran as a never-ending job, none of the interactive sessions were captured by trace program. The installation was traced from 1/28/78 to 2/10/78. During the trace period 490 users submitted 25039 batch jobs which referenced 1950 tape files, 6865 disk files, and 142576 temporary files. The number of temporary files was high since all spool files (print files) were classified as temporary files.

The trace from Hughes Aircraft (Hughes) is from the corporate computer center. The workload includes time-sharing, large scale business processing, and scientific processing. The installation had two processors: an IBM 370/165 (3M bytes) and an Amdahl 470/V6 (6M bytes). Both ran the operating system MVT, and they were loosely linked by ASP. The IBM processor was dedicated to database processing (IMS), while the Amdahl machine was used for time sharing and batch processing. The Amdahl machine was traced for 190 hours from 9-19-77 beginning at 00 am. Both batch and interactive jobs are included in the traces. During the trace period 1063 users submitted 16815 jobs which referenced 4837 tape files, 27465 disk files, and 133797 temporary files. The number of temporary files was high since all spool files were classified as such.

The trace from Amdahl Corporation (Amdahl) is from the corporate computer center. The installation had two Amdahl 470s running MVS. Both processors shared access to the same disks. The one processor that was traced processed both time-sharing (TSO) and batch jobs. However, all spooling was handled by the other processor. This does not affect the usefulness of the traces, since spool files do not represent a placement problem. The processor was traced for 833 hours (35 days) starting the 2-27-78. During the trace period 611 users submitted 47316 jobs which referenced 5919 tape files, 36409 disk files, and 902 temporary files.

2.2.2. Method

IBM's System Management Facility (SMF) [IBM73][IBM78] was used to obtain the traces. SMF collects data on system performance and the resources used by different jobs. It is primarily an accounting tool and it is not suited either for our analysis or the ensuing trace driven simulation. The SMF data was therefore reorganized and condensed into a reduced trace. Porcar's thesis [Porcar82] contains a description of the procedure for creating the reduced trace. Briefly summarized, the reduced trace ordered all events chronologically and assigned unique identifiers for all jobs, users and files. This simplified the ensuing analysis and made it simpler to keep track of files through name changes. To maintain consistency, dummy opens and closes were added for files that were opened outside the trace period. Any files left opened at the end of the trace period were considered to be closed at the end of the trace period. However, files that were opened before the trace started and remained opened during the whole trace were not included.

2.2.3. Description of File Organization in IBM Environment

To clarify the analysis we included a brief overview of the file organization used by the operating systems we traced (the OS/VS2 family). The files are organized hierarchically, logically in records, fields, and bits, and physically in disk cylinders, tracks, and blocks. Each file is composed of one or more records which are stored in disk or tape blocks. The smallest logical entity retrieved from a file is a record, which consists of one or more fields, each containing one or more bits. One field is often designated the key field which uniquely identifies the record to the users. The smallest physical entity retrieved from a file is a disk block. In the OS/VS2 environment storage space for a file is allocated in integer number of disk tracks.

Four different file organizations were used, sequential organization, index sequential organization, direct organization, and partitioned data sets. In addition, a fifth organization *concatenated data sets* was used to combine several sequential or partitioned files into one logical file.

A *sequential* file is as the name implies, a collection of ordered records that can only be accessed sequentially. Individual parts of the file cannot be updated. The only choices are to either overwrite the file or append new records at the end of the file.

An *index sequential* file is organized for fast retrieval/updating of a particular record. The records are ordered with ascending/descending key fields. For each file there is an index of the key field of the last record on each track. It is not necessary to sequentially search a file to find a particular record; the index gives the address of the track where the search must start. The individual tracks of an index sequential file can be updated without overwriting the remaining part of the file. Each disk track contains free space so records can be inserted, and overflow pointers can be used to insert new tracks. To maintain good performance, a periodic reorganization of the file is necessary.

In a *direct* file each record can be addressed directly without any search for the correct record, since there is a one to one mapping between the key field and the offset from the start of the file to the record. Direct files can be updated in place.

A *partitioned* file consists of several members and a directory. Each member is a regular sequential file, and the directory contains the member's name and its offset from the start of the file. The partitioned file is a space saving way of organizing small sequential files. In the OS/VS2 environment the minimum space allocated to a file is a disk track. By collecting the small sequential files into a partitioned data set, the disk space can be utilized more efficiently compared to allocating one track for each small file. Each member of a partitioned file can be updated in place by overwriting as long as the member's size is not increased. If the size increases, the old version must be erased and the new version appended to the end of the set. Occasionally, the partitioned file must be reorganized to reclaim the space from members that have been erased.

A *concatenated* data set is used to process two or more sequential or partitioned files as one file. The files themselves are not changed, but to the program using them they appear as one file (the physical files are concatenated). Concatenated data sets can only be read. The individual files in a concatenated data set can be used separately by other programs, so the concatenation is only temporary (for the duration of an open). SMF uses the name of the first file as the name of the concatenated data set.

A file's organization will to some extent determine the access method that can be used. The records in a sequential file can only be accessed sequentially. However, all the other file types can either be referenced according to their organizational structure, or they can be treated as sequential files.

2.2.4. Content of the Traces

The traces capture all interaction between the users and the file system at a coarse level. The events traced were: creating/scratching files, opening/closing files, and beginning/ending of job-steps. The traces did not capture any of the individual I/O operations, read, update, and seek, that were bracketed by the open and close call. In addition, the traces also contained the name of the file, when it was created, and where it was stored. For the opening/closing of files the following information was recorded: 1) the event time, 2) the file ID, 3) the ID of the user referencing the file, 4) the organization of the file, 5) the mode of the open (read or update), 6) the file size when the file was closed, 7) number of I/O operations, 8) and the block size. Each job submitted to the system

consisted of one or more separate job-steps. Among the information recorded for job-steps were event time, job ID, and cpu consumption.

The original SMF traces contained no information that determined when a file was opened. Instead, the open time was constructed. In the system traced a file was either opened in a non-exclusive read mode or in an exclusive access update mode. The event time of an open to a file was bounded by several events: 1) the start of a job or a job-step, 2) a file could not be opened if it was already opened by another user with exclusive access, 3) a file could not be opened in exclusive access if another user had opened but not closed the file, 4) files must be opened and closed within the same job-step, and 5) each open to a file was identified by a "dd name", and a job-step can only have one file opened using a particular "dd name".

The constructed open time is set not to violate these bounds. It is set to the earliest possible time, but not earlier than : 1) the end of the previous job-step or the start of the job, 2) the previous close associated with a particular "dd name" by the same job, 3) the close time of any open with exclusive access to the file, or 5) if the open is in exclusive mode, the previous close time.

In some cases the constructed open time will overestimate the actual open time of a file, in particular for editing sessions (TSO sessions). Each TSO session appears as one job-step, which starts when the user logs on. In many cases the logon time will be the only available bound for the epoch when a file was opened.

A file can be opened in four different modes: read mode, non-sequential update, append of a sequential file, and overwrite of a sequential file. A sequential file is the only organization where it is not possible to update a record in place. Instead the whole file will have to be overwritten. Although files with different organizations can be updated in place, they can also be treated as sequential files. They may therefore be overwritten and appended as sequential files.

Unfortunately, SMF records only the mode of the open, not whether the file actually was updated or not. The number of bytes transferred per open is the sum of the number of bytes read and written, and it is not possible to identify the bytes that were read or written. To be conservative, the number of bytes written is set equal to the total number of bytes transferred for opens in update mode, which implies the update traffic is over-estimated. This will only affect file placement algorithms that replicate files, since these algorithms must update all file copies whenever the file is updated. Over-estimating the update traffic implies under-estimating the advantage of these schemes, since the cost of updating the copies is set higher than the actual cost. However, even with this conservative assumption, replication is advantageous. This is documented in chapter 5.

The traces lack granularity for partitioned data sets. Each partitioned file is treated as one unit and references to the individual members cannot be identified. Instead, the opens are assigned to the data set as a whole. Similarly, only the size of the whole partitioned data set is recorded. The traces have the same lack of granularity for a concatenated data set, which consists of several files viewed as one logical file. The name of the data set is the name of the first file and all references are attributed to this file. However, the reported size is the size of the whole data set.

2.2.5. Validity of the Traces

The age of the traces is a disadvantage, but they are still suitable for analyzing file placement/migration algorithms for user files. We are mainly concerned with high level file reference patterns which change slowly, at least in the IBM OS environment. Changes in the hardware capabilities have little direct effect on the reference patterns to user files

(at the open/close level). These are mostly determined by user think time, file content, user workload and work habits, and less by the speed of the underlying hardware. With increasing hardware capabilities more users and tasks can be supported by the system. The growing system workload can increase both the number of references to files and the number of files referenced. However, we are only analyzing user files, and most of these files are only referenced by a few users. Any system size growth will most likely increase the number of files in the system, and not the number of users sharing a file or the number of references per file. We believe the traces are representative of current reference patterns to user files in large commercial installations.

The reference patterns to system files are more closely related to the underlying software, and the traces are not representative of reference patterns to system files in current MVS installations. Even if they were, they would still not be usable. We are using traces from a centralized system to project the behavior in a distributed system. Since the actual storage structure is transparent to the users, they will have the same reference patterns to user files both in a centralized and in a distributed system. This is not true for system files which are implemented differently in a distributed environment.

The evolution of hardware also affects the use of temporary files, which are created when a job-step starts executing and are erased when the job-step finishes. Typical examples are temporary files used by compilers. The reference patterns to these files may be affected by new compilers and a larger main memory. For our purpose, temporary files are of little importance, since they exist only for a brief interval and they are not likely to be shared. They can be placed either where they are created or where they are sent (for print files). Regardless, they will never be relocated, and they can safely be excluded from the analysis.

The traces cannot reflect any changes in reference patterns due to new tasks in a user's workload. Tasks like text processing, CAD/CAM, and electronic mail may represent a larger part of the current workload than what they did when the traces were recorded. In addition, some of the tasks in the old workload have now been shifted to mini and micro machines. In spite of this, we feel the traces are representative of user reference patterns found in large commercial installations.

2.2.6. Modification of the Traces

We found it necessary to modify two aspects of the traces. Some of the opens did not transfer any bytes, and in the operating systems traced, such opens preceded the actual opening calls to fetch the properties of the file. In current systems such preceding calls are unnecessary. In addition, they are irrelevant for the file placement, since they do not transfer any data. The opens that transferred no data were therefore removed from the traces.

In order to use the number of bytes transferred per open from the traces it is necessary to assume that no disk blocks are repeatedly referenced inside the same open call. However, this is a reasonable assumption. Ousterhout et al. [Ouster85] traced three academic UNIX† systems. They found that more than 90% of the files were accessed sequentially. This justifies the assumption that no block is repeatedly referenced within the same open.

According to the traces, some opens transfer more bytes than the file length. This is either due to 1) several passes through a file, 2) check pointing, or 3) error in SMF.

† UNIX is a trademark of Bell Laboratories.

We are mainly concerned with file placement done at the open / close level. With a large buffer size and local disks, it is sufficient to read a file only once from the permanent storage location. Additional passes through a file will most likely find the file blocks in the buffer cache. No additional transfers will then be necessary. For file placement, the interesting aspect of an open is the number of bytes that potentially will have to be transferred over the network. The number of bytes read or written to the buffer is of less interest. We therefore limited the number of bytes transferred per open to the maximum number that potentially would be transferred over the network. The following limits were used:

read mode	file size, since with effective buffer management a file needs only to be transferred once
update non-sequential or overwrite sequential	2 * file's size, since in the worst case the file may be read in its entire length and then overwritten
append sequential	file size when the file is closed, since the file can be read in its entire length before the difference between the previous and the new size is added.

For files used for checkpointing, these limits may underestimate the number of bytes transferred per open, but the effect is assumed negligible. Check pointing is used to periodically dump a program's data structure to a file. This dump serves as a starting point if the processor fails before the program has finished. Each time the program is checkpointed, the whole file is overwritten. Checkpointing need not be done in the same way in a distributed system. There are two choices, either using a temporary local file or forcing the dump to a file at another location. The latter choice increases the reliability, since the program can be restarted immediately at another location. The disadvantage is the increased network traffic. In addition, all files used by a job must be replicated to the backup location. Files used for checkpointing must then be handled outside the file placement scheme. If a temporary local file is used for checkpointing, the maximum number of bytes transferred over the network will either be less or equal to the given limits. It will be less only in the case when a program starts overwriting the file without reading the content first. However, we believe files used for checkpointing are seldom shared. They contain data that is valuable only to the user running the job which is checkpointing to the file.

With the modifications explained in the table above, the traces are valid representations of the maximum number of bytes that potentially can be transferred over the network by an open. The only potential source for error is the few files used for checkpointing. For these files we may underestimate the actual number of bytes transferred.

2.3. Results

This section contains the results of the exploratory analysis. The first part is a general overview of the traces, where the importance of file placement / migration for system performance is shown. The latter part of the section contains a more detailed analysis of shared files. These are the files that potentially can cause a performance degradation since some of the references to them will result in network traffic. The main characteristic of these files is their lack of uniformity among the various measures. Before presenting the results of the analysis, we describe how files are classified.

2.3.1. Classification of Files

The permanent disk files were classified into three groups, system files, user files, and shared user files. System files are meant to encompass all files used by the operating system. As we mentioned, our study is primarily for placement of user files. Further, the traces themselves are not a valid representation of the reference pattern to system files in a distributed system. System files were identified based on their name and volume identifiers. In addition, database files [IMS files] were classified as system files, since placement of data bases is a separate research problem.

The remaining files are either user files or shared files. Shared files are user files that are referenced by more than one user during the trace period. Only a few of these will be referenced concurrently by more than one user. In the traces we excluded any open calls that did not transfer any data. However, these open calls were included when the status of the file was determined. A file can therefore be classified as a shared file even if there is only one user who transfers any data to and from the file. The I/O traffic to these files represents a minute part ($<1\%$) of the I/O traffic to shared files, and they do not affect the results of the analysis.

The remaining files are designated user files. There may be some system files among these files that were used by only one user. We believe that any such misclassification will not affect the results, since the analysis is mainly concerned with shared user files.

2.3.2. Overview of the Traces

This section contains an overview of the traces. It shows why shared files must be analyzed in more detail. As mentioned earlier, these are the only files that can potentially represent a placement problem, since some of the I/O traffic may have to be done over the network. Even though shared files represent only a small fraction of all files, they represent a substantial part of the I/O traffic to permanent disk files (up to 47% of all bytes transferred). They are larger than ordinary user files and they are referenced more often. Using them is also costlier from a system perspective since both the reference and the file administration itself must be done remotely. For each remote open, a communication path must be set up and maintained between the user and where the file is stored. Doing this plus transferring the open over the path cost substantially more than a local open. How shared files are managed will therefore have an impact on file system performance.

The number of bytes transferred to and from files varies over time but with a periodic pattern. Figure 2.1 displays the average number of bytes transferred per hour from both disk files and tape files (all) as a function of time. The solid line displays the average for shared files alone. In spite of the logarithmic scale, it is possible to distinguish spikes due to increased activity during weekdays. The difference between the highest and lowest average bytes transferred per hour is about an order of magnitude.

Shared files represent a substantial part of all the bytes transferred to permanent disk files (47%, 23% and 39%, see Table 2.1). They display the properties one would expect from shared files: they are referenced more often than ordinary files, have a larger average size, and are more stable (compared to ordinary user files a lower percentage of them are created or erased during the trace). At Hughes, many of the shared files are erased and created during the trace, since they are used as one time scratch pads for communication between two users. These files are opened by one user in read mode and by the other user solely in update mode. The detailed behavior of shared files is analyzed further in the next section.

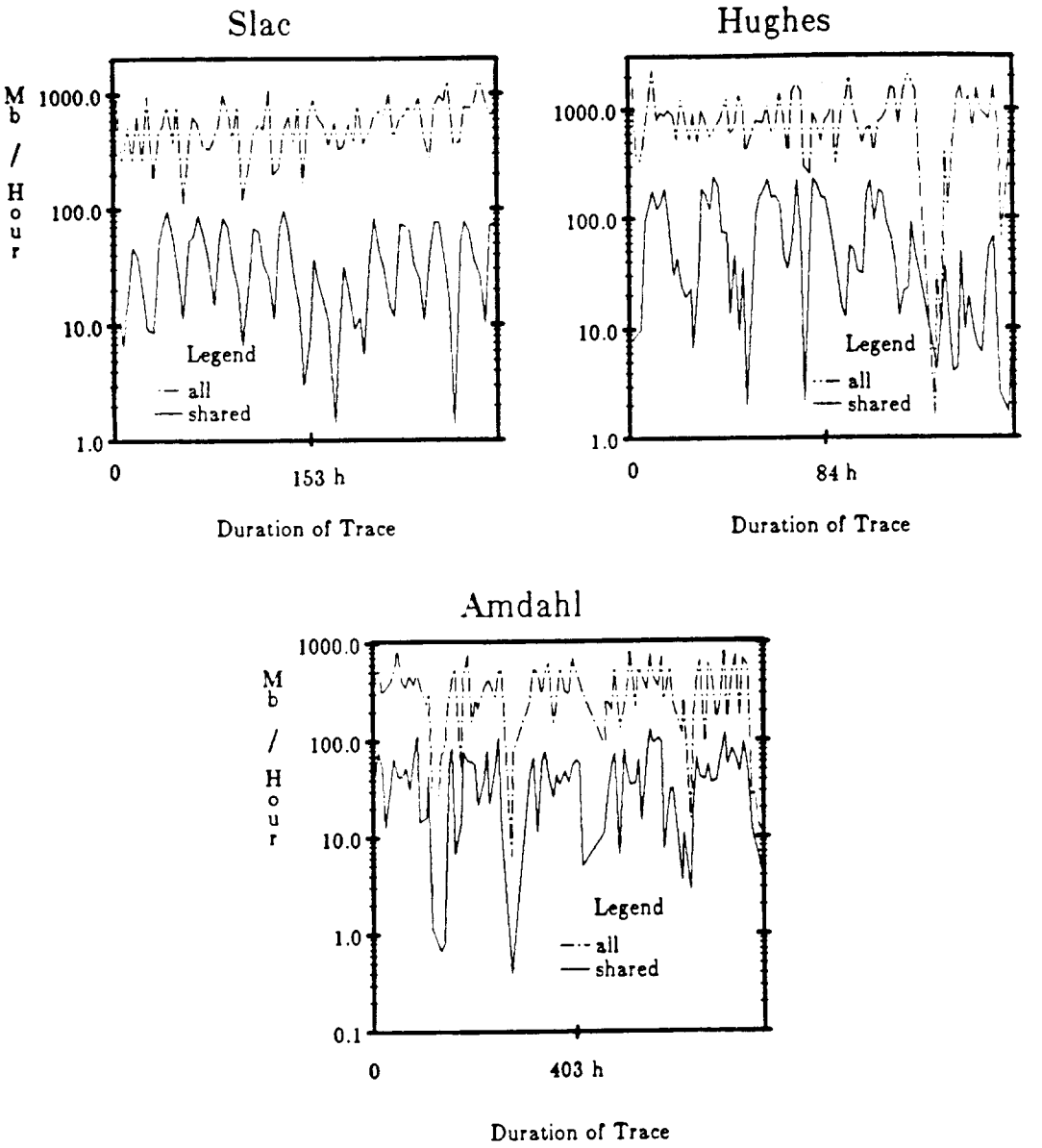


Figure 2.1: The average number of bytes transferred to and from permanent files per hour. The curve labeled "all" includes all permanent disk and tape files, while the curve labeled shared includes only shared user files stored on disk.

Ordinary user files are on the average referenced only a few times. Most of them are created and erased during the trace period and their average file size is small. This corresponds well to the results from other studies. Both Smith [Smith81b] and Floyd [Floyd86] reported that most files are only opened a few times. Floyd traced a UNIX machine at University of Rochester for 7 days. The average number of opens per user file

was 7.5 with 3 opens as the median. Smith used traces of user files at SLAC for 384 days. The granularity of his trace was at a day level, i.e., the trace only recorded whether a file had been referenced in a particular day. The actual number of references per day was unknown. The average number of days a file was referenced was 10.6 and the median was 2.0. In spite of the large difference in the length of the trace period, the numbers are comparable. As Ousterhout et al. reported, most files have a very short lifetime, in the order of minutes [Ouster85]. The number of opens to these files will be nearly independent of the trace period, and only for files with a long lifetime will the trace period influence the number of opens per file. The average number of opens per file will therefore only slowly increase with the length of the trace period.

To summarize, shared files are typically referenced more often than ordinary user files. They are more stable, and they have a large average size. Even though only a fraction of the files are shared (at most 20%), their management will affect system performance. Shared files represent a large fraction of all bytes transferred to disk (from 23% to 47% of all bytes transferred to permanent disk). In addition, referencing shared files is costly, since some of the references may have to done remotely, which incur a performance penalty.

Table 2.1: The table displays an overview of the properties of system files, shared user files, and ordinary user files. The various measures are accumulated over the trace periods, which were 14 days for SLAC, 190 hours for Hughes, and 35 days for Amdahl. Only results for disk files are included in the table.

	Slac			Hughes			Amdahl		
	system files	user files	shared files	system files	user files	shared files	system files	user files	shared files
Bytes Transferred 10^9 bytes	8.68	7.82	11.54	16.44	27.10	13.56	26.09	26.69	34.00
Mean no of accesses per file	155.5	6.7	92.0	95.3	4.1	9.0	69.6	4.9	95.8
Mean file size (Kbytes)	1950	488	1043	6276	584	292	1889	416	1905
# Files	288	6150	427	339	21460	5666	1637	32287	2485
# Created during trace	84	4063	92	13	14308	4561	689	28474	743
# Erased during trace	55	4370	83	15	13432	3799	697	28898	679
# sequential files	147	4527	163	135	19813	5066	604	28878	885
# partitioned files	126	1528	249	109	1207	433	1018	2583	1463
# direct files	13	93	13	37	277	153	12	742	89
# index seq. files	0	2	2	58	136	6	2	83	46
# others	2	0	0	0	27	8	1	1	2

2.3.3. Detailed Analysis of Shared Files

We first investigated the general properties of shared files, and how those properties vary for files with different organizations. The measurements for this analysis are contained in Table 2.2 and in Figure 2.2 to Figure 2.5. The remaining part of the chapter contains the analysis of the differences between files with different sizes, content, and sharing patterns.

Two of the traces, from SLAC and Hughes, were analyzed in Porcar's thesis [Porcar82]. He analyzed the file size, the number of opens per file, the fraction accessed and the inter-open time. His exploratory analysis was limited to tape files, disk files, user files, shared files as a group, and shared files with different sizes. The initial part of our exploratory analysis for SLAC and Hughes will duplicate some of his research. However, our analysis extends beyond the results found in Porcar's thesis. We analyze several additional aspects of the reference patterns to shared files, for example the number of opens per user and the updating patterns. In addition, the differences in reference patterns for files with different sharing patterns, file types, and contents are analyzed in more detail.

The reference pattern to shared files cannot be described by one set of fixed parameters common for all files. There are large variations among them in size, number of references per file, number of references per user/file pair, time between opens and the fraction accessed per open. This is true even for files with the same organization, size, or sharing pattern. As expected, partitioned files are referenced more often and by more users than sequential files. In addition, partitioned files are typically larger than sequential files. However, the distribution for the number of opens per user/file pair is similar for the different organizations.

2.3.3.1. Size

As already mentioned, shared files tend to be larger than ordinary user files, but there are large variations between their sizes. The distribution of the sizes of the shared files is displayed in the left part of Figure 2.2 and in Table 2.2. Sequential files tend to be smaller than the other files. For SLAC the mean size for sequential shared files was 838K bytes, while the average size was 1043K bytes. The same numbers for Hughes and Amdahl were 144K bytes versus 292K bytes and 827K bytes versus 1905K bytes. However, the sequential files are typically larger than those traced at academic installations [Ouster85][Floyd86][Pope84][Satyan81].

In a study conducted by Floyd [Floyd86] of a Unix system at the University of Rochester, the results indicated that the average user file was $\approx$ 8K bytes. This number corresponds well with those found by Ousterhout [Ouster85] in a trace of three Unix systems at University of California, Berkeley. Satyanarayanan reported in a study of a PDP-10 running TOPS-10 at CMU-CSD that 50% of the files were smaller than 5 blocks (one block is 128 36 bit words). The 95% quantile for the size was 100 blocks. Similarly, Pope [Pope84] reported that 76% of all files referenced were less than 4K bytes in a CMS system which he traced (presumely a CMS system at IBM Yorktown).

There are at least 4 reasons for these differences:

1) Shared files are typically larger than ordinary files. All the other measurements we referenced are averaged over all user files, and they will therefore be smaller than those reported in Table 2.2.

2) In the analyzed traces, small sequential files are usually organized into a partitioned data set to save space. Otherwise, each sequential file would be allocated at least one disk track. The fraction of small files is therefore much lower.

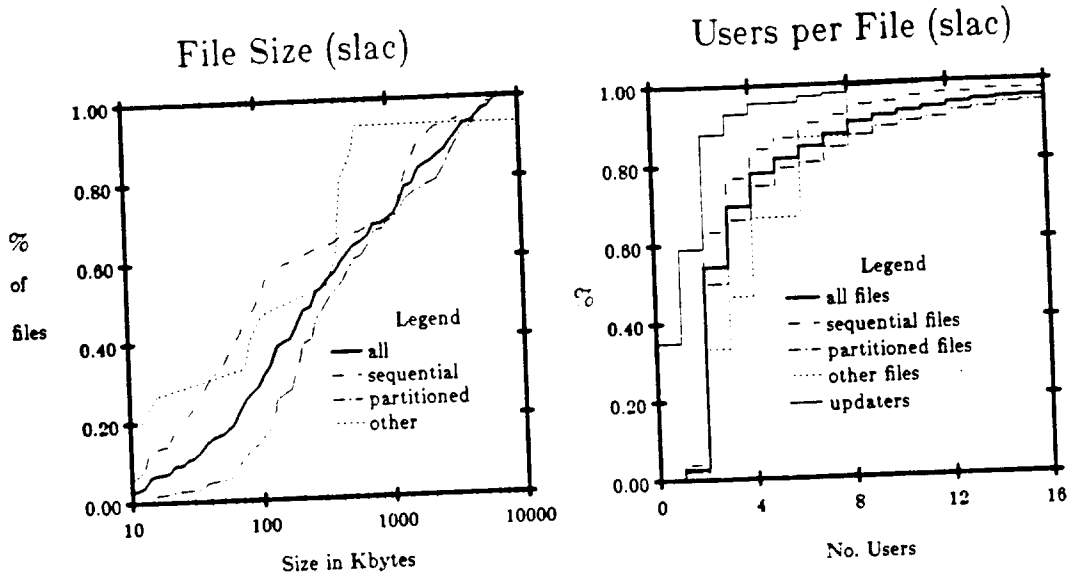


Figure 2.2a: The left figure displays the distribution of the file sizes found at SLAC, while the right figure is the corresponding distribution for the number of users referencing a file. The curves labeled "other" are for all other file organizations, mostly index sequential and direct files. The mean for each distribution is found in Table 2.2. The curve labeled "updaters" displays the cumulative distribution for the number of users updating a file at least once.

3) In an IBM system disk space is allocated only in whole tracks. The minimum file size is 7K bytes (on a IBM 2314 disk with 7K byte track and a total 29M bytes capacity). For a Unix system, space is allocated in increments of 1K bytes [McKusi83], while for the CMS system the increments are 4K bytes.

4) The various installations have different workloads. Two of the traces are from corporate computer centers, which have a different workload from academic installations.

In another study of SLAC, Smith [Smith81b] found that the average static file size for all user files accessed during 384 days was $\approx$ 50K bytes. Unfortunately, Smith's measurements cannot be directly compared with the numbers in Table 2.2, since our measures are for shared files, while Smith's numbers are for all user files. The average file size for user files at SLAC (in Table 2.1) is higher than those measured by Smith. There are two reasons for this difference: 1) Smith used traces with a much longer trace period. The average static file size will then be determined by the size of the many short lived files. These files are most likely shorter than the files that are long lived. The trace we used has a much shorter trace period. The average file size should then be less influenced by the length of the short lived files, since in a short trace there will be fewer of these files. 2) The average for user files at SLAC includes the user files written to scratch disks, while Smith excluded these files from his study [Smith81b]. The files on these scratch disks tend to be larger than the ordinary disk files [Smith81b]. The average size reported in Table

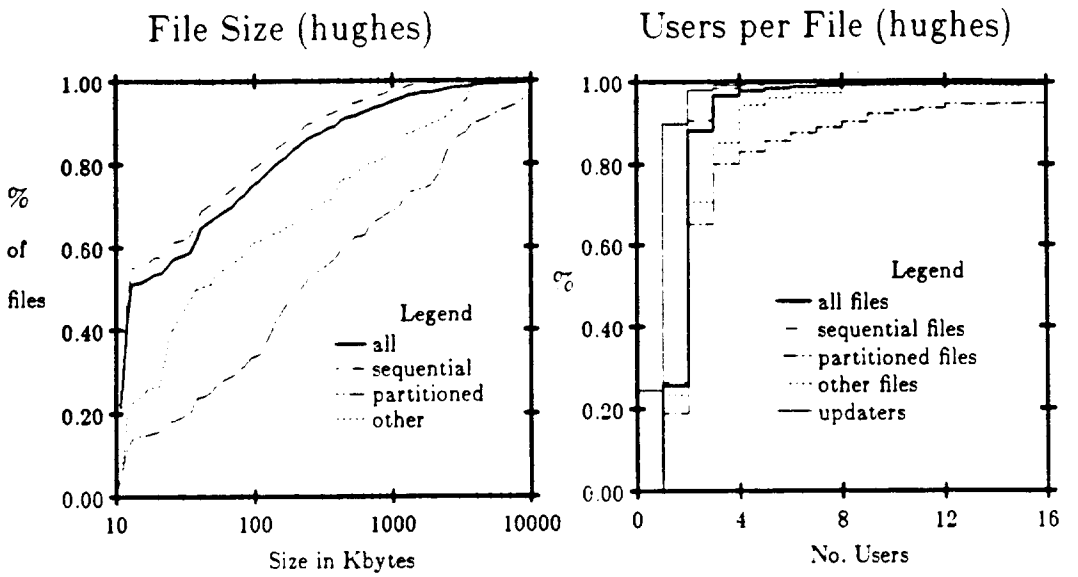


Figure 2.2b: The left figure displays the distribution of the file sizes found at Hughes, while the right figure is the corresponding distribution for the number of users referencing a file. The curves labeled "other" are for all other file organizations, mostly index sequential and direct files. The mean for each distribution is found in Table 2.2. The curve labeled "updaters" displays the cumulative distribution for the number of users updating a file at least once.

2.1 should therefore be larger than the static average reported by Smith. Our results correspond well to those reported by Porcar [Porcar82].

2.3.3.2. User Sharing of Files

Shared files are typically referenced by only a few users, most often only in read mode. Sequential files are typically shared by fewer users than the other file types. The right part of Figure 2.2 displays the distribution of the number of different users referencing a file during the trace period. The average number is less than 5 and the median is $\approx$ 2 users per file for all installations and file types. Only users transferring data to or from a file are counted in the measurements, and as previously mentioned, a few of the shared files are referenced by only one user that transfers any data to or from the file.

It is difficult to compare our numbers to those of other studies, since few have published data on the number of users sharing a file. However, the system Floyd traced for 7 days [Floyd86] displayed the same trend. Of the few user files that were shared, only 42% were referenced by more than 2 users.

Most users will open a file only in read mode (70%, 64% and 60%), and replicating files should be a reasonable strategy. However, the content of a file is likely to change. More than 65% of the files are updated during the trace period, but typically only one user will update the file; from 3% to 12% of the files are ever updated by 2 or more users.

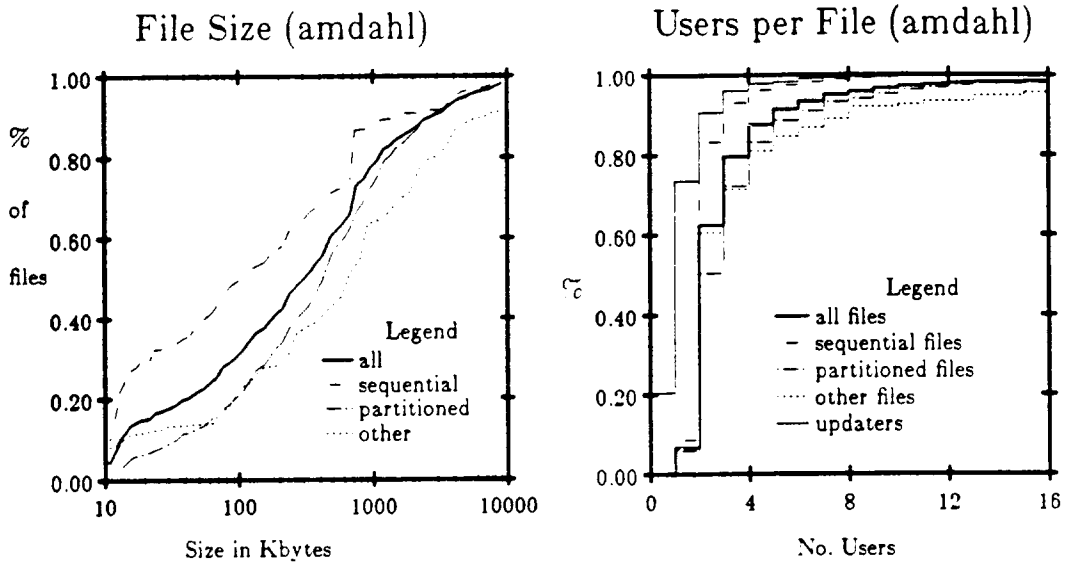


Figure 2.2c: The left figure displays the distribution of the file sizes found at Amdahl, while the right figure is the corresponding distribution for the number of users referencing a file. The curves labeled "other" are for all other file organizations, mostly index sequential and direct files. The mean for each distribution is found in Table 2.2. The curve labeled "updaters" displays the cumulative distribution for the number of users updating a file at least once.

There are differences between the various file types. Sequential files tend to be referenced by fewer users than any of the other file types; the average is lower, and the cumulative distribution is shifted to the left compared with the other types. Index sequential files and direct files are best suited for retrieval of random records, and they can easily be updated or expanded. Most users referencing them take advantage of these properties. More than 74% of the users referencing these files will do so at least once in update mode. As a result, from 87% to 96% of all shared index sequential and direct files are updated.

Table 2.2: The table displays some of the characteristics of shared files for the three traces. The first column is for all shared files, while the two next ones are for sequential and partitioned shared files. The last column contains the combined results for all remaining file types. A more detail picture of the different measures can be obtained from Figure 2.2-5. All measurements are accumulated over the trace period.

	Slac				Hughes				Amdahl			
	all shared files	sequential files	partitioned files	other	all shared files	sequential files	partitioned files	other	all shared files	sequential files	partitioned files	other
no of files	427	163	249	15	5666	5066	433	167	2485	885	1463	137
in % of shared files	100%	38.2%	58.3%	3.5%	100%	89.4%	7.6%	3.0%	100%	35.6%	58.9%	5.5%
% of all opens to shared files	100%	16.7%	81.3%	2.0%	100%	44.0%	41.1%	14.9%	100%	5.2%	88.4%	6.4%
average size (Kbytes)	1043	838	1182	949	292	149	1681	1012	1905	827	1902	8903
average no opens per file	92.03	40.26	128.32	52.07	9.04	4.45	48.61	45.55	95.85	14.03	143.92	110.98
average no opens in update mode per file	11.58	17.90	5.27	47.20	3.13	1.77	6.65	35.16	23.76	4.40	29.55	87.01
average no users accessing a file	4.78	3.45	5.69	4.13	2.16	1.91	4.90	2.68	3.29	2.29	3.81	4.16
average no users accessing a file in update mode	1.43	1.68	1.14	3.47	0.95	0.85	1.46	2.52	1.31	1.04	1.31	3.09
% of files not updated	34.9%	38.7%	33.7%	13.3%	24.6%	24.3%	35.8%	3.6%	20.4%	16.8%	24.3%	2.9%
average no opens per user/file pair	19.23	11.66	22.53	12.60	4.18	2.33	9.92	17.02	29.12	6.12	37.73	26.67
Fraction accessed read mode	0.323	0.684	0.280	0.555	0.359	0.662	0.159	0.078	0.084	0.438	0.061	0.334
update mode	0.593	0.570	0.726	0.444	0.380	0.505	0.146	0.304	0.148	0.357	0.127	0.160

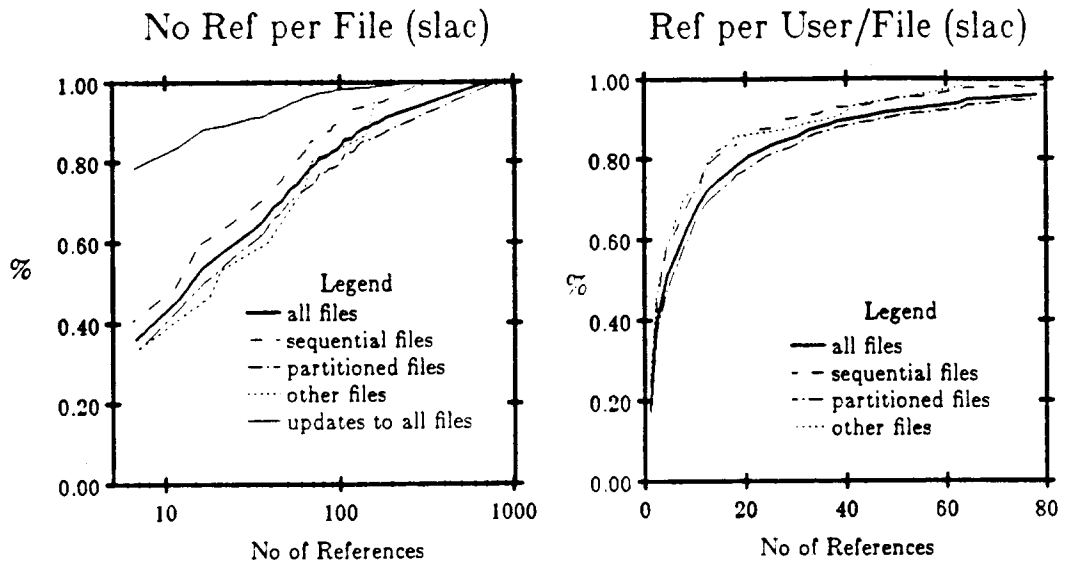


Figure 2.3a: The left figure contains the distribution of the number of opens per shared file for the trace from SLAC, while the right one contains the distribution of the number of opens per user file pair. As with the previous figures, the means of the various distributions are found in Table 2.2. The curve labeled "updates to all files" displays the cumulative distribution of number of opens in update mode.

2.3.3.3. Number of Opens per File

Shared user files are opened more often than ordinary user files. They tend to be more stable, and fewer of them are erased during the trace period.

Shared files are typically opened in read mode and there are large variations in the number of opens per file. Most users will, however, reference a file only a few times (the median is $\approx$ 4 opens per user/file pair). The distribution of number of opens per shared file and per user/file pair (only shared files) is displayed in figure 2.3.

The distribution for the number of opens per shared file is skewed towards few opens. For all the installations the median is much smaller than the mean; two of the traces have a median $\approx$ 12 while the mean is around 94 opens per file. Sequential files are opened less frequently than other file types (see Figure 2.3).

Most of the opens are in read mode (more than 65%), and replication may be an advantageous strategy. This is, however, not true for index sequential and direct files. Most of the opens to these files (more than 77%) are in update mode; the mean varies from 35 to 87 updates (see table 2.2), but the median is around 7 updates per file.

The average number of times a user will open a file varies with the file organization. However, most of the users referencing a file will do so only a few times. The distribution of the number of opens per user/file pair is skewed; the median for all traces is around 4 opens per user/file pair, while the mean is from 4 to 30 opens per user/file pair.

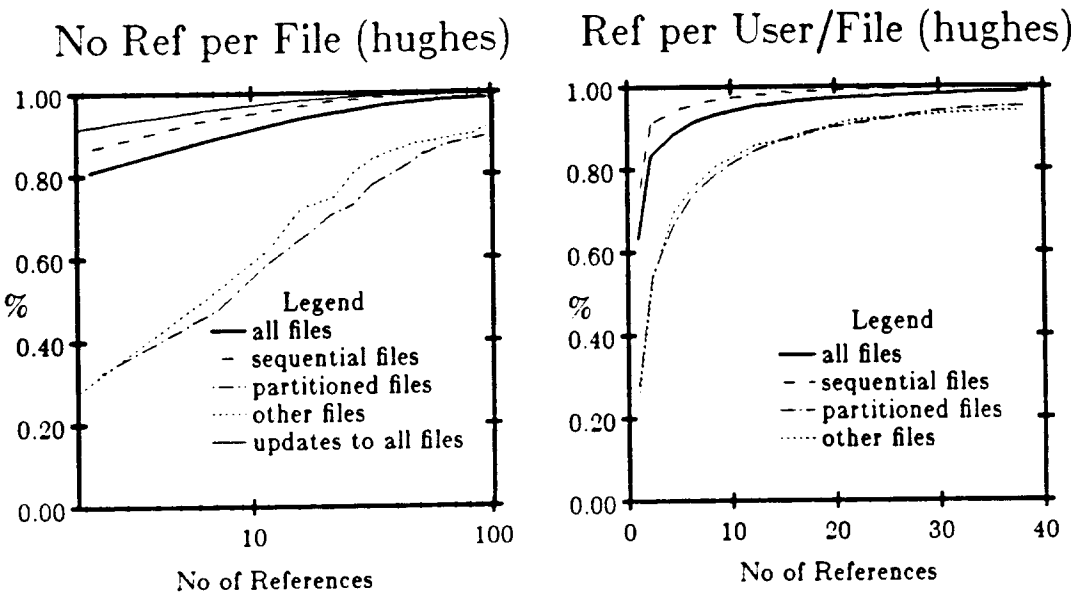


Figure 2.3b: The left figure contains the distribution of the number of opens per shared file for the trace from Hughes, while the right one contains the distribution of the number of opens per user file pair. As with the previous figures, the means of the various distributions are found in Table 2.2. The curve labeled "updates to all files" displays the cumulative distribution of number of opens in update mode.

There are some differences in the number of opens per user/file pair. A user will typically open a sequential file less often than the other types of files (see Figure 2.3). Based on the distributions found in Figure 2.3, there do not appear to be any major differences between the distribution of the size for partitioned, index sequential, and direct files. However, the distributions for the different file types do not have the same tail, since there are larger variations in the average number of opens per user (see Table 2.2).

The skewness of the distribution for the number of opens per user/file pair has some implication for the file placement algorithms. An implicit assumption of most work on file placement is that the reference pattern can be observed until the reference rates from the different users are accurately estimated. (for examples see the works surveyed in Dowdy's article [Dowdy82]). The typical user will open a file only a few times, usually during a short interval; parameter estimates are therefore likely to oscillate over time. Consequently, a file placement/ migration scheme should not be based on having accurate parameter estimates available for all users who will ever open the file. Instead, the placement/migration scheme should be able to use rough estimates or no estimates at all. They must also have the ability to reevaluate a placement decision if the parameter estimates should change. Dynamic placement algorithms should be well suited for this type of reference pattern.

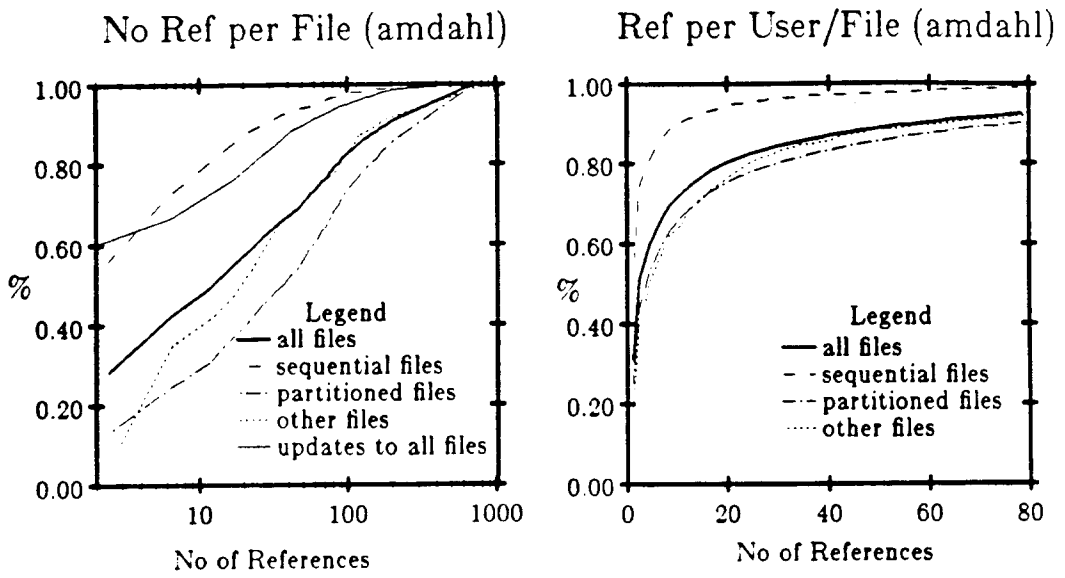


Figure 2.3c: The left figure contains the distribution of the number of opens per shared file for the trace from Amdahl, while the right one contains the distribution of the number of opens per user file pair. As with the previous figures, the means of the various distributions are found in Table 2.2. The curve labeled "updates to all files" displays the cumulative distribution of number of opens in update mode.

2.3.3.4. Fraction Accessed per Open

In a MVS environment, most opens will not access the whole file. This was expected for partitioned files, since each member can be referenced individually; in most cases, only a fraction of the partitioned file will be transferred at each open. The same behavior is expected for index sequential and direct files, since these file types are designed so it is easy to reference a small part of the file. Most likely, the opens to these files will take advantage of this property. However, even for sequential files, there are a substantial number of opens that do not access the whole file (see Figure 2.4); although the fraction accessed per open file is larger than for other types, the median is smaller than 80% of the file accessed per open, and the mean is smaller than 65% of the file accessed per open.

This is a difference from the UNIX environment, where more than 2/3 of the opens accessed the whole file [Ouster85]. A possible explanation for this difference is the large number of small files typically found in an academic UNIX installation. Many of the files will occupy a single disk block, so any access to the file will access the whole file (the whole disk block). In addition, the short files typically have contents that are read in their entirety, like C definition files, command files, and directories [Ouster85]. The number of opens that transfer the whole file will therefore be larger in traces from such installations. For the systems we analyze, it is no longer valid to assume that each open will access the whole file. An important implication is that demand copy or migration algorithms no longer will be the best strategy. If an open only will access a fraction of a file, several consecutive opens are needed to justify moving a file to a new location. If the

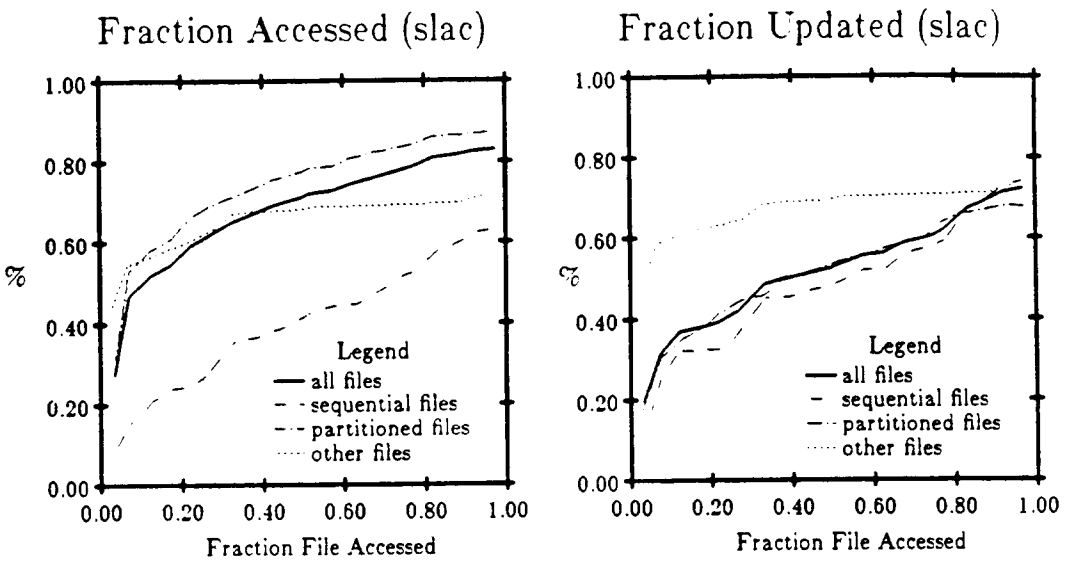


Figure 2.4a: The left figure displays the distribution of the fraction accessed per open for the trace from SLAC, while the right figure is the corresponding distribution of the fraction updated per open in update mode. The curves labeled "others" is for all other file organizations, mostly index sequential and direct files. The mean for each distribution is found in Table 2.2.

node opens the file only a few times, the file transfer traffic will be larger than the saved remote I/O traffic, and moving or copying the file each time a node starts referencing the file can be non-optimal.

Opens in update mode display the same overall trend. Most of them will update only a fraction of the file each time. This is true even for sequential files; the fraction updated per open is much larger than for any other file type. The median is around 70% of the file updated per open, while the mean is from 35% to 57%. This is mainly due to opens which append only a small amount of data to a sequential file. An additional source is opens that overwrite only a fraction of a track. In the file systems we traced, disk space for a file is allocated in integer number of tracks. An overwrite that does not fill up the space allocated to the file will to SMF appear as overwritting only a fraction of the file.

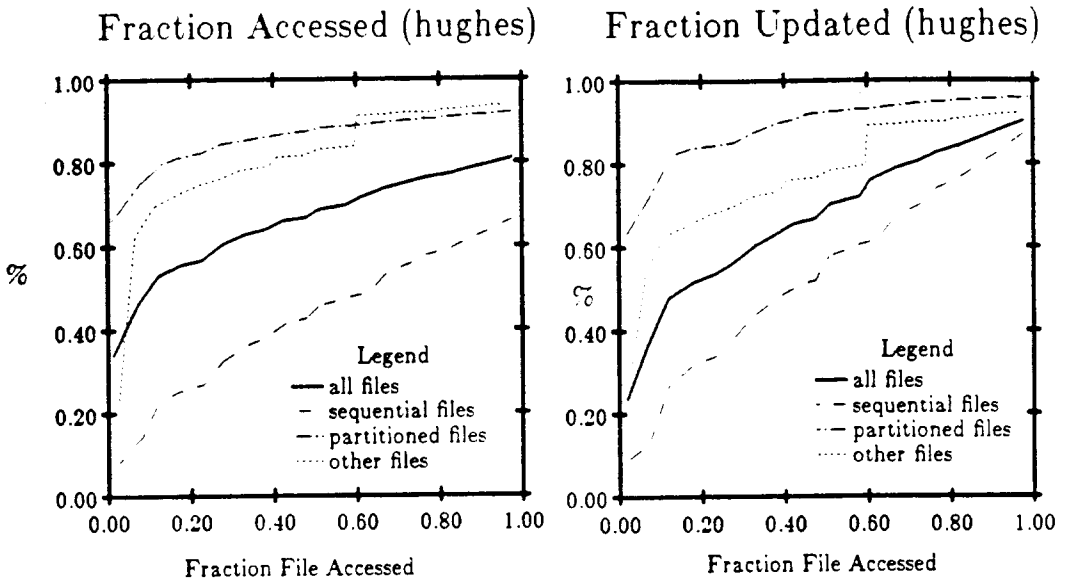


Figure 2.4b: The left figure displays the distribution of the fraction accessed per open for the trace from Hughes, while the right figure is the corresponding distribution of the fraction updated per open in update mode. The curves labeled "others" is for all other file organizations, mostly index sequential and direct files. The mean for each distribution is found in Table 2.2.

2.3.3.5. Time between Opens

There is a distinct pattern to the order users will reference a file. Figure 2.5 displays the distribution of the time between opens to the same file (labeled all). Few of the opens to a file follow each other instantaneously (are back to back)(see Figure 2.5). The mean times between opens are 107, 208, and 231 minutes for the three installations, while the medians are from 1 to 4 minutes. The fraction of opens that were back to back is higher for the trace from SLAC, but only batch jobs were traced at this installation. The second curve in Figure 2.5 displays the distribution of the time between opens for a subset of all opens. Only inter-open times between opens from different users are included in the subset. If the order users referenced files was random, the distribution for this subset should have the same distribution as the whole set. The subset clearly has a different distribution from the whole set; The means are 189, 402, and 869 minutes, while the medians are from 12 to 30 minutes.

There are two possible explanations for this shift. 1) The order users reference a file is not random. Instead, a user will typically reference a file several times within a short interval followed by a longer period before the next user starts using the file. Most of the longer inter-open times will then be between opens from different users. This would explain the shift in the distribution towards the longer inter-open times for the subset. With such a reference pattern, any model with random access pattern, for example a Poisson model, will not be a suitable one. This type of clustering reference pattern is

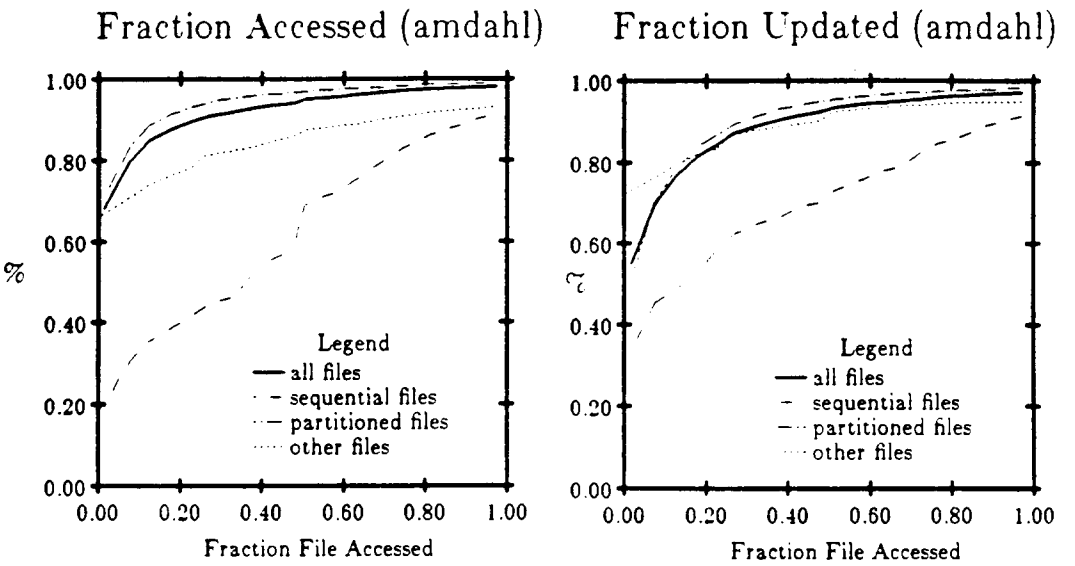


Figure 2.4c: The left figure displays the distribution of the fraction accessed per open for the trace from Amdahl, while the right figure is the corresponding distribution of the fraction updated per open in update mode. The curves labeled "others" is for all other file organizations, mostly index sequential and direct files. The mean for each distribution is found in Table 2.2.

investigated further in the next chapter. 2) Another possible explanation is random references per file but with a negative correlation between the time between opens and the number of users sharing a file. If this was true most of the longer inter-open time would come from files referenced by many users. Most of the longer inter-open times would be between opens from different users, and distribution for the subset would be shifted towards longer inter-open times. However, a negative correlation between number of users sharing a file and the time between opens is not a plausible model. It implies that files shared by many users would be referenced less often than files shared by only a few users.

2.3.3.6. Analysis by Size

To obtain a more detailed view of the reference patterns to shared files, we classified the files into three classes according to their size. The reference patterns to these classes were then compared. The shared files were classified as small ($\leq$ 200 K bytes), medium (between 200 K bytes and 1 M bytes) and large ($>$ 1 M bytes). The first four columns of Table 2.3 contain the results for the analysis across the different sizes.

We expected large files to be more actively referenced than the smaller sizes, since these files should be used relatively often to justify their existence on disk instead of on tape. For two of the traces, Amdahl and Hughes, this was true. The average number of opens, users sharing, and users updating a file were all increasing with the size. For the remaining trace from SLAC, the opposite trend was apparent; the average number of

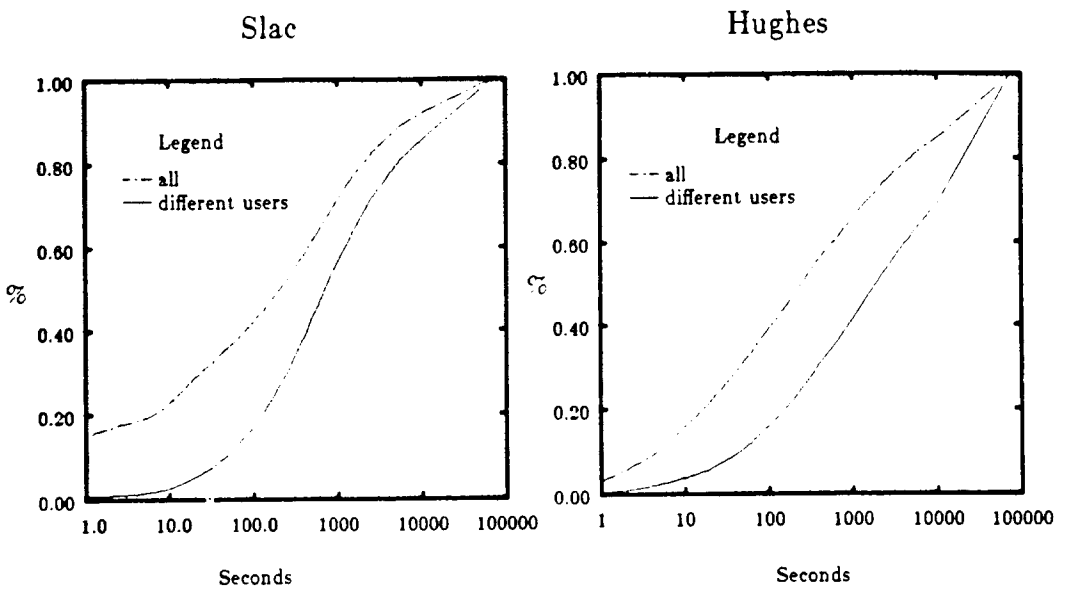


Figure 2.5a: The figure displays the distribution of the time between opens to the same file. The left figure is for the trace from SLAC, while the right is from the trace from Hughes. The curve labeled "all" is the distribution for the time between all opens to the same file, while the curve labeled "different users" encompass only the time between opens from different users to the same file.

opens and opens in update mode is decreasing with the file size. The numbers cannot be compared directly, since small and large files are composed of files with different types. Sequential files tend to be smaller than the other file types, and they are typically referenced less often. Due to this bias, the average number of opens for large files should be larger than the average for small files. However, this did not fully explain the difference between the average number of opens for each size group in Table 2.3. Let us assume that all files with a particular organization have the same distribution for the number of opens per file. The average number of opens for large and small files with the same organization should then be the same, and any difference is due to random events. Based on the numbers in Table 2.2, the expected number of opens per size group would be:

	Small	Medium	Large
SLAC	79.5	122.3	92.9
Hughes	7.4	12.3	28.7
Amdahl	78.5	102.7	116.4

For Amdahl and Hughes, the expected numbers of opens for the various sizes are smaller than the ones observed in the traces (see Table 2.3 b and c). The average number for large files at Hughes was 46.9 opens per file and 211 opens per file at Amdahl. Large files must typically be opened more often regardless of organization. The opposite is true for SLAC. The expected number of opens for large files is larger than the empirical average

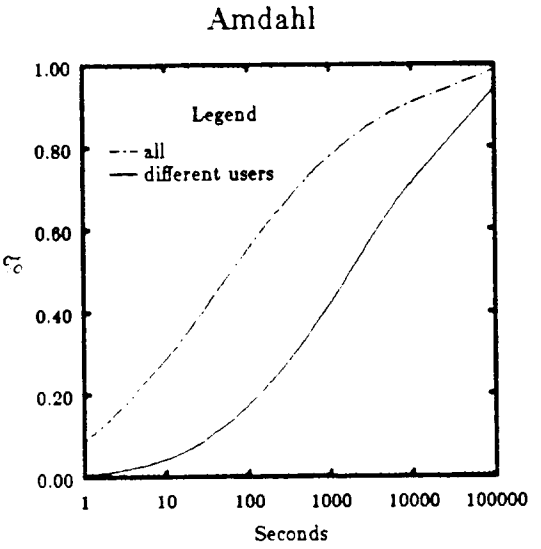


Figure 2.5b: The figure displays the distribution of the time between opens to the same file for the trace from Amdahl. The curve labeled "all" is the distribution for the time between all opens to the same file, while the curve labeled "different users" encompass only the time between opens from different users to the same file.

(83 opens per large file), and at SLAC, large files must be opened less often than small files regardless of organization.

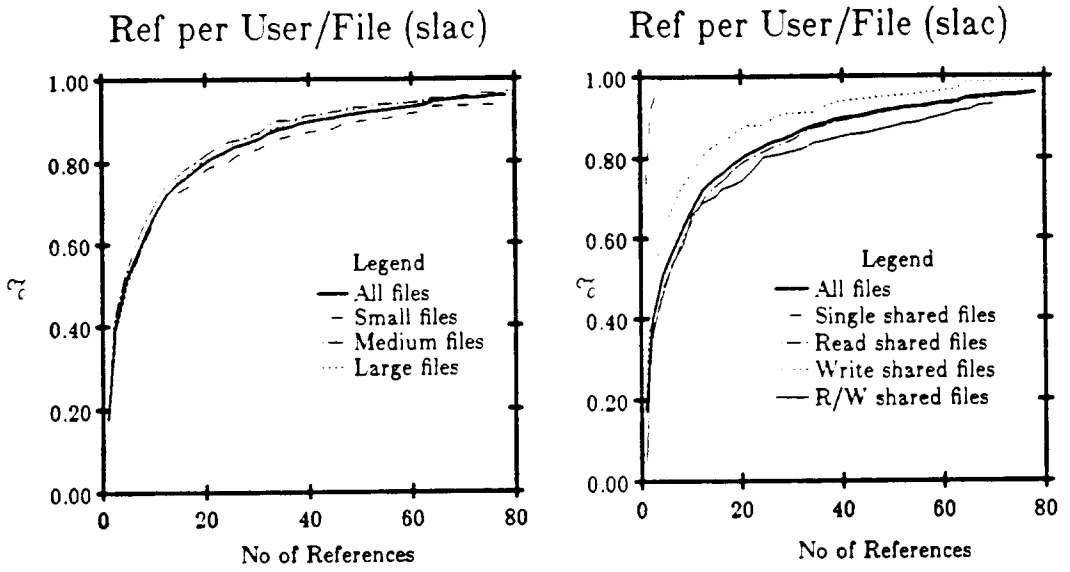
For all traces, the average number of users sharing a file and the number of users updating a file increase with the file size. Further, larger files are more likely to be updated during the trace than smaller files. The fraction of files that are not updated decreases with the file size. The probability that an open will update a file cannot be a constant, at least not for SLAC. Otherwise, the probability a file would be updated should decrease with the file size, since large files tend to be opened less frequently than small files at this installation. Any difference between large and small files cannot be explained by the composition of the groups, since both partitioned and sequential files have roughly the same probability of being updated.

The difference in update probability between small and large files is is important for replication schemes. Due to their size, large files should not be replicated "unnecessarily", and demand copying of large files is less attractive, since these files are more likely to be updated.

Even though there are large variations in the average number of opens per user/file pair for small and large files, the difference is mostly due to the tail of the distribution (see Figure 2.6). Further, for all groups and all traces 60% of the user/file pairs will issue less than 10 references to a file.

To summarize, classifying files into three groups according to size showed that there is a trend between size and number of references. For two of the traces, Hughes and Amdahl, large shared files are more likely to be opened than small shared files. This is

Figure 2.6a: The left figure displays the distribution of the number of opens per user/file pair with the files classified into three groups according to size. The right figure displays the same distribution but with the files classified into 4 groups according to how they are shared. All the distributions are for the trace from SLAC.



similar to what has been observed in other analysis [Smith81b][Ousterh85]. For the last trace, SLAC, the trend is the opposite; small files are typically opened more often than large ones. In addition, large files are more likely to be shared and to be updated.

2.3.3.7. Analysis by Sharing Patterns

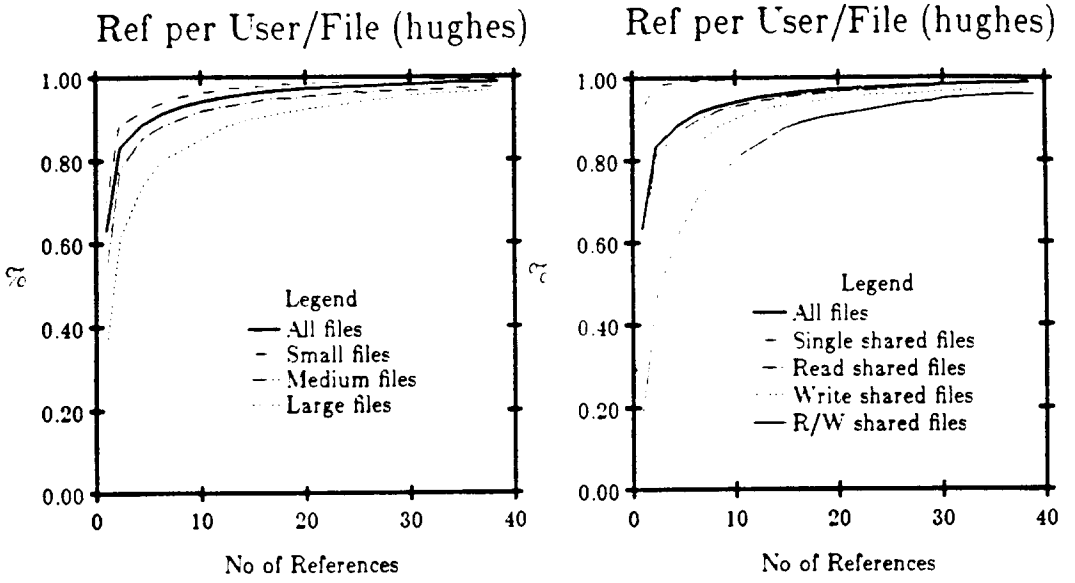
Four different patterns of sharing were identified and the traces were analyzed in terms of these groups. The definitions for the four classes are:

Single-shared files. These files are referenced in read mode by only one user and in update mode by a single different user. The files are used in a consumer/ producer relationship between these two users, where one user is feeding information to only one consumer. For lack of a better classification, we also included into this group those files shared by two users where the second user only opened the file without transferring any bytes to or from the file.

Read-shared files. These files are referenced in read mode by at least two users, but at most one user is updating the file. The read-shared classification is meant to encompass the library files, where a file is used by many users in read only mode. Since the content of a library file must be maintained by somebody, the classification allows updating limited to one user. If this was not the case, the class would contain only static library files that seldom changed.

Write-shared files. These files are referenced in update mode by at least two users, and in read mode by at most one user. The write-shared classification covers a many-

Figure 2.6b: The left figure displays the distribution of the number of opens per user/file pair with the files classified into three groups according to size. The right figure displays the same distribution but with the files classified into 4 groups according to how they are shared. All the distributions are for the trace from Hughes.



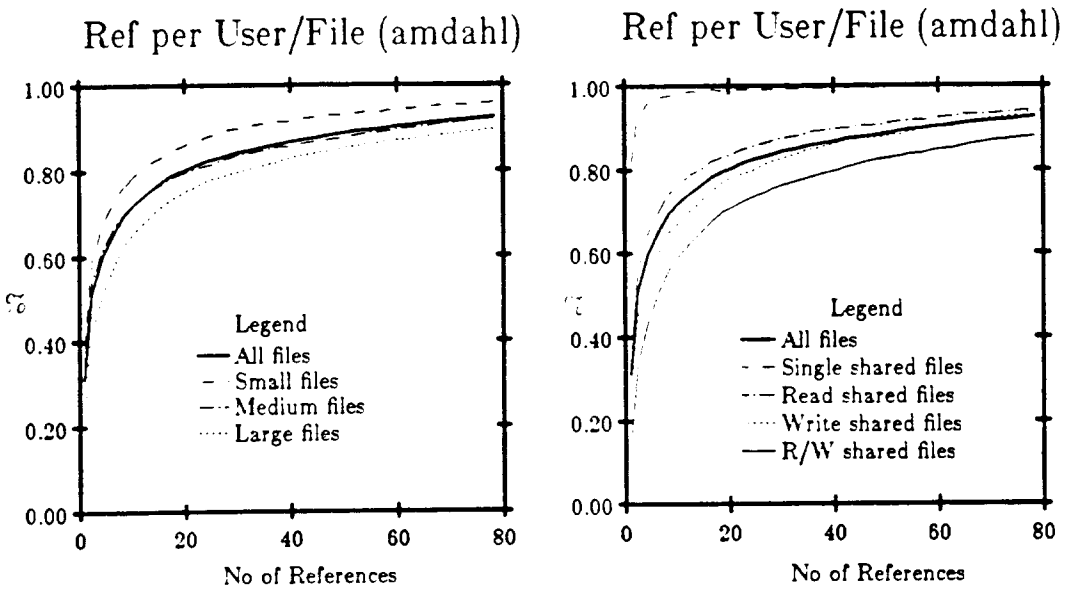
producer/one-consumer situation. The files in this group should mostly be logging files.

Read/write-shared files. These files are referenced in both read and update mode by at least two users. This group is the most interesting one, since the files are shared both in update and read mode. They are therefore more difficult to place.

The results from the analysis according to sharing behavior are contained in the 4 right most columns of Table 2.3 and in Figure 2.6. Read-shared files are the largest group among the shared files for two of the traces, while single-shared files are the largest group for the trace from Hughes. However, for this trace nearly 50% of the single-shared files are files referenced by only one user that transfers any bytes to or from the file.

All the measures we collected for files with the same sharing behavior had large coefficients of variation. There are also differences between the reference pattern at the three installations for files with the same sharing pattern. This is best illustrated by the read-shared files. At SLAC, they behave as one would expect library file to do, while at Amdahl they are updated much more frequently. At SLAC, the read-shared files have the highest average number of users sharing them, the highest average number of references per file, only 37% of them are updated, and the average number of updates per read-shared file is less than 2. At Amdahl, the read-shared files display a different behavior. Except for single-shared files, they have the lowest average number of opens, the lowest average number of users, 69% of them are updated, and the average number of updates per read-shared file is larger than 12. To summarize, the content of the read-shared files is more dynamic at this installation compared to the other ones. Even among files with

Figure 2.6c: The left figure displays the distribution of the number of opens per user/file pair with the files classified into three groups according to size. The right figure displays the same distribution but with the files classified into 4 groups according to how they are shared. All the distributions are for the trace from Amdahl.



the same sharing pattern, there will be differences in the reference behavior between the various installations.

The read/write-shared files are the ones that represent the largest potential placement problem since these files are both updated and read by at least two users. These files also have the largest average file size.

We expected that most read/write-shared files would be partitioned files, because the various members of a partitioned file can be read and updated by different users. However, from 5% to 71% of the all the read/write-shared files were sequential files. Another feature of the read/write-shared files is the low number of opens in update mode; typically, the files are opened in read mode. Less than 23% of the opens to these files are in update mode. Similarly, at most 66% of the users referencing a read/write-shared files will ever do so in update mode.

The reference pattern to write-shared files is similar to the one for a logging file; most of the opens are in update mode, and the majority of the files are sequential files. However, these files are not overlooked system files that should have been filtered out. The average number of users accessing these files is low ($\leq$ 3.3), and they are predominately used by only a few users for their own internal logging.

Table 2.3a: The table contains various measures from the trace from SLAC classified according to size of the shared files and how the files are shared. The different classification schemes are described in the text.

Slac	all shared files	small files	medium files	large files	single shared files	read files	write shared files	read/ write shared files
no of files	427	188	110	129	24	228	116	59
no sequential files	163	98	14	51	8	62	90	3
no partitioned files	249	83	89	77	16	163	14	56
average size (Kbytes)	1043.08	78.82	440.16	2962.46	1043.08	600.98	1129.30	1091.36
average no opens per file	92.03	101.19	87.07	82.91	2.33	127.50	32.09	109.29
average no opens in update mode per file	11.56	16.54	8.24	7.15	0.92	1.94	31.24	14.37
average no users accessing a file	4.78	4.10	5.40	5.26	1.54	6.21	2.97	4.17
average no users accessing a file in update mode	1.43	1.39	1.45	1.48	0.71	0.38	2.97	2.78
% of files not updated	34.9%	41.5%	30.0%	29.5%		62.3%		
average no opens per user/file pair	19.23	24.67	16.12	15.77	1.51	20.53	10.82	26.21

Table 2.3b: The table contains various measures for the trace from Hughes classified according to size of the shared files and how the files are shared. The different classification schemes are described in the text.

Hughes	all shared files	small files	medium files	large files	single shared files	read files	write shared files	read/ write shared files
no of files	5666	4710	654	302	3203	1881	343	239
no sequential files	5066	4392	535	139	3050	1648	197	171
no partitioned files	433	209	88	136	111	227	30	65
average size (Kbytes)	292.48	34.70	445.47	3981.50	134.39	353.63	853.49	1124.63
average no opens per file	9.04	5.89	14.24	46.85	2.24	12.17	24.01	54.05
average no opens in update mode per file	3.13	2.47	4.04	11.38	1.18	1.74	22.07	13.05
average no users accessing a file	2.16	1.90	2.60	5.37	1.55	2.70	2.74	5.42
average no users accessing a file in update mode	0.95	0.85	1.20	1.84	0.70	0.77	2.69	3.10
% of files not updated	24.6%	27.4%	8.8%	14.2%		22.7%		
average no opens per user/file pair	4.18	3.10	5.48	8.73	1.45	4.51	8.77	9.98

Table 2.3c: The table contains various measures for the trace from Amdahl classified according to size of the shared files and how the files are shared. The different classification schemes are described in the text.

Amdahl	all shared files	small files	medium files	large files	single shared files	read files	write shared files	read/ write shared files
no of files	2485	1024	897	564	512	1316	124	533
no sequential files	885	506	272	107	372	381	65	67
no partitioned files	1463	480	576	407	130	904	16	413
average size (Kbytes)	1905.50	63.72	530.76	7435.89	1033.73	1278.56	2254.65	4209.65
average no opens per file	95.85	41.53	85.43	211.02	4.13	59.18	68.41	280.87
average no opens in update mode per file	23.76	10.41	27.21	42.50	2.09	11.84	62.68	64.95
average no users accessing a file	3.29	2.62	2.99	4.99	1.68	2.95	3.32	5.67
average no users accessing a file in update mode	1.31	1.08	1.27	1.81	0.80	0.69	3.29	2.87
% of files not updated	20.4%	24.6%	16.9%	18.4%		31.0%		
average no opens per user/file pair	29.12	15.83	28.59	42.31	2.45	20.05	20.59	49.54

The high average number of references per user/file pair for read and read/write-shared files (see Table 2.3) is mostly caused by the behavior of a few files. Except for single-shared files, there is only a small difference in the distribution of the number of opens per user/file pair between the different groups. For all traces and all groups, the median is less than 10 references per user/file pair. The estimation problem we have discussed previously will also apply to the different groups.

To summarize, read-shared files are used differently at the different installations. Read/write-shared files are typically large files, and from 5% to 71% of these files are sequential files. However, read/write-shared files are predominately used in read mode, and at most 23% of the opens is in update mode. Similar to the previous analysis, we found that most users will reference a file only a few times (less than 10 times), regardless whether the file is read and/or write shared.

2.3.3.8. File Content

Two of the installations, Amdahl and Hughes, followed the TSO naming convention "userid.file-name.qualifier". The qualifiers are standardized and accurately describe the content of the file (see Table 2.4). In most cases, the meaning of these qualifiers is obvious.

At SLAC, each user had a partitioned data set where the shorter files were kept. The qualifier for this set was usually *lib*. Otherwise, no naming convention was strictly followed, and the content at SLAC is difficult to compare with the other installations.

Table 2.5 contains an overview of the file content for shared files at Hughes and Amdahl. In addition, the table shows how files with a particular content were organized. As expected, the three file systems contain different types of files. A priori, we thought most of the shared files would be TSO, job control language files and load files (with qualifiers CLIST, CNTL and LOAD). These files are typical library files that tend to be shared between users. As Table 2.5 shows, both data and source language files were shared. In addition, there is a large group of files we could not identify the content of.

The analysis of shared files by content was not helpful. The two installations, Hughes and Amdahl, displayed few common features in the reference patterns to files with the same content. A contributing factor was that the installations did not use the same file structure for files with the same content. As an example, most source code files are partitioned files at Amdahl, while they are stored as sequential files at Hughes. We previously showed that file use was correlated with the storage structure. It is therefore difficult to judge whether any difference in reference patterns with different content is due to 1) the installation, 2) the difference in file organization, or 3) file content. We found no apparent pattern and did not include the results in the analysis. However, for each installations there are differences in reference patterns to files with different content. Unfortunately, these differences are not replicated at both installations.

Table 2.4: The table describes the most common qualifiers used in the installations that used the TSO naming convention.

qualifier	contains
ASM	Assembler source code
CLIST	TSO commands and subcommands
CNTL	JCL and SYSIN for submit command
COBOL	Cobol source code
DATA	upper case text
FORT	Fortran source code
LINKLIST	Output listing from linkage editor
LIST	Listing
LOAD	Load module
LOADLIST	Output listing from loader
OBJ	Object module
OUTLIST	Output listing from output command
PLI	PLI source code
TESTLIST	Output listing from test command
TEXT	Upper and lower case text
VSBASIC	Vsbasic source code
GxxxxVyy	generation qualifier created for files with several versions or generations.

Table 2.5: The table contains the content of the shared files for all three traces. In addition the file organization is also included in the table.

Qualifier	# of files	# of sequential files	# of partitioned files
Slac			
lib	44	0	44
loadmods	19	1	18
not identified	364	162	187
Hughes			
data	516	461	43
cntl	202	150	45
clist	173	101	70
cobol	70	68	2
fort	141	125	8
pli	61	56	5
asm	20	9	10
txt	11	10	0
obj	38	31	7
load	88	0	88
Gxxyy	3921	3919	1
not identified	425	136	154
Amdahl			
data	352	127	221
cntl	397	61	335
clist	187	64	122
cobol	19	2	17
fort	14	2	12
pli	82	10	72
asm	82	7	75
txt	37	13	24
obj	74	7	67
load	129	2	127
not identified	1112	590	391

2.4. Conclusion

Management of shared files is important for file system performance. Even though shared files are a small fraction of all permanent disk files, they generate a much larger fraction of the total I/O traffic (measured in bytes). The shared files do not have uniform reference patterns. They vary in size, number of opens per file, and number of users referencing them. This is also displayed by the large tail of the distributions of the various measures, and the reference patterns cannot be characterized by a single set of fixed parameters common to all files and users.

We found that most users typically reference a file only a few times ($\leq$ 10 times), and file placement algorithms cannot be based on having accurate estimates before the placement decision can be made.

In two of the traces there is a tendency for large files to be opened more often than small files, while for the last trace, SLAC, the tendency is the opposite; small files are likely to be opened more often than large files. There is also a clear relationship between

file activity and file organization; sequential files are smaller, accessed by fewer users, and have fewer references per file than the files organized as direct, index sequential and partitioned files.

We identified four different patterns of sharing: a producer/consumer relationship between two users (single shared files), files shared solely in read mode, files shared solely in update mode, and files shared both in read and update mode. Except for one trace, the read shared files are most numerous. However, on the average the read/write shared files are referenced most often both in read and in update mode. The read, write, and read/write shared files have very similar distributions for the number of opens per user/file pair. How often a user will reference a file seems to be independent of how the file is shared (again except for single shared files).

For environments similar to the ones we analyzed, it is not valid to assume an open will access the whole file. The median is less than 80% of the file accessed per open, while the mean is less than 60%. The distribution for sequential files is skewed toward large fractions accessed per open, but there are a number of opens that append only a small fraction to the sequential files. The median for other file organizations is substantially small than for sequential files.

We also conclude that the order users access a file is not random. Instead, the typical pattern is several opens from the same user inside a short interval, followed by a longer interval before a different user starts accessing the file. This aspect of the reference pattern will be investigated further in the next chapter.

Chapter 3
Statistical Analysis of File Reference Patterns

Summary

This chapter contains the analysis of the stochastic properties of references to shared files. The best file placement strategy is to occasionally relocate some files, and stochastic models are needed to determine when and where files should be relocated. We find that the number of bytes transferred per open is adequately characterized by a constant for each user/file pair. The time between opens from a user to a same file can be represented by a renewal process with a high coefficient of variation, while the number of consecutive opens from the same user to a file can be modeled by a geometric distribution.

Four different models, a stack model, a Markov chain model, a modulated Poisson model, and a batch Poisson model were evaluated. The first two models only deal with the order users access files, while the last two incorporate the time between opens. The batch Poisson model with geometric batch sizes is the most appropriate one of the models we evaluated, and it has all the required properties. In such a model, a user will issue batches of opens to a file. The time between batches has an exponential distribution, and the time between opens in the same batch is substantially shorter than the time between batches. The batch Poisson model is also a plausible model of user behavior.

3.1. Introduction

This chapter presents the initial results of a statistical study aimed at determining appropriate stochastic models of how users reference files. We concentrate on models relevant to file placement in distributed systems. These models are used in Chapters 4 and 5 to develop suitable file migration and file replication algorithms. The performance of these algorithms are then evaluated with trace driven simulation.

In a distributed system with several storage locations, the cost or delay of referencing a file from a particular processor (node) depends on where the file is stored relative to the accessing node. Ideally, we would like files to be stored at or "near" the processor where they are used. This requires knowledge about future file reference patterns. While such information is unobtainable, we can develop stochastic models which predict, with reasonable accuracy, future reference patterns to files.

We derive our models from analyzing traces of system activity from three different installations. Our methodology is similar to the one used by Smith in his analysis of long term file reference patterns for hierarchal file migration [Smith81b], by Revelle in his analysis of file reference patterns [Revell75], and by Lewis and Shedler in their analysis of program references [Lewis73]. In the previous chapter, an exploratory analysis was conducted to provide a description of the traces. For that analysis, we relied mostly on descriptive tools like graphs and tables. The results from the exploratory analysis are used as a basis for the statistical analysis, but for brevity the results are not repeated here.

The first part of the section justifies why stochastic models are needed. To occasionally relocate some files can reduce the network traffic. Stochastic models of the reference pattern are therefore needed to determine when and where files should be relocated.

The main part of the chapter analyzes appropriate stochastic models. The analysis focuses on three aspect of the file reference process: the number of bytes accessed per

open, the time between opens to a file, and the order users will reference a file. In Section 3 we justify why the analysis is limited to these aspects and also outline the approach used. The next two sections outline the specific statistical hypotheses and tests used. The results of the analysis are in Section 6, where we explain why a Poisson process is not a good model of file references; it underestimates the coefficient of variation for the time between references.

Section 7 analyzes four different models of the reference process to a file; a stack model, a Markov chain model, a modulated Poisson model, and a batch Poisson model. The section discusses both the the features of the models and their implications.

We think a batch Poisson model with geometric batch sizes is the most appropriate model. In such a model, a user will within a short period issue a batch of opens to a particular file. The batches themselves are issued according to a Poisson process, and the number of references in each batch has a geometric distribution. The differences between the various models are discussed in more detail in the next two chapters, where we look at file placement in single copy file systems and in file systems that allow replication.

3.2. Motivation for Stochastic Models

This section justifies why stochastic models of the reference process are necessary. In a file system that does not allow replication, the minimum network traffic is obtained by occasional relocation of some files, and detailed stochastic models are needed to determine when these relocations should take place.

In the previous chapter we only showed that management of shared files will affect the system performance, since the I/O traffic to these files represents a substantial part of all bytes transferred to disk. However, there are several placement schemes that don't require any detailed stochastic models. The strategy of always moving the file to the node referencing it (MRU) is such an example. This strategy is based on the implicit model that the current user most likely will continue to use the file. MRU will then be optimal as long as the advantage of local I/O is larger than the cost of creating the copy. Another example is the simple static placement of always placing the file at the first node to use it. Neither these nor other static placement schemes are the optimal ones. Instead, a scheme that selectively relocates some files can have a lower network traffic, and stochastic models are needed to develop schemes that will work well.

To illustrate this claim, we measured the number of bytes transferred to/from a file for consecutive opens by the same user. If the number of bytes transferred by consecutive opens is smaller than the file size, the MRU scheme will increase the network traffic; the file transfer traffic will be larger than the remote network traffic avoided by the transfer. However, if the number of bytes from consecutive references is larger than twice the file size, MRU will always reduce the network traffic; the advantage of local access is larger than the transfer traffic to and from the new location. The number of bytes accessed by consecutive opens is therefore a good measure of whether either static placement or the MRU scheme is the best placement strategy.

3.2.1. Motivation: Results and Conclusion

To judge the effect of a particular file placement scheme, it is necessary to make some assumptions about how users are placed in the system. We assume each user is placed at an individual processor, and that all jobs from one user will run on that processor (node). Even though this is an extreme placement, we show in the next chapter that this is also a reasonable approximation for random placement in a large system (more than 10 nodes). This is not the only possible scheme for user placement, but it is a

plausible one. It is discussed in more detail in the next chapter.

We use the number of bytes transferred to/from a file by consecutive opens from the same user to illustrate the need for stochastic models. For the measure to be useful, it is normalized with the file size before the first of the consecutive opens, i.e., the potential transfer size. Figure 3.1 displays the distribution of this normalized measure for all sets of consecutive opens to shared files. The mean is 1.4, 0.96, and 0.86 (measured in units of file size), and from 89% to 96% of all measures are smaller than the file size. Clearly, the MRU scheme is wasteful; in at least 89% of the cases, the file transfer creates more network traffic than it saves.

Unfortunately, the number of bytes transferred by consecutive opens from the same user does not convey any information about the optimality of static placement. Even if there are sets of consecutive opens that accessed more than twice the file size, static placement may still be optimal; if, for each file, the sets belong to only one user, static placement is the optimal scheme. We therefore collected two additional distributions, one where all opens from the first user to reference the file were excluded, and one where all opens from the user with the largest amount of bytes transferred to the file were excluded. With network traffic as the metric, the best static location is the node with the largest number of bytes transferred to/from the file. The first distribution illustrates the advantage of moving the file away from the first node to access it, while the second distribution illustrates the potential advantage of relocating the file to nodes other than the optimal static location.

As we see in Figure 3.1 there are users and files that have measures larger than 1. From 0.7% to 1.8% of the observations are larger than 2, even when we exclude opens from users at the optimal static location. To relocate the file to these users (users that access more than twice the file size contiguously) is guaranteed to reduce the network traffic; the advantage of local I/O traffic is larger than the transfer traffic of the file to and from the new location. It can also be advantageous to relocate files for sets of consecutive opens that access between one or two times the file size, but this depends on the order users reference the file. Regardless, the measurements show that the network traffic can be reduced by occasionally relocating a file away from the best static placement. Static placement is therefore not the best possible file placement scheme. As we can see from Figure 3.1, the advantage of relocations is larger if static placement at the first node to reference the file is used instead of the optimal static placement.

The real advantage of occasional relocation of files cannot be judged from figure 3.1. The figure gives only the advantage or disadvantage of relocation for each set of consecutive opens. This is best illustrated with an example. Let us assume a file is sequentially shared by two users, *a* and *b*. User *b* will not open the file until user *a* has finished referencing it. These two users represent most of the opens to the file. In addition, several other users are randomly referencing the file. However, none of these users reference the file often enough to justify migrating it to them. The best strategy is to first place the file at user *a*, and then move it to *b* when user *a* has finished with it. Since the two users, *a* and *b*, share the file sequentially, both of them can get local reference to the file at the cost of only one file transfer. All opens from the other users are handled remotely, since none of them reference the file often enough to justify migrating the file to them. The advantage of this strategy cannot be calculated from the benefit of moving for each set of consecutive opens. There will be several changes in locality caused by the random references by the other nodes, which should not result in a file move. The average number of bytes transferred per set of consecutive opens from a node is then not a measure of the advantage of moving the file from *a* to b. The measurements are only useful for comparing policies that treat all users the same, like MRU or static placement. In

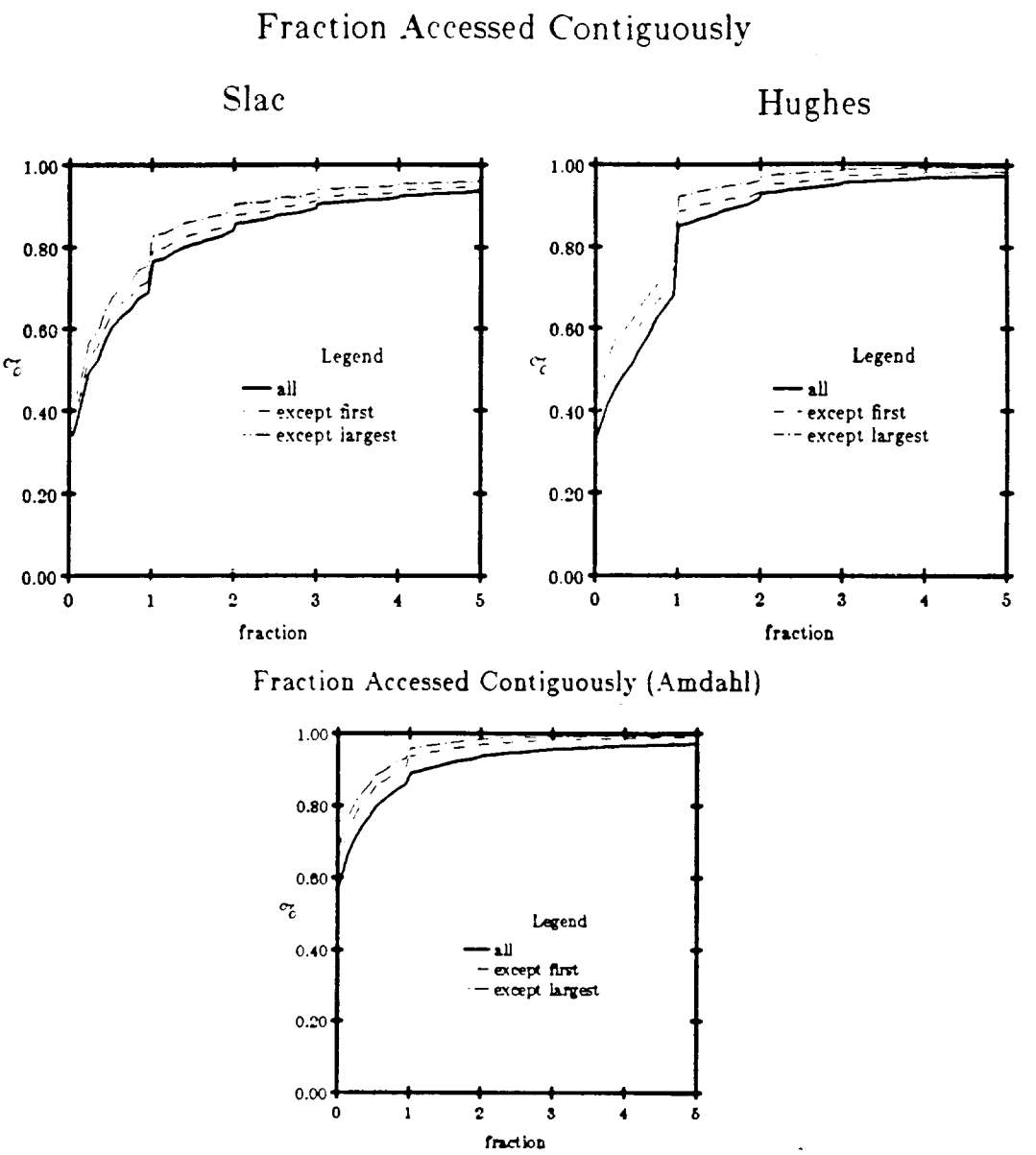


Figure 3.1: The figure displays the distribution of the total fraction of a file accessed by consecutive opens from the same user. The left figure is from the trace from SLAC, while the right one is from the trace from Hughes and the bottom one from Amdahl. The solid line is the distribution when all sets of consecutive references are included. The two other lines are the distribution for a sub-set of the opens. The dashed line is the distribution when all opens from the first user are excluded. The dashed-dotted line is the distribution when we exclude all opens from the user with the largest number of bytes transferred to the file. Why these distributions are of interest is further explained in section 3.2.1.

these policies, all changes in locality will result in the same action, either the file is moved or it is never moved, and the advantage of the policy is equal to the average advantage for a set of consecutive opens.

would, however, require stochastic models to determine when and where these relocations should take place. In addition, the best static placement is a posterior placement that is unknown until the file is erased [Smith85b]. Stochastic models may therefore be needed to identify the best static placement while the file is still being used.

3.3. Scope and Approach of the Study

3.3.1. Scope

We are only considering file migration/placement algorithms that make placement decisions when files are opened. All migration/placement algorithms must make a decision before or when there is a binding between a file name and a storage location, i.e., before or when a file is opened. The migration decisions could be made while the file is open, i.e., as part of the individual I/O operations. Such algorithms could potentially make "better" decisions, but the overhead of the I/O calls will increase, and we may also incur problems migrating the state information of the open file [Powell83]. In addition, the analysis of this class of algorithms is beyond the scope of our study, since the individual I/O operations have not been traced.

In order to place files at the "best" location, ideally we should be able to predict the future references to the file. Among the variables of special interest are:

- the time until the next open to a file, so the event time of all future references can be predicted.
- the number of bytes to be transferred (read and/or written) while the file is opened, to judge the advantage of relocating a file.
- the file size as a function of time, to judge the the cost of future file moves.
- the mode of an open (read, append, or update).
- the order users will access a file, to determine when a file should be relocated.
- the time until the file is erased, to judge the potential benefit of relocating.

These variables cover all aspects of the file reference process that are of interest for different file placement schemes.

3.3.2. Aspects not Analyzed

Only three aspects of the file reference process are analyzed in more detail. These are the bytes transferred per open, the time between opens, and the order users access a file. This section discusses why the other aspects are not analyzed; they are either not important or the traces cannot be used to analyze them.

To simplify our models, we do not consider the duration of an open or the individual I/O operations. File references are viewed as atomic operations with all I/O operations performed immediately when the file is opened, but after the migration decision is made. This simplification has little effect as long as we are only considering file migration strategies that make placement decision solely when files are opened.

By disregarding the duration of an open, we overlook the effect concurrent opens from several users can have on the file placement. With several concurrent opens, a migration of the file may change the accessing cost for some of the users already using the file. This could then affect the migration decision. Files can, however, only be opened simultaneously by several users in read mode. Moving a file that is opened concurrently by several users in read mode will not affect any of the users, since a temporary copy of the file can remain at the old location. The nodes that had opened the file before it was

moved can then continue to use the temporary copy at the old location, and the accessing cost will remain unchanged. The effect of not modeling the duration of an open will therefore be small.

The mode of an open (read, append, update, or overwrite) is initially not modeled, since it mostly affects file placement with multiple copies (replication). When a file is opened in update mode, the changes to the file must be migrated to all copies to preserve the consistency of the file. File systems with replication will be considered in Chapter 5. For file systems with only a single copy, whether a file is opened in read, update or append mode will not affect the number of bytes transferred per open. The overwrite mode could potentially affect the cost of migrating a file to a new location. Each overwrite can be viewed as erasing the old copy and creating a new file with the same name. Provided the file is not read before it is overwritten, opens in overwrite mode should not create any network traffic; a new version of the file is created locally, while the old version is erased. In our traces, it is impossible to distinguish between bytes read or written, and the number of bytes read before the file was overwritten cannot be determined. To avoid over-estimating the advantage of relocation, an open in overwrite mode cannot be equated with creating a new copy and erasing the old one. Instead, we assume each overwrite can potentially result in network traffic equal to the number of bytes transferred per open in overwrite mode. As we will show in the next chapter, this conservative assumption does not affect our results, and the mode of an open is not important to model.

The file size was not investigated. When a file is opened the size is known and need not be predicted, but a part of the placement decision will be to evaluate the cost of moving the file back again to the current storage location. This cost is a function of the file size. However, updates to the file will seldom substantially change the file size. Most of the opens that appended to a file increased the file size by only a small fraction (see Chapter 2). At least initially, it is not necessary to predict these changes accurately.

The distribution of a file's lifetime is not analyzed, since the available traces are not suited for such an analysis. The traces have a relative short duration (8 to 30 days) and they can only provide lifetime information about files that are both created and erased during the trace period. Instead, we rely on results from other studies [Smith81b], which traced files with less granularity than we do, but for a longer period.

3.3.3. Aspects Analyzed

The three remaining aspects that are analyzed are the bytes transferred per open, the time between opens, and the order users access files. The analysis for the first of these aspects, the bytes transferred per open, is limited, since a simple model with a constant fraction of the file transferred per open is appropriate. The results of the analysis are therefore contained in this section. The analysis of the two other aspects, the time between opens, and the order users access files, is more complex. It is the topic for most of this chapter. In this section we will therefore only discuss the scope and approach for the analysis of these two aspects.

3.3.3.1. Bytes Accessed per Open

Network traffic is used to measure the performance for the various placement schemes. The cost function is then proportional to the number of bytes transferred per open, and the mean number of bytes transferred per open is a sufficient characterization. For a particular file with size S, let $f(b \mid S)$ be the density function for the number of bytes transferred per open with mean $E(B \mid S)$ and set the transfer cost per byte to c. The

expected cost of an open is then:

$$\int c \cdot b \cdot f(b \mid S) db = c \cdot E(B \mid S)$$

This implies the mean is sufficient to determine the expected cost of an open for a particular file.

As long as the number of bytes transferred is constant or an independent random variable, previous reference history will not affect the mean. The goal of this analysis is to determine the correlation between previous reference history for the file and the expected number of bytes transferred per open. The analysis is focused on the one step correlation coefficient for the number of bytes transferred per open for each user/file pair. Instead of using the number of bytes accessed per open directly, we used the number of bytes divided by the file size (the fraction accessed). Any changes in the size of a file will then not affect the one step correlation coefficient. Otherwise, for users that accessed the whole file, the one step correlation coefficient would be different from 1.0 when the file changed its size.

In the previous chapter we showed the fraction of a file accessed per open varied between the files. However, in this section we show that the expected fraction accessed per open can be modeled by a constant for each user/file pair, but the constant will different for the various files and users.

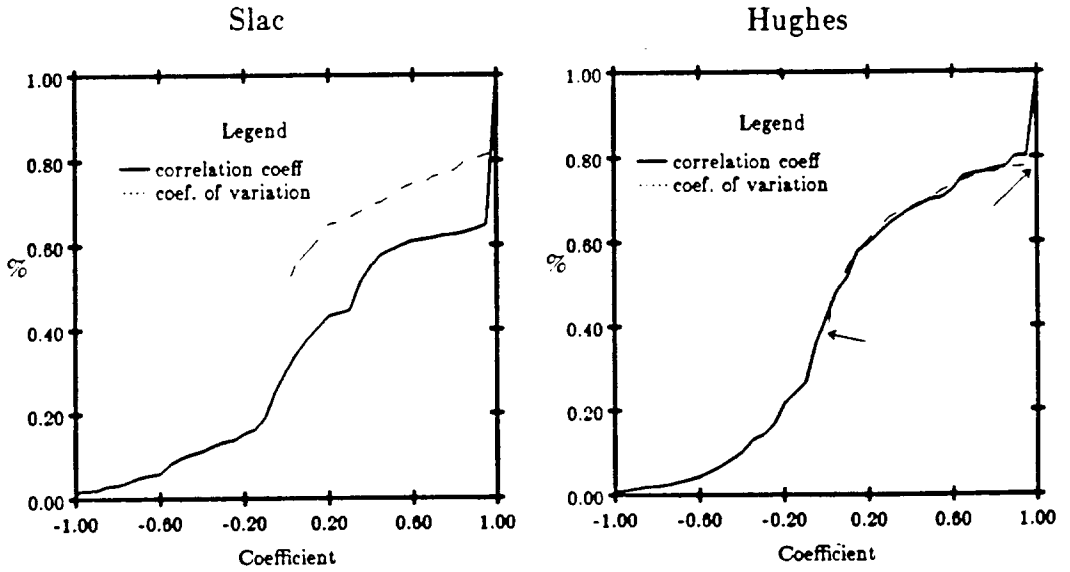


Figure 3.2a: The solid line is the cumulative distribution over all user/file pairs for the one step correlation coefficient for the fraction accessed per open. The dotted curve is the cumulative distribution for the coefficient of variation for the fraction accessed. The two coefficients are collected for each user/file pair with more than 4 opens, and the distributions are weighted with the number of opens for each pair. The distribution of the one step correlation coefficient is not adjusted for the bias. The left figure is for the trace from SLAC, while the right is for the trace from Hughes.

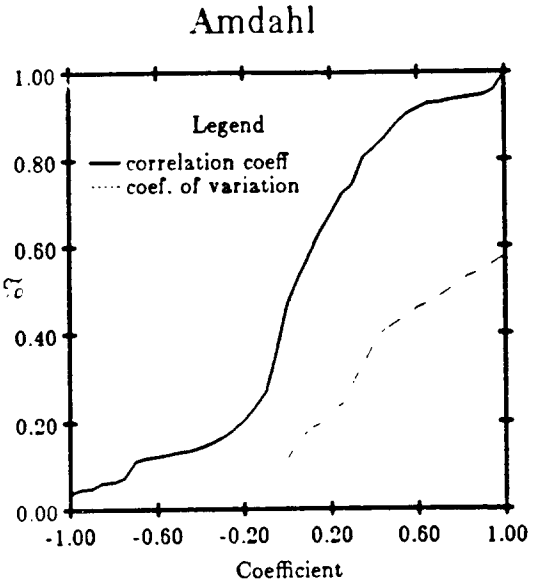


Figure 3.2b: The solid line is the cumulative distribution over all user/file pairs for the one step correlation coefficient for the fraction accessed per open. The results are from the trace from Amdahl. The dotted curve is the cumulative distribution for the coefficient of variation for the fraction accessed. The two coefficients are collected for each user/file pair with more than 4 opens, and the distributions are weighted with the number of opens for each pair. The distribution of the one step correlation coefficient is not adjusted for the bias.

Figure 3.2 displays the distribution for the one step correlation coefficient for the fraction accessed per open for all user/file pairs that issue more than 4 references. The distribution is weighted with the number of opens issued by each user/file pair to reflect the properties of the "typical" open instead of the "typical" user/file pair. The estimate for the one step correlation coefficient has a negative bias of $-\frac{1}{n-1}$, where n is the number of opens for the user/file pair. The bias has little effect on the results, since more than 89% of the opens are from user/file pairs with more than 10 opens. The dotted line in Figure 3.2 is the distribution of the coefficient of variation for the fraction accessed per open. As for the one step correlation coefficient, the distribution is limited to user/file pairs with more than 4 opens, and the distribution is weighted with the number of opens for each pair.

For all three traces, some of the users will transfer a constant fraction of the file at each open. For the trace from SLAC, 35% of the opens come from users that have a one step correlation coefficient equal to 1, while the number for the trace from Hughes is 20% and 4.2% for the trace from Amdahl. On a per user/file pair basis, a larger percentage of the pairs have a coefficient equal to 1. The percentage is between 13% and 47%. A one step correlation coefficient equal to one need not imply a constant fraction accessed per open. For example, an autoregressive time series where the current value equals the old value plus a random noise will have a one step correlation coefficient equal to 1, while the

values in the time series will fluctuate. For our purpose it does not matter whether the fraction accessed is a constant or a constant distorted by random noise. With a one step correlation coefficient equal to 1.0, the best predictor for future values is the average of the fraction accessed per open for that particular user/file pair.

For the remaining user/file pairs, the average is the best model we can find. A substantial fraction of the opens have one step correlation coefficients close to 0.0. In the trace from SLAC, 18% of the opens have a one step correlation coefficient between -0.2 and 0.2, while the percentage for Hughes and Amdahl are 29% and 47%. For these opens, the fraction accessed can be modeled by an independent random variable. The expected fraction accessed is then a constant independent of previous history, and we use the observed average for each user/file pair as an estimate.

The remaining $\approx$ 50% of the opens have one step correlation coefficients in the range 0.2 - 1.0 and below -0.2. The fraction transferred can therefore not be modeled as an independent random variable, and using a constant for the expected fraction accessed may no longer be the best predictor. Unfortunately, no other simple models are appropriate. An analysis of user/file pairs with more than 100 references showed the fraction accessed per open for most of them is not 1) a first or second order autoregressive process, 2) a first or second order moving average process, or 3) a second order combined autoregressive and moving average model. Any model of the fraction accessed per user/file pair must therefore be of at least of 3rd order, and a simple Markovian model is clearly not appropriate.

Only a few files are opened frequently enough to estimate the parameters in any model more complicated than a first order Markov model. We therefore have no other choice than to use the simple model with a constant fraction accessed per open. This is a satisfactory model, since most of the opens will have a low coefficient of variation for the fraction transferred (on a per user/file pair basis); more than 70% of the opens have a coefficient smaller than 1. As already mentioned, from 4% to 35% of the opens transfer a constant fraction of the file for each open. An additional 47% to 18% of the opens transfer a fraction nearly independent of previous history. For all of these opens, the average observed so far is a reasonable predictor of the expected fraction transferred.

3.3.3.2. The Time between Opens and the Order Users Access Files

The remaining two aspects, the time between opens, and the order users access files are the subjects for the analysis. We are restricting ourselves to Markovian models based on the previous history of the variables, and do not consider any external factors like time of day, date, file content, project deadlines and so on. These external factors are excluded for a number of reasons: (1) Too many factors would make a model intractable, and may also tailor the model too closely to a specific trace. The model would then lose its predictive power for other traces. For example, for most files, the time of day and the date influence the time until the next opens, but to include them would make a model intractable. The results of the analysis would then also be valid only for the point in time when the traces were obtained. 2) Some of the external factors are not available in the system traces. For example, the various projects, their deadline, and the files used for a particular project are not recorded. (2) The external factors that were available in the traces did not provide any useful information. For example, the exploratory data analysis found no apparent pattern between the file content and the file references.

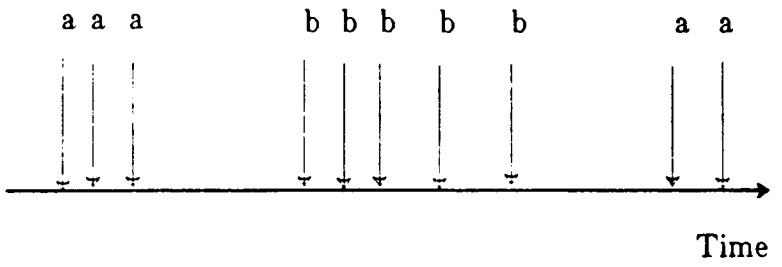


Figure 3.3: An example of the reference process to a file accessed by two users (A and B). There are a total of 10 opens between *max*(trace started, file created) and *min*(trace ended, file erased). The reference process from each user can be viewed as a point process.

3.3.4. Approach

As already mentioned, we view opens as atomic operations where all I/O operations take place immediately when the file is opened. The set of references made by a user to a particular file is viewed as point process independent of the reference processes from other users. The processes for all users sharing a particular file can be superpositioned to create a new point process, one for each file. Each event in the point process is characterized by time, number of bytes transferred, and user id. In the study, both the individual processes for each user/file pair and the superpositioned process for each file are analyzed. Figure 3.3 illustrates a realization of the reference process to a file. The reference process to the file is the superpositioned process of user *a*'s and user *b*'s references processes.

Since a large number of processes are analyzed, we can only test models with broad properties like stationarity, renewal, or type of distribution as was done in Smith's analysis [Smith81b]. Specifically, we tested whether the references to a file can be characterized as 1) a stationary process, 2) a renewal process, and 3) as a Poisson process. In addition, coefficient of variation for the various measures is collected.

In the exploratory analysis, it was observed that users tend to cluster their opens, like in Figure 3.3. A potential model of this locality is a geometric distribution for the number of consecutive opens from a user to a particular file. This simple model was the only one tested for the order users access files. If the model is rejected, a Markov chain model of the order users access a file, and a model similar to a LRU stack model, must also be rejected. A Markov chain model assumes that with constant probability p_{ij} user j will reference the file next, given user i was the last one to open the file [Easton75]. In the LRU stack model [Spirn77], the users referencing a file are ordered on a stack, with one stack per file. The current user is placed at the top of the stack, and with constant probability a_j a user at stack distance j will open the file next. Both of these models imply the number of consecutive opens from the same user has a geometric distribution.

3.4. Method

Statistical tests, formulated as testing of hypotheses, are used to test appropriate models. A typical test accepts a hypothesis if the deviations between a measure from the actual process and the expected one, given the hypothesis correct, is less than a set limit. The limit is calculated so with probability $1-\alpha$ the deviation will be less than the limit

when the hypothesis is correct. Usually, a level of $\alpha = 0.05$ is used.

Statistical tests are not always appropriate tools for finding models that are "good" abstractions. When only a few observations are available, the limits for accepting a hypothesis are large, and statistical tests are not able to distinguish between different models. The opposite effect can be observed when many observations are available; all models are rejected. Our purpose is to find a good abstraction of the file reference process. For a complex process like references to files, we can never hope to find the "correct" model, regardless of how many factors we include in our analysis. Therefore, given enough observations, all models, whether they are "good" abstractions or not, will be rejected. As an illustration, consider a time series with 10000 data points. If the first order correlation coefficient is 0.02, the time series will be rejected as being a renewal process at a 5% α level. For most modeling purposes, a time series with a first order correlation coefficient = 0.02 can be considered a renewal process. A rejection of a model, when many data points are available, only means the model is not identical to the actual process; the model can still be a "good" abstraction.

We take a more pragmatic view than the strict statistical testing. If the percentage of rejected cases is "reasonably" close to the α value, we accept the hypothesis as an acceptable abstraction of the actual process. What we consider "reasonable" varies from hypothesis to hypothesis. Among the factors we take into consideration are the importance of the rejected cases and how well the mean and variance of the model correspond with the mean and variance of the actual process. We use the fraction of opens to shared files from the rejected cases as a measure of their importance.

3.5. Statistical Tests

This section describes the various hypothesis formulations and rejection criteria used in the analysis. There are two sets of statistical tests used; the first is used to determine whether the time between references to files can be characterized as 1) a stationary process, 2) a renewal process, 3) a Poisson process. The second set consists of different tests to determine whether the number of consecutive opens from a user to a particular file has a geometric distribution. Most of the tests are taken from Cox and Lewis's book [Cox66].

Usually, only the approximate distribution of the test variable is known, and only for files and for user/file pairs with a reasonable number of observations will the approximation be valid. We did not try to extend the validity of our tests to cases with only a few observations, since these cases represent only a minority (usually only a few percent) of all opens made to shared files. Even though users with few opens represent a majority of the user/file pairs, they are excluded since there are too few observations to statistically distinguish between different modeling alternatives. The median is less than 4 opens per user/file pair (see Chapter 2). In addition, our primary interest is to develop models for the user/file pairs that represent most of the opens made to shared files. The distributions of the number of opens per user/file pair and per file are given as part of Table 3.2 and Table 3.3. Even though user/file pairs with more than 20 opens constitute about 20% (22% at Slac, 3% at Hughes, and 21% at Amdahl) of all user/file pairs among shared files, they represent the majority of all opens (78%, 49% and 89%). As a rule, all tests are only performed for user/file pairs and files with more than 20 opens. For some of the tests, however, the actual distribution of the test variable is known, and user/file pairs and files with fewer references can also be tested.

3.5.1. Tests for the File Reference Process

Stationarity. The centeroid test for monotone trends [Cox66] is used to determine whether the time between opens is a stationary process.
If this is the case, the references will be evenly distributed over time. Let T_j be the time from the first to the j+1'st reference to a file.
Let

$$S = \frac{1}{n} \sum_{i=1}^{n} T_i \qquad 3.1$$

where $n+2 =$ number of references to the file.
For a Poisson process, the events will be uniformly distributed over the trace interval t_0 and

$$E(S) = \frac{t_0}{2} \quad V(S) = \frac{t_0^2}{12n}$$

Since some of the files have a short life time, a fixed trace period could not be used, instead t_0 is defined as the time between first and last reference. The remaining n references are then uniformly distributed on t_0.

$$U = \frac{S - t_0}{\frac{t_0}{\sqrt{12n}}} \qquad 3.2$$

U's limiting distribution is a $N(0,1)$ as $n \rightarrow \infty$. We reject the stationarity assumption when $|U| > \Phi_{\frac{\alpha}{2}}$, where $\Phi_{\frac{\alpha}{2}}$ is the $\frac{\alpha}{2}$ quantile for the N(0,1). The accuracy of the test depends on how close the actual process is to a Poisson process. If it is overdispersive compared to a Poisson process, it can be shown that $V(S)$ will be overestimated, and the rejection criteria must be higher [Lewis73]. The test can therefore not be interpreted strictly, but just taken as an indicator as Lewis and Shedler did in their paper [Lewis73]. To give some room for errors in the estimation of $V(S)$, the test was performed at an α level of 0.05%. This corresponds to a rejection criteria 2.18 times higher than the correct one at a 5% level.

The normal approximation for the test variable, U, is only valid for large number of observations. A simulation study of the actual distribution of U showed that the α quantile is smaller than the α quantile of the N(0,1) distribution. If the hypothesis is correct, the test will reject fewer than α percent of the cases when it is used for cases with only a few observations. In these cases the test is conservative, and it is used for cases with as few as 6 observed references.

Renewal. If a process is a renewal process, the one step correlation coefficient ρ_1 is 0.0. If the absolute value of the empirical coefficient $\tilde{\rho}_1$ is larger than K, the renewal hypothesis is rejected. For a renewal process with n observed events $\rho_1\sqrt{n-1}$ will have a $N(0,1)$ as the limiting distribution. K is therefore set to $\Phi_{\frac{\alpha}{2}}$. The estimator $\tilde{\rho}_1$ for ρ_1 is taken from Cox's book [Cox66]. Let X_j be the time between reference j and j+1. The estimator $\tilde{\rho}_1$ is then given by:

$$\tilde{\rho}_1 = \frac{C_1}{\sqrt{V_1' \cdot V_1''}}, \qquad 3.4$$

where

$$C_1 = \frac{1}{n-1} \cdot \sum_{i=1}^{n-1} (X_i - \bar{X}')(X_{i+1} - \bar{X}''), \quad \bar{X}' = \frac{1}{n-1} \sum_{i=1}^{n-1} X_i, \quad \bar{X}'' = \frac{1}{n-1} \sum_{i=1}^{n-1} X_{i+1},$$

$$V_1' = \frac{1}{n-1} \cdot \sum_{i=1}^{n-1} (X_i - \bar{X}')^2, \quad V_1'' = \frac{1}{n-1} \cdot \sum_{i=1}^{n-1} (X_{i+1} - \bar{X}'')^2$$

The number of opens to the file is $n + 1$. The renewal hypothesis is rejected when $|\tilde{\rho}_1| \cdot \sqrt{n-1} \geq \Phi_{\frac{\alpha}{2}}$, the $\frac{\alpha}{2}$ quantile for $N(0,1)$. Since $E(\tilde{\rho}_1) = -\frac{1}{n-1}$ and ρ_1 is only approximately normally distributed, the renewal hypothesis was only tested for user/file pairs and for files with more than 20 opens. The test is therefore restricted to from 3% to 20% of all the user/file pairs which represent from 49% to 90% of all opens made to shared files.

Poisson hypothesis. The events in a Poisson process are distributed uniformly over a fixed interval. In a Poisson process, observed over k intervals of equal length t, the number of events in interval i, n_i, has a multinomial distribution. D, defined as

$$D = \frac{1}{\bar{n}} \cdot \sum_{i=1}^{k} (n_i - \bar{n})^2 \quad \bar{n} = \frac{1}{k} \sum_{i=1}^{k} n_i \qquad 3.5$$

has a χ^2_{k-1} as its limiting distribution. The Poisson hypothesis is rejected for user/file pairs and files with D $>$ α quantile for χ^2_{k-1}. For small values of n ($<$10) the exact distribution of D is used, instead of the χ^2 approximation. D is a discrete variable, and only for certain values of k is it possible to find a distribution function with $\approx$ 5% quantiles. For each n between 10 and 4, the number of intervals was chosen so the distribution of D would have an approximate 5% quantile. The χ^2_{k-1} approximation was used for n $>$ 10 with k=5.

3.5.2. Test for the Order Users Access Files

We only tested if the number of consecutive opens from a user to a particular file (a run) has a geometric distribution. Let X_j be the number of consecutive opens in the j-th run for a particular user/file pair. We used $Y_j = X_j - 1$ instead of X_j in our tests, since the tests based on Y_j were easier to implement. If X_j has a geometric distribution with parameter p, Y_j has a geometric+1 distribution, and they have the following properties:

$$\begin{aligned}
&P(X_i = k) = p^{k-1}(1-p) && E(X_i) = \frac{1}{(1-p)} \\
&V(X_i) = \frac{p}{(1-p)^2}, && C(X_i) = \frac{V(X_i)}{E(X_i)^2} = p \\
&P(Y_i = k) = p^k(1-p) && E(Y_i) = \frac{p}{(1-p)}, \qquad 3.6 \\
&V(Y_i) = \frac{p}{(1-p)^2}, && C(Y_i) = \frac{V(Y_i)}{E(Y_i)^2} = \frac{1}{p}
\end{aligned}$$

To simplify the discussion we use the term "a run" for a set of consecutive opens from the same user to a particular file. For each file, the last run for the last user to access the file

is dropped , since it is truncated either by the end of the trace period or the deletion of the file.

In a sample space divided into k intervals, let e_i be the expected number of events in interval i for a geometric distribution. Let n_i be the observed number. Then

$$R = \sum_{i=1}^{k} \frac{(n_i - e_i)^2}{e_i} \qquad 3.7$$

is a measure how well the empirical distribution fits the geometric one. Cox [Cox66] recommends the intervals be set so the expected number of events in each interval is about the same and at least larger than 5. We stretched this criteria by testing all user/file pairs and files with more than 6 runs observed (k=2).

A simple difference sign test similar to the one found in Kendall and Stuart's book [Kendal75] was also used. This is not a powerful test and is only intended as a supplement to the goodness of fit test. If X_j (the number of opens in the j-th set of consecutive opens for a particular user/file pair) has a geometric distribution with parameter p, then the probability $P(X_j \geq X_{j+1}) = \frac{1}{1+p}$. From k observed runs we compared every even numbered run with its predecessor. The number of runs larger than the predecessor is then Binomial $(\frac{k}{2}, \frac{1}{1+p_i})$, and the geometric hypothesis is rejected if the numbers are too high or too low.

Table 3.1: Distribution of number of sets of consecutive opens from the same user to a file. The distribution is tabulated on a per user/file pair and per file basis.

	Slac			Hughes			Amdahl		
number of runs observed	number	% of total number	% of all references to shared files	number	% of total	% of all references to shared files	number	% of total	% of all references to shared files
User/File Pairs									
0- 9	1716	84.0	47.1	12063	98.4	76.6	7497	91.7	68.9
10-30	248	12.1	33.5	168	1.4	17.4	493	6.0	17.1
30-	79	3.9	19.4	28	0.2	6.0	189	2.3	14.0
Files									
0- 9	307	71.9	35.0	5530	97.6	56.3	2096	84.3	59.3
10-30	62	14.5	11.7	81	1.4	10.4	232	9.3	13.8
30-	58	13.6	53.3	55	1.0	33.3	157	6.3	26.9

Table 3.1 shows the distribution of the number of runs that can be used for the analysis. The distributions are on a per user/file pair and per file basis. The number of user/file pairs where more than 10 observations can be used is a small fraction of all pairs (from 1.6% to 15%), but they represent from 23% to 53% of all opens. The low numbers for the trace from Hughes are due to the large number of files referenced by only two users once or twice (single shared files). On a per file basis there are more cases (from

2.4% to 28%) with 10 or more observations that can be tested. These cases also represent a substantial larger fraction of all opens (from 41% to 65%).

A goodness of fit test cannot be used when only a few runs are observed. An additional test was developed to determine the property for user/file pairs with only a few runs. We based the test on the coefficient of variation for the number of opens in the first 2 or 3 runs. Assume k runs are drawn from a geometric distribution. Let n be the total number of opens in the k runs. These n opens can be combine into $\binom{n-1}{k-1}$ possible combinations. As an example, there are 3 different ways one could observe a total of 4 opens over 3 runs, 1-1-2,1-2-1, and 2-1-1. Each of these combinations have the same probability for occurring, and the coefficient of variation for the number of opens per run can be calculated. Owing to the computational complexity, the test was only performed for k=2 and 3.

All the test are performed at a 5% rejection level. The results are reported for the goodness of fit test and the combination of the goodness of fit test, the sign test, and the coefficient of variation test.

3.6. Results

This section contains the results of the statistical analysis. The first part justifies why it is appropriate to model file references from a user to a file as a renewal process, but not as a Poisson process; the coefficient of variation for the time between references is too high. In the second part we show why it is appropriate to model the number of consecutive references from a user to a file by a geometric distribution. The users sharing a file will, however, typically have different mean number of consecutive opens.

3.6.1. Results for the File Reference Process

The references from a particular user/file pair were tested for stationarity, renewal, and closeness to a Poisson process. The results are shown in Tables 3.2a, b, and c. Since the tested properties are hierarchal (a renewal process is a stationary process, and a Poisson process is a renewal process), the testing is also hierarchal; only user/file pairs with stationary reference processes are tested for renewal, and only those accepted as renewal processes are tested for closeness to the Poisson process. The third column of Table 3.2 shows the % of cases for which the stationarity hypothesis is accepted. The fourth column gives the percentage of all cases for which the renewal assumption is accepted, while the fifth column gives the percentage of all cases for which the Poisson hypothesis is accepted. Since the hypotheses are hierarchal, the numbers in column 3 are always higher than in column 4, which again are higher than those in column 5. The exceptions are cases with less than 20 opens, since the renewal hypothesis is not tested for these cases. The difference between columns 3 and 4 is the percentage of cases for which the stationary hypothesis was accepted but not the renewal hypothesis. A similar interpretation can be given for the difference between the fourth and the fifth column.

Stationarity hypothesis. The stationarity hypothesis is accepted for 80% to 90% of the user/file pairs, but the acceptance percentage decreases with the number of observed references. The coefficient of variation for the time between references is larger than 1, but the stationarity tests assumed it would be close to 1.0. The test values for stationarity will therefore be high, even when the processes are stationary (see Lewis and Shedler's paper [Lewis73]). We anticipated this to a certain degree by using a rejection criteria about 2 times higher than it should be at a 5% level. Unfortunately, the observed coefficients of variation are large and the test criterion should have been even higher (see Lewis and Shedler's paper [Lewis73]). The stationarity hypothesis can therefore not be

Table 3.2a: Results for the statistical test for the file reference process of user/file pairs at SLAC. The reference process for each user/file pair is tested to see whether it is a stationary, a renewal, and a Poisson process. All tests use an α level of 5%.

number references observed	no of user/file pair tested	% of all opens to shared files	% acceptance of stationarity hypothesis	% acceptance of renewal hypothesis	% acceptance of Poisson hypothesis	mean coefficient of variation
5- 9	333	5.7	100.0		76.6	2.49
10-19	324	11.0	96.0		43.5	4.25
20-29	130	7.8	87.7	83.1	27.7	5.38
30-49	138	13.2	89.1	86.7	13.8	6.92
50-	173	57.3	71.7	67.8	8.1	19.28
total	1098	95.0	91.5	78.5	42.4	6.66

rejected, in spite of it being rejected in from 10% to 20 % of the cases. For user/file pairs with many references (> 50) between 28% and 48% of the cases reject the stationarity hypothesis. These cases represent a large fraction of all opens to shared files (between 33% and 79%). On the other hand, these cases also have the largest average coefficient of variation for the time between accesses (between 19.3 and 31.7). At best, the test results are questionable. Instead we use the one step correlation coefficient for the time between opens to judge whether the reference process has a monotone trend or not. The one step correlation coefficients are clustered around 0.0 (see Figure 3.4), implying there is no monotone trend in the time between opens.

Table 3.2b: Results for the statistical test for the file reference process of user/file pairs at Hughes. The reference process for each user/file pair is tested to see whether it is a stationary, a renewal, and a Poisson process. All tests use an α level of 5%.

number references observed	no of user/file pair tested	% of all opens to shared files	% acceptance of stationarity hypothesis	% acceptance of renewal hypothesis	% acceptance of Poisson hypothesis	mean coefficient of variation
5- 9	815	10.4	99.5		78.8	2.63
10-19	473	12.3	93.9		36.6	4.74
20-29	150	7.0	90.0	88.7	22.0	7.71
30-49	115	8.3	80.9	78.3	7.8	10.57
50-	114	33.4	51.8	50.9	2.6	26.13
total	1667	71.4	92.5	74.1	51.6	5.84

Renewal hypothesis. The renewal hypothesis was tested only for those cases for which the stationarity assumption was accepted. The hypothesis is rejected by only a small percent (4.2%, 2.1%, and 3.3%) of the user/file pairs that are accepted as stationary ones. This implies a renewal process is a reasonable model of how users reference files. This is further confirmed by the distribution of the one step correlation coefficient for the time between references (Figures 3.4).

The solid lines are the distribution of the correlation coefficient for user/file pairs with more than 5 opens, while the dotted is for files. The plots are for all user/file pairs and not only those accepted as stationary ones. The distributions are weighted with the

Table 3.2c: Results for the statistical test for the file reference process of user/file pairs at Amdahl. The reference process for each user/file pair is tested to see whether it is a stationary, a renewal, and a Poisson process. All tests use an α level of 5%.

number references observed	no of user/file pair tested	% of all opens to shared files	% acceptance of stationarity hypothesis	% acceptance of renewal hypothesis	% acceptance of Poisson hypothesis	mean coefficient of variation
5- 9	1072	3.0	99.4		72.4	3.06
10-19	807	4.7	88.2		27.4	5.58
20-29	364	3.7	74.2	71.2	22.0	8.59
30-49	378	6.1	72.5	69.8	14.8	10.12
50-	946	79.1	56.9	53.1	5.9	31.66
total	3567	97.0	80.2	60.8	33.3	12.57

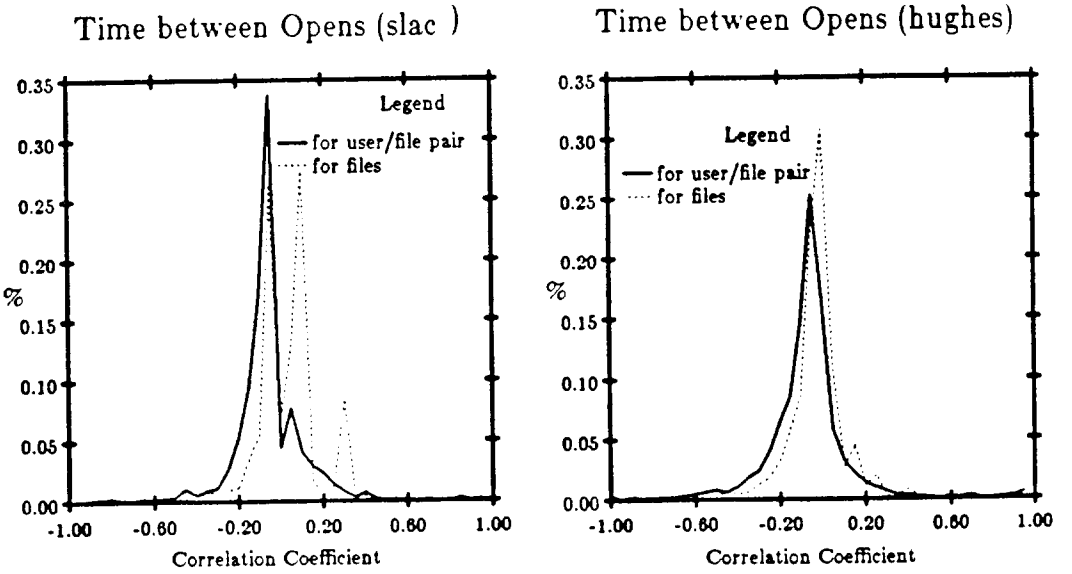


Figure 3.4a: The figure contains the distribution of the one step correlation coefficient for the time between opens. The solid line is for user/file pairs, while the dotted one is for files. All user/files or files with more than 5 opens are included in the the measurements, and the correlation coefficient are weighted with the number of opens for each user/file pair or file. The left figure is for the trace from SLAC, while the right is for the trace from Hughes.

number of opens for each user/file pair to represent the property of a "typical open" instead of the "typical" user/file pair. The results are identical if the time between closing a file was used instead of the time between opening a file. The distribution has most of its mass close to 0.0, with an average of -0.1 (SLAC), -0.13 (Hughes), and -0.07 (Amdahl). It is slightly skewed towards negative values since the estimator for the one

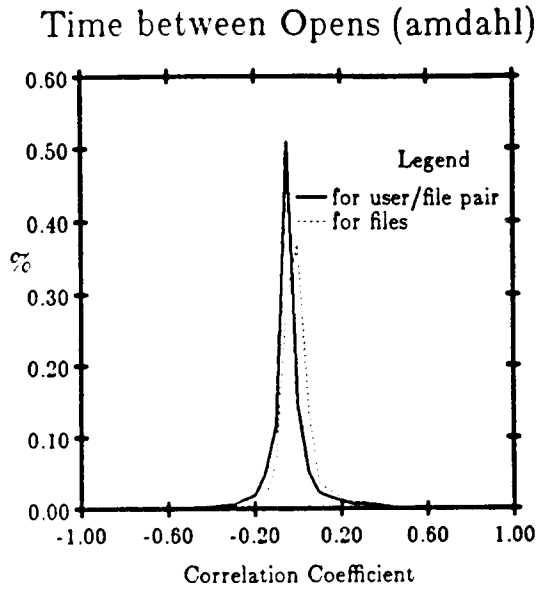


Figure 3.4b: The figure contains the distribution of the one step correlation coefficient for the time between opens. The solid line is for user/file pairs, while the dotted one is for files. All user/files or files with more than 5 opens are included in the the measurements, and the correlation coefficient are weighted with the number of opens for each user/file pair or file. The figure contains measurements from the trace from Amdahl.

step correlation coefficient has a negative bias, $E(\tilde{\rho}_1) = -\frac{1}{n-1}$.

The plots in Figure 3.4 confirm that a renewal process is a reasonable model of how users reference shared files. For the user/file pairs that represent the majority of the opens, the one step correlation coefficient is close to 0.0. This also supports the hypothesis of stationary processes, since processes with any kind of monotone trend would have either positive or negative coefficients.

Poisson hypothesis. The Poisson hypothesis is rejected for all installations; the coefficient of variation for the time between references is too high. In particular the acceptance percentage is low for the cases that represent a large fraction of the opens; among user/file pairs with more than 50 opens, the Poisson assumption is accepted for less than 8% of the cases.

3.6.1.1. Results for the Superpositioned Process for Each File

So far we have only analyzed the properties of each user's reference process to a file. The same analysis can be repeated on the total reference process to each file. This process is the superpositioned process from all users sharing the file. Even though the process for the individual users can be model by a renewal process, there is no reason to expected the superpositioned process to be a renewal model. Only in a few restricted cases will the superpositioning of several renewal processes result in a renewal process. Neither is it

intuitive that the reference process to a file should "start all over" each time it is opened. Most likely each user will have different reference patterns and the future of the process will depend on the current user.

The results of the analysis are shown in Table 3.3a, b, and c. Briefly summarized, the results are similar to the ones found for the user/file pairs; but the hypothesis is accepted by a lower percentage of the cases. Figures 3.4a and b plot the distribution of the one step correlation coefficients for the time between opens on a per file basis (dotted line). As for user/file pairs, the coefficients are weighted with the number of opens made.

Table 3.3a: Results for the statistical test for the file reference process at SLAC. The reference process for each file is tested to see if it is a stationary, a renewal, and a Poisson process. All tests use an α level of 5%.

number references observed	no of files tested	% of all opens to shared files	% acceptance for stationarity hypothesis	% acceptance of renewal hypothesis	% acceptance of Poisson hypothesis
5- 9	72	1.2	100.0		73.6
10-19	74	2.6	93.3		56.8
20-29	31	2.0	96.8	87.1	35.5
30-49	40	3.9	75.0	67.5	10.0
50-	128	89.7	78.1	64.1	10.9
total	345	99.4	87.3	68.3	35.9

Table 3.3b: Results for the statistical test for the file reference process at Hughes. The reference process for each file is tested to see if it is a stationary, a renewal, and a Poisson process. All tests use an α level of 5%.

number references observed	no of files tested	% of all opens to shared files	% acceptance for stationarity hypothesis	% acceptance of renewal hypothesis	% acceptance of Poisson hypothesis
5- 9	436	5.6	99.3		60.8
10-19	305	8.1	92.1		35.7
20-29	111	5.1	80.2	77.5	10.8
30-49	109	7.9	83.5	82.6	9.2
50-	129	54.5	81.1	87.4	42.6
total	1090	81.1	87.4	66.2	36.5

The stationarity hypothesis is rejected for from 13% to 30% of the files. The one step correlation coefficients are clustered around 0.0, and there does not seem to be any trends in the reference pattern to most of the files. We believe there is not enough evidence to reject the hypothesis of a stationary reference process to each file. Even though a renewal process is not a likely model for the total reference process to a file (see previous paragraphs), a renewal process seems to be reasonable model of how files are used in the traces we analyzed; most of the empirical correlation coefficients are clustered around 0.0, and the hypothesis is accepted by a large number of the cases for which it was tested. From the test results, it is obvious file references cannot be modeled as a Poisson process. Among files with more than 50 opens, representing from 54% to 92% of all opens, the

Table 3.3c: Results for the statistical test for the file reference process at Amdahl. The reference process for each file is tested to see if it is a stationary, a renewal, and a Poisson process. All tests use an α level of 5%.

number references observed	no of files tested	% of all opens to shared files	% acceptance for stationarity hypothesis	% acceptance of renewal hypothesis	% acceptance of Poisson hypothesis
5- 9	353	1.0	97.5		60.6
10-19	306	1.8	81.7		23.9
20-29	177	1.8	76.8	75.1	15.8
30-49	181	2.9	68.5	63.5	14.4
50-	766	91.8	53.0	47.9	6.4
total	1783	99.3	70.7	54.7	21.9

Poisson hypothesis is accepted in less than 9% of the cases.

3.8.2. Results for the Order Users Access Files.

We only tested whether the number of consecutive opens from a user to a particular file could be modeled by a geometric distribution. To simplify the discussion we use the term "a run" for a set of consecutive opens from a user to a particular file. As previously explained, we based all our tests on the number of consecutive opens - 1 (run size - 1), and used the geometric+1 distribution instead of the geometric one. An α level of 5% is used, and the results for each user/file pair are found in Table 3.4 a, b, and c.

Table 3.4a: Results for testing the hypothesis of a geometricly distributed number of consecutive opens by a user to a particular file at SLAC. Two sets of tests are used; the goodness of fit test, and a combination of three test described in the previous section. An α level of 5% is used. The row labeled "total" gives the overall results for the cases that were tested. These cases represented 76.6% of all opens.

number of runs observed	no of user/file pairs tested for goodness of fit	% of user/file pairs with geometric run size accepted by goodness of fit test	no of user/file pairs tested combined test	% of user/file pairs with geometric run size accepted by the combined test	mean coefficient of variation for the run size - 1
2- 9	196	98.0	353	92.6	2.17
10-30	248	90.7	248	79.4	4.64
30-	79	79.8	79	72.2	6.08
total	523	91.8	680	85.4	3.52
probability of consecutive opens					
0.00-0.50	305	99.7	316	93.0	5.96
0.50-0.75	138	84.1	201	78.6	1.60
0.75-1.00	80	75.0	163	79.1	1.17

The hypothesis was tested with 1) a goodness of fit test between the empirical run size

distribution and a geometric distribution, and 2) a combination of the goodness of fit test, the difference sign test, and the coefficient of variation test (all described in the previous section). If we assume that the result for each test is independent of the other tests and that each test uses an α level of 5%, the combined test has an α level of $\approx$ 15%. (1 - Prob(not rejected by any of the tests) = $1-(1-0.005)^3$). The results for the combined test is only included to complement the goodness of fit test which is used as the basis for the analysis. The coefficient of variation for run size - 1 is also reported.

From a strictly statistical viewpoint, the hypothesis of a geometric distribution for the number of consecutive opens for all user/file pairs, can at best be accepted for one trace (Hughes). For the two other traces, the percentage of cases for which the hypothesis is accepted is too low at an α level of 5%. This only implies that not all sets of consecutive opens have a geometric distribution. To model them by a geometric distribution is still appropriate; the hypothesis is accepted by a large percentage of the cases that are tested (from 88% to 93%). Among user/file pairs with more than 30 sets of consecutive opens, which represent a majority of all opens to shared files (see table 3.1), the hypothesis of a geometric distribution is accepted by at least 73% of the cases.

Table 3.4b: Results for testing the hypothesis of a geometricly distributed number of consecutive opens by a user to a particular file at Hughes. Two sets of tests are used; the goodness of fit test, and a combination of three test described in the previous section. An α level of 5% is used. The row labeled "total" gives the overall results for the cases that were tested. These cases represented 43.7% of all opens.

number of runs observed	no of user/file pairs tested for goodness of fit	% of user/file pairs with geometric run size accepted by goodness of fit test	no of user/file pairs tested combined test	% of user/file pairs with geometric run size accepted by the combined test	mean coefficient of variation for the run size - 1
2- 9	167	94.0	406	93.6	1.74
10-30	168	92.3	168	84.5	3.70
30-	28	85.7	28	71.4	6.12
total	363	92.6	602	90.0	2.49
probability of consecutive opens					
0.00-0.50	223	97.8	234	93.6	4.38
0.50-0.75	88	88.6	164	88.4	1.50
0.75-1.00	52	76.9	204	87.3	1.13

The test results are however, to some degree deceptive. For many of the cases, the goodness of fit test is not powerful, and it cannot distinguish between different distributions. In the second part of table 3.4, the test results are displayed based on the empirical parameter of the geometric model (p), where p is the probability the the next open will come from the same user as the current one. One group consists of all user/file pairs with an average of up to 2 consecutive opens ($0<p<0.5$), the second group consists of all user/file pairs with an average between 2 and 4 consecutive opens ($0.5<p<0.75$), while the third group consists of all user/file pairs with an average larger than 4 ($0.75<p<1$). For small parameter values, in the 0 to 0.5 range, most of the runs will consist of one or two opens. In the corresponding geometric model at least 75% of the runs should have

Table 3.4c: Results for testing the hypothesis of a geometricly distributed number of consecutive opens by a user to a particular file at Amdahl. Two sets of tests are used; the goodness of fit test, and a combination of three test described in the previous section. An α level of 5% is used. The row labeled "total" gives the overall results for the cases that were tested. These cases represented 59.3% of all opens.

number of runs observed	no of user/file pairs tested for goodness of fit	% of user/file pairs with geometric run size accepted by goodness of fit test	no of user/file pairs tested combined test	% of user/file pairs with geometric run size accepted by the combined test	mean coefficient of variation for the run size - 1
2- 9	379	96.3	1159	88.5	1.51
10-30	493	86.4	493	74.4	3.20
30-	189	73.0	189	60.9	3.71
total	1061	87.6	1841	81.9	2.19
probability of consecutive opens					
0.00-0.50	416	97.1	442	87.8	4.29
0.50-0.75	321	85.1	463	78.8	1.74
0.75-1.00	324	77.8	936	80.7	1.42

one or two opens. Since so much of the distribution mass is concentrated on one or two values, the goodness of fit test cannot distinguish between different distributions, unless there are many observations. This is reflected in the high acceptance for the cases with a small parameter value ($p < 0.5$).

For user/file pairs with parameter in the 0.75 to 1.0 range, the geometric distribution has its mass spread more evenly between the different values, and the goodness of fit test can better determine whether the empirical distribution fits a geometric one. The hypothesis of a geometric distribution is accepted by $\approx$ 76% of the user/file pairs with parameter in the 0.75 to 1 range. Regardless, from a strictly statistical viewpoint, the hypothesis should be rejected.

From a modeling viewpoint, a geometric distribution of the number of consecutive opens is an appropriate model; it is accepted by $\approx$ 76% of all user/file pairs with an average of more than 4 consecutive opens, and the acceptance is even higher for the remaining cases. In addition, the coefficient of variation for the number of consecutive opens corresponds well to the one for a geometric distribution. In table 3.4 we measured the coefficient of variation for the number of opens per run - 1. The corresponding model is the geometric +1 distribution with a coefficient of variation equal to $= \frac{1}{p}$, where p is the geometric parameter (see section 3.5). The coefficient of variation should therefore be between 2 and 1.33 if $0.5 < p < 0.75$, and smaller than 1.33 for $p > 0.75$. These numbers correspond nicely with the empirical coefficients given in Table 3.4.

To summarize, we believe it is appropriate to model the number of consecutive opens from the same user with a geometric distribution. In a strict statistical sense the model should be rejected. However, a geometric model is accepted for most of the user/file pairs (>73%) , and the observed coefficient of variation are similar to the ones predicted by a geometric model.

3.8.2.1. Results for the Superpositioned Process

In the previous section, we made a distinction between the different users sharing a file. The distribution for the number of consecutive opens (a run) was tested individually for each user. In this section, the identity of the users is neglected, and the distribution of the number of consecutive opens by the same user for the file as a whole is tested. This is equivalent to testing whether all users sharing the file have the same geometric distribution. The procedure of the previous section is repeated, but instead of testing each user/file pair, only the superpositioned process for each file is analyzed. The results are given in Table 3.5a, b, and c. The results on a per file basis are in many aspects similar to the results on a per user/file pair basis. The overall acceptance percentage is nearly the same (from 77% to 86%). However, we conclude that on a per file basis a geometric model of the run size is not an appropriate model, since several other properties of this model are not observed in the traces.

Table 3.5a: Results for testing the hypothesis of a geometricly distributed number of consecutive opens by the same user to a particular file at SLAC. Two sets of tests are used; the goodness of fit test, and a combination of three test described in the previous section. An α level of 5% is used. The row labeled "total" gives the overall results for the cases that were tested. These cases represented 72.5% of all opens.

number runs observed	no of files tested for goodness of fit	% files with geometric run size accepted by goodness of fit test	no of files tested by combined test	% of files with geometric run size accepted by the combined test	mean coefficient of variation for the run size - 1
2- 9	30	96.7	77	80.5	1.84
10-30	62	83.9	62	72.6	2.43
30-	58	60.4	58	50.0	4.03
total	150	77.3	197	69.0	2.67
probability of consecutive opens					
0.00-0.50	40	82.5	40	75.0	4.40
0.50-0.75	71	85.9	75	73.3	2.49
0.75-1.00	39	56.4	82	62.2	1.99

The model is accepted by less than 66% of all files with more than 30 sets of consecutive opens. These are the files representing the majority of the opens to shared files (see table 3.1). Further the empirical coefficients of variation for the run size are too high to be compatible with a geometric model. As previously described, files with a geometric parameter in the 0.5 to 0.75 range should have a coefficient between 2 and 1.33, while the files with parameter larger than 0.75 should have a coefficient smaller than 1.33. None of the traces have empirical coefficients within these limits; they are too high.

Even though for each user it is appropriate to model the number of consecutive opens by a geometric distribution, it is not an appropriate model on a per file basis. On a per file basis, the run size is distributed according to a mixture of geometric distributions (one for each user sharing a file) with different means. It is a mixture of geometric distributions, since the users sharing a file typically will have a different average number of opens per run. This is also a plausible explanation of user behavior. Typically there will be one user that opens a file often, while other users open the file less frequently. The

Table 3.5b: Results for testing the hypothesis of a geometricly distributed number of consecutive opens by the same user to a particular file at Hughes. Two sets of tests are used; the goodness of fit test, and a combination of three test described in the previous section. An α level of 5% is used. The row labeled "total" gives the overall results for the cases that were tested. These cases represented 59.9% of all opens.

number runs observed	no of files tested for goodness of fit	% files with geometric run size accepted by goodness of fit test	no of files tested by combined test	% of files with geometric run size accepted by the combined test	mean coefficient of variation for the run size - 1
2- 9	75	100.0	193	84.5	1.87
10-30	81	88.9	81	75.3	3.19
30-	55	65.5	55	63.6	5.14
total	211	86.4	329	78.7	2.74
probability of consecutive opens					
0.00-0.50	77	93.5	77	90.9	5.16
0.50-0.75	86	88.4	88	80.7	2.17
0.75-1.00	48	72.9	164	72.0	1.91

Table 3.5c: Results for testing the hypothesis of a geometricly distributed number of consecutive opens by the same user to a particular file at Amdahl. Two sets of tests are used; the goodness of fit test, and a combination of three test described in the previous section. An α level of 5% is used. The row labeled "total" gives the overall results for the cases that were tested. These cases represented 64.8% of all opens.

number runs observed	no of files tested for goodness of fit	% files with geometric run size accepted by goodness of fit test	no of files tested by combined test	% of files with geometric run size accepted by the combined test	mean coefficient of variation for the run size - 1
2- 9	188	97.9	636	63.7	1.92
10-30	232	81.9	232	61.2	2.93
30-	157	49.1	157	42.0	5.45
total	577	78.2	1025	59.8	2.69
probability of consecutive opens					
0.00-0.50	67	92.5	67	86.6	4.05
0.50-0.75	156	86.5	163	72.4	2.96
0.75-1.00	354	71.8	795	55.0	2.52

users will not have the same probability of opening (p) the file, and their average run size will differ. This aspect will be further analyzed in the next section.

3.7. Discussion of Possible Models

The previous analysis showed that an appropriate model of the file reference process should have the following properties:

1 It should be a stationary, renewal process both on a per user/file pair basis and on a per file basis.

2 The coefficient of variation for the time between references should be high (>1).

3 The number of consecutive opens from a user to a particular file can have a geometric distribution.

4 Users sharing a file will have different average number of consecutive opens. On a per file basis, a geometric distribution for the run size is not an appropriate model. The individual users sharing a file have different average run sizes.

We limit our search for models with these properties to the ones that are simple and have a reasonable model of user behavior. Specifically, we will investigate four models, a stack model with constant reference probability adapted from a model of locality in program references [Spirn77], a Markov chain model developed to explain locality in data base references [Easton75], a modulated Poisson model which models different levels of user interest in a file [Segal76], and a batch Poisson model suggested by the observed reference patterns. The two first ones only model the order users access files, while the two last ones also model the time between opens. All of these models provide reasonable explanations of user behavior, and they are mathematically fairly simple, but only the batch Poisson model with a geometric batch size has all the required properties.

3.7.1. Stack Model

A stack model is a simple Markovian model of how users reference a particular file. It is derived from the LRU stack model of paging [Spirn77], and it only describes the order users access a particular file and neglects the time between opens. The model assumes the probability user i will open the file next is solely determined by the number of users that have accessed the file since the last time user i opened it.

3.7.1.1. Description

The model keeps the users referencing a file ordered on a stack with the most recent one at the top. The height of the stack is smaller or equal to the number of users accessing the file. The probability the user in stack position j will open the file next is p_j, where p_j is constant and independent of the identity of the user. The probability of an open from a user not on the stack is $\bar{P}_n = 1 - \sum_{i=1}^{n} p_i$. Usually, p_i is decreasing with increasing i, and with $p_1 > p_j$ for all j, the model reproduces the locality observed in the traces; the current user is the most likely one to continue referencing the file. Table 3.6 displays the p_i for a stack model averaged over all opens in the trace. Only the first 7 stack positions are modeled; p_8 is the probability the open will come from a user not on the 7 element stack. As we see from the empirical stack parameters, the current user is the most likely to continue to open the file, with p_1 in the 0.60 to 0.86 range. The probability a user will open the file next decreases with increasing stack position. However, there is a high probability that a user not on the stack will open the file. About half of this probability is due to opens from users that have not accessed the file so far (0.042 for SLAC, 0.145 for Hughes, and 0.024 for Amdahl.)

3.7.1.2. Implications

The stack model provides a satisfactory explanation of locality, since the probability the current user will continue to access the file is a constant p_1. The number of consecutive opens from a user to a file has a geometric distribution with mean $\frac{1}{1-p_1}$. This corresponds well to the property observed in the previous section.

Table 3.6: The table contains the empirical stack probabilities averaged over all opens in the trace. The empirical probabilities are measured for the the 7 first stack positions. $p_{>7}$ is the probability an open will come from a user not on the stack.

Stack position	Slac	Hughes	Amdahl
1	0.6775	0.6040	0.8588
2	0.0936	0.0651	0.0514
3	0.0424	0.0199	0.0176
4	0.0245	0.0105	0.0088
5	0.0175	0.0074	0.0055
6	0.0108	0.0050	0.0036
7	0.0095	0.0038	0.0035
>7	0.1242	0.2793	0.0517

Table 3.7: The table contains the average ranking of the number of opens per user. The users are labeled in the order they access a file, and the user with the most opens to a file is assigned rank 1. The rank is then averaged over all files. The three columns to the right contain the expected ranking with a stack model.

	Actual Ranking			Expected Ranking		
Order users access a file	Slac	Hughes	Amdahl	Slac	Hughes	Amdahl
1	2.016	1.223	1.479	2.892	1.582	2.146
2	2.572	1.988	2.099	2.942	1.783	2.226
3	4.123	3.120	3.333	4.577	3.244	3.313
4	5.269	4.778	4.304	5.865	6.106	4.408
5	5.457	5.734	5.641	7.154	7.870	5.662
6	6.709	8.323	6.448	7.943	9.838	6.879
7	8.358	8.815	6.732	8.739	11.247	7.826

An important disadvantage of the stack model is the lack of distinction between users. In a stack model, all users have the same number of expected opens to a file. Let π_i be the stationary probability a particular user will be in stack position i. π_i is then given by equating the rates in and out of the state:

$$\sum_{j=i}^{n} p_j \pi_{i-1} = \sum_{j=i+1}^{n} p_j \pi_i + p_i \pi_i \qquad 3.8$$

The solution is $\pi_{i-1} = \pi_i = \frac{1}{n}$ where n is the number of users sharing the file. Regardless of a user's current stack position, over a longer period they will spend the same amount of time in each stack position (measured in terms of all opens to the file), and

they will have the same number of expected opens to the file.

This contradicts the reference pattern we observed; most opens will typically come from the first user to reference the file. This is further illustrated by the measurements in Table 3.7. For each file, we ranked users according to the number of times they opened the file. The user with rank 1 opened the file most often, followed by the user with rank 2 and so on. The users were labeled according to the order they accessed the file. Table 3.7 contains the average ranking of users with the same label, and there is a definite ordering; the ranking is decreasing with the order users reference a file. The user with the most opens will typically be the first user to open the file. The numbers in Table 3.7 clearly contradict the claim that users have the same expected number of opens per file. If this was true, all users would have the same probability of being the one with the most opens. The probability the first user to reference the file would be ranked 1 is then $\frac{1}{n}$, where n is the number of users sharing the file. The expected ranking of the first user would be $\sum_{i=1}^{n} \frac{i}{n} = \frac{n+1}{2}$, and, average over all files, the expected ranking of the first user to open a file is:

$$\sum_{k=1}^{\text{max number of users}} f_k \cdot \frac{k+1}{2}$$

where f_k is the fraction of files accessed by k users. The sum $\sum_{k=1}^{\text{max users}} f_k k$ is the average number of users sharing a file given in Table 2.2. The expected rank for the j'th user to access a file can be calculated in a similar way by using the fraction of files that are shared by k users among the files that are shared by at least j users, $\left[\frac{f_k}{1 - \sum_{i=1}^{j-1} f_i}\right]$. The expected rank for the j'th user is then:

$$\sum_{k=j}^{\infty} \left[\frac{f_k \cdot \frac{k+1}{2}}{1 - \sum_{l=1}^{j-1} f_l} \right]$$

The expected and the measured ranking are given in Table 3.7. If the stack model is correct, the expected ranking of the first user to reference the file should be 2.89 (SLAC), 1.58 (Hughes) and 2.145 (Amdahl). These numbers are substantially higher than the actual average ranking; typically, the first user will open the file more often than other users. In addition, the expected ranking is higher than the measured ranking (see Table 3.7), which implies that the number of opens per user tends to decrease with the order users access the file.

The disadvantage of the stack model can also be illustrated by the results from the previous section. Since the stack model does not distinguish between different users, all users sharing a file should have the same mean number of consecutive opens. In section 3.6.1.1 we concluded this was not an appropriate model; users sharing a file will not have the same mean number of consecutive opens. File placement algorithms should therefore not assume that users sharing a file will have the same reference behavior. Even though the stack model does not distinguish between the different users, placement algorithms based on this a model may still have a reasonable performance. Such algorithms are therefore investigated further in the next chapter.

To summarize, a stack model provides a reasonable explanation of the observed locality among references to files. It does not distinguish between the various users sharing a file, and it assumes they all have the same properties. We therefore believe it is not an appropriate model, but in the next chapter file placement strategies based on a stack model are investigated.

3.7.2. Markov Chain Model

As already mentioned, the stack model is a Markov chain model, where the states of the chain are the stack positions of the users. The common denominator for Markov chain models is that the future reference pattern to a file depends only on the current state. In the models discussed in this section, each user corresponds to a separate state.

Table 3.8a: The Table contains the transition probabilities averaged over all opens for the trace from SLAC. The column labeled π contains the stationary probability distribution calculated from the observed transition probabilities.

	Transition Probability between states								π
State	1	2	3	4	5	6	7	8	
1	0.8701	0.0625	0.0172	0.0106	0.0088	0.0052	0.0052	0.0204	0.268
2	0.0911	0.7728	0.0517	0.0249	0.0129	0.0067	0.0063	0.0337	0.145
3	0.0635	0.0586	0.6517	0.0730	0.0325	0.0219	0.0275	0.0713	0.078
4	0.0582	0.0512	0.0656	0.6865	0.0375	0.0140	0.0153	0.0718	0.066
5	0.0438	0.0253	0.0357	0.0257	0.7317	0.0293	0.0217	0.0867	0.067
6	0.0369	0.0246	0.0442	0.0217	0.0326	0.6959	0.0326	0.1115	0.037
7	0.0293	0.0182	0.0298	0.0173	0.0227	0.0120	0.7888	0.0818	0.061
8	0.0240	0.0165	0.0182	0.0173	0.0219	0.0130	0.0168	0.8722	0.278

3.7.2.1. Description

Each file is modeled by a separate Markov chain. All users sharing a particular file are represented as a separate state in the chain. Each open is a transition between the states, so an open from user i following an open from user j is a transition from state j to state i. The transition probability p_{ji} is the probability an open from user i (the i'th user to open the file) will follow an open from user j. The expected fraction of opens from a particular user k is given by the stationary probability $\pi_k = \sum_{i=1}^{n} \pi_i p_{ik}$, provided the chain is nonabsorbing, i.e., that the file will continued to be shared as time progresses. Table 3.8 contains the average transition probabilities over all opens for the three traces. The users are labeled according to the order they accessed the file. The last state, 8, contains all users with index 8 or higher. As we see, there is a strong degree of locality; the same user is likely to continue referencing the file. Most likely there is a bias towards transitions between state 1 and 2, since files are not shared by the same number of users. Transitions between the two first states will be more frequent, and this can bias the transition probabilities.

Some of the files display a sequential access pattern. Under these conditions, with n users referencing a file, user n is likely to be the last one to open the file. Sometimes no other user will reference the file after user n has started using it, and there will be no transitions away from node n. In these cases, the n'th user will appear as an absorbing state in a Markov chain model. The transition probabilities in Table 3.8 are averaged over all opens, so it is unlikely that any of the states will be absorbing ones; most likely

Table 3.8b: The Table contains the transition probabilities averaged over all opens for the trace from Hughes. The column labeled π contains the stationary probability distribution calculated from the observed transition probabilities.

	Transition Probability between states								π
State	1	2	3	4	5	6	7	8	
1	0.7622	0.2091	0.0113	0.0042	0.0021	0.0021	0.0007	0.0084	0.238
2	0.1234	0.7418	0.0887	0.0114	0.0068	0.0029	0.0019	0.0223	0.266
3	0.0756	0.0573	0.7210	0.0611	0.0136	0.0106	0.0074	0.0534	0.139
4	0.0556	0.0478	0.0508	0.6663	0.0778	0.0209	0.0102	0.0706	0.059
5	0.0427	0.0519	0.0448	0.0610	0.5025	0.0743	0.0193	0.2035	0.034
6	0.0797	0.0550	0.0664	0.0626	0.0323	0.3586	0.1309	0.2144	0.017
7	0.0414	0.0186	0.0331	0.0290	0.0538	0.0393	0.3747	0.4099	0.016
8	0.0281	0.0227	0.0268	0.0162	0.0281	0.0160	0.0200	0.8422	0.232

Table 3.8c: The Table contains the transition probabilities averaged over all opens for the trace from Amdahl. The column labeled π contains the stationary probability distribution calculated from the observed transition probabilities.

	Transition Probability between states								π
State	1	2	3	4	5	6	7	8	
1	0.9289	0.0415	0.0108	0.0061	0.0056	0.0016	0.0014	0.0041	0.421
2	0.0812	0.8552	0.0271	0.0098	0.0085	0.0042	0.0022	0.0118	0.184
3	0.0530	0.0273	0.8519	0.0223	0.0140	0.0053	0.0054	0.0208	0.106
4	0.0451	0.0284	0.0232	0.8499	0.0210	0.0076	0.0069	0.0178	0.064
5	0.0445	0.0221	0.0244	0.0154	0.8469	0.0136	0.0079	0.0251	0.060
6	0.0310	0.0259	0.0245	0.0152	0.0223	0.7901	0.0227	0.0682	0.027
7	0.0223	0.0164	0.0211	0.0147	0.0126	0.0142	0.8390	0.0597	0.028
8	0.0210	0.0182	0.0186	0.0105	0.0130	0.0169	0.0126	0.8893	0.110

there will at least be one file that has a transition out of state that is absorbing for one of the other file. The stationary transition probabilities will, however, be biased towards the higher states, and the stationary probability of the chain will be more skewed towards the higher states compared to the empirical distribution.

3.7.2.2. Implications

The Markov chain model is a versatile model, but with n users sharing a file it requires $n(n-1)$ parameters. From an application view point this is a clear disadvantage. However, there are several simple Markov chain models requiring only $n+1$ estimates that are also plausible models of user behavior. Among these are a model proposed by Easton [Easton75]. He modeled references to a data base by a Markov chain model where each page in the data base was a state. The probability a transaction would continue to reference the same page was a constant r. If the transaction stopped using page j, a new transaction would start using the page with probability λ_j. The transition probabilities are then:

$$p_{ii} = r + (1-r)\lambda_i$$
$$p_{ij} = (1-r)\lambda_j.$$

where λ_i is the probability a transaction will start referencing page i. The model can

directly be translated into a model for opens to files, where each state in the model is a user and a transition corresponds to an open. In this model, the users are viewed as issuing batches of opens to a particular file, with no overlapping of batches from different users. Each batch of opens has a geometric distribution with mean $\frac{1}{1-r}$, and the probability a batch will come from user i is λ_i. The stationary probability distribution for this model is $\pi_i = \lambda_i$, i.e., the expected number of opens to a file from a particular user will depend on the identity of the user. This model will have a geometric distribution of the number of consecutive opens from a user/file pair (requirement 3), and the users will not have the same average run size (requirement 4).

The only disadvantage of the model is the stipulation that all users will have the same batch size. If we allow the batches to have different mean sizes for different users, the model can be reformulated to:

$$\begin{aligned} p_{ii} &= k_i + (1-k_i)\lambda_i \\ p_{ij} &= (1-k_i)\lambda_j. \end{aligned}$$

A special version of this model has $\lambda_i = \lambda_j = \frac{1}{n}$. In this formulation the users have different mean batch size $\frac{1}{1-k_i}$ and uniform transition probabilities.

Under this model the stationary probability π is:

$$\pi_i = \frac{\frac{1}{1-k_i}}{\sum_{j=1}^{n} \frac{1}{1-k_j}} \qquad 3.9$$

All of these Markov chain models have the desired properties; on a per user/file basis, the number of consecutive opens will have a geometric distribution, while on a per file basis the distribution is not a geometric one.

3.7.3. Modulated Poisson Model

The two previous models dealt only with the order users accessed files. The two remaining models also incorporate the time between opens. One of these models, the modulated Poisson model [Segal76] assumes a user's interest in referencing a file is not a constant but changes over time.

3.7.3.1. Description

In a modulated Poisson process, a user will reference the file according to a Poisson process with time variant rates. Segal [Segal76] proposed a model where the rates changed according to an underlying Markov chain. Each user/file pair is represented by a Markov chain, where the states represent specific levels of interest in the file, like active and moderate. In each state, the user references the file according to a Poisson process with a state dependent intensity λ_i. The state of the underlying Markov chain can only change at the reference epochs. With probability q_{ij} the user will move from state i to state j whenever a reference is made in state i. All users act independently, and we assume there is no correlation between references to different files. The references process to a particular file is therefore the superposition of several independent modulated Poisson processes. To state the model formally we need the following definitions:

i reference level for a user, and there are m of these levels

q_{ij} probability the interest level will change from level i to j
λ_i reference rate when user is in interest level i

3.7.3.2. Implications

The modulated Poisson model creates a much more complex model of how users reference files. The order they use it can no longer be expressed by a simple Markov chain where each user is a separate state. Instead, the state space must be expended to $n \cdot m^n$ vectors with n+1 elements: the first element in the vector is the identity of the user referencing the file, while the remaining n elements are the current state of the underlying Markov chain for each of the users.

A modulated Poisson process is not a renewal process, except in some special cases where the model degrades into a hyper-exponential file reference model; if there are only two levels of interest (active and passive), and $q_{12} = q_{21} = 0.5$, the time between opens from the same user has a hyper-exponential distribution. However, it is easy to formulate modulated Poisson models which will have a low one step correlation coefficient ($|\rho_1| \leq 0.1$) and a high coefficient of correlation for the time between opens. With the tests we have used, this reference pattern could not be separated from a stationary renewal process. The model is therefore compatible with our requirement of a stationary, renewal process with a high coefficient of variation (requirement 1 and 2).

To simplify the discussion, let us assume the extreme case where each user only has two levels of interest (a and p). Further let $p_a = q_{aa}$ be the probability of remaining in the active state and p_p the one of remaining in the passive state. The fraction of opens made from active state (a) is $\pi_a = \frac{1-p_p}{2-p_a-p_p}$. If the two reference rates λ_a and λ_p are such that $r = \frac{\lambda_a}{\lambda^p} >> \frac{\pi_a}{1-\pi_a}$, the coefficient of variation CV is: $C \approx \frac{2}{1-\pi_a} - 1$ and the one step correlation coefficient is : $\rho_1 \approx 1 - \frac{2-p_p}{1+\pi_a}$. Based on these formulas, we can formulate models that fulfill the requirements of a low correlation coefficient and a high coefficient of variation. As an example, a two state modulated Poisson process with $p_p = 0.56$ and $p_a = 0.707$ implies $\pi_a = 0.606$. If $r = \frac{\lambda_a}{\lambda_p} = 10.0$, the one step correlation coefficient $\rho_1 = 0.0863$ and the coefficient of variation (CV) = 2.83.

Such a reference pattern corresponds well to the one we found in the traces. The modulated Poisson model we outlined also provides a plausible model of user behavior, with bursts of file references with short time between the references, followed by long intervals with only sporadic references.

In a modulated Poisson process, the number of consecutive opens from a user to a file does not have a geometric distribution. The probability the next open will come from the same user depends on the state the user is in. This probability is higher if a user is in an active state than in a passive state. Consecutive opens can be issued with the user in different states, and the distribution for the number of consecutive opens will be complex. The distribution will be highly skewed, with a few long runs due to references from the active state and more numerous short runs due to references from the passive state. At best, it can be approximated by a mixture of geometric distributions. Only in the extreme case where all references are made from the active case ($p_p \approx 0$), will the run size have a geometric distribution, but the model then closely resembles a batch Poisson model with

geometric batch sizes.

In addition, the probability the next reference will come from the same user as the current is not constant; it increases with the number of consecutive opens observed. As the length of the sequence increases, so does the probability all other users are in a passive state. The probability the next reference will come from the same user as the current one will increase with the number of consecutive opens.

3.7.4. Batch Poisson Model

The batch Poisson model is an extension of Easton's Markov chain model. In this model, the time between transitions to different states has an exponential distribution. A batch Poisson model emphasizes the bursty nature of the file reference process, and it assumes users issue batches of opens to a particular file.

3.7.4.1. Description

In the model each user accesses a particular file in independent spurts (batches) of activity. To comply with the results of the analysis, we assume the number of opens in a batch has a geometric distribution. The time between the batches is modeled by an exponential distribution. However, the time between opens in the same batch can have any distribution, but the time between opens must be such that the duration of a batch is substantially smaller than the time between batches; the probability the batches will overlap can then be disregarded. For convenience we assume the time between opens in a batch has an exponential distribution. Similar to the previous model, all users act independently, and we assume references to different files are uncorrelated.

3.7.4.2. Implications

The batch Poisson model with geometric distributed batch sizes has all the required properties. For each user/file pair the reference process is stationary. The one step correlation coefficient for the time between opens (for the same user/file pair) is equal to 0.0, since the process regenerates at each open. The time between opens from the same user will have a hyper-exponential distribution, and the coefficient of variation will therefore be high. If mean batch size for node i is $\frac{1}{1-k_i}$, the mean time between batches is $\frac{1}{\lambda_b}$, and the mean time between opens in a batch is $\frac{1}{\lambda_o}$, the coefficient of variation (CV) is :

$$CV = \frac{k_i \frac{2}{\lambda_o^2} + (1-k_i)\frac{2}{\lambda_b^2}}{\left[k_i \frac{1}{\lambda_o} + (1-k_i)\frac{1}{\lambda_b}\right]^2} - 1$$

As an example, in a model with mean batch size = 2 ($k_i = 0.5$) and with $\frac{\lambda_o}{\lambda_b} = 10.0$, the coefficient of variation for the time between opens is 2.3. Such a reference pattern also fits the one observed in chapter 2; the distribution of the time between opens from the same user was skewed towards shorter intervals compared to the distribution for the time between opens from different users.

If n users access the file independently according to a Batch Poisson model, the order they accessed the file can be modeled by a Markov chain.. The transition between two states, *i and j*, corresponds to an open from user *j* following an open from user *i*. If we make the additional approximation that no batches will overlap, the transition

probabilities p_{ij} are :

$$p_{ij} = (1-k_i)\frac{\lambda_j}{\sum_{i=1}^{n}\lambda_i} \qquad 3.12$$

where λ_i is the reference rate for batches from user i and $\frac{1}{1-k_i}$ is the mean batch size. The batch Poisson process therefore leads to models of the order users will reference files which are close to the one suggested by Easton [Easton75]. On a per file basis, the batch size will have a distribution that is a mixture of the geometric distributions for each individual user's batch size. This satisfies requirement 4. Unfortunately, the time between opens on a per file basis will not be a renewal process. Each open is not a renewal point anymore, since the stochastic properties of the next open will depend on the identity of the current user. However, the end of each batch is a regeneration point, since the stochastic properties for the next open is the same at the end of each batch.

The batch Poisson model is not only a reasonable abstraction of the file reference process, but it is also a plausible description of user behavior. Users often adhere to the pattern of "open a file and do something to the file", "observe the results", "think", and if not "successful" repeat the cycle. Examples of such behavior are debugging, incremental program design, and queries to data bases. The cycle time is typically shorter than the time between each startup of a cycle. With a constant probability for success, but one that may vary with users and tasks, the number of consecutive opens from the same user to a file will have a geometric distribution.

3.8. Conclusion

In this chapter we have analyzed the stochastic properties of the file reference process to shared files. The best file placement strategy is to occasionally relocate some of the files, and stochastic models are needed to determine when and where files should be relocated. In the analysis we discovered that:

a) There is a clear ordering in the number of opens per user. The user who first reference the file will most likely open the file more often than other users.

b) The number of bytes transferred per open can be modeled by a constant for each user/file pair.

c) A reasonable model for the time between opens is a stationary renewal process with a high coefficient of variation. Such a model is appropriate both on a per user/file pair basis and on a per file basis.

d) The number of consecutive opens from the same user to a particular file can be modeled by a geometric distribution. Users sharing a file will, however, have a different mean number of consecutive opens.

Four simple but plausible models of how users reference files were evaluated to see how well they fitted the properties of the file reference process. Specifically, we investigated , a stack model with constant reference probability adapted from a model of locality in program references [Spirn77], a Markov chain model developed to explain locality in data base references [Easton75], a modulated Poisson model which models different levels of user interest in a file [Segal76], and a batch Poisson model suggested by the observed reference patterns. The first two only model the order users reference files, while the last two also model the time between opens. Only the Markov chain model and the batch Poisson model have all the required properties, and only the batch Poisson model deals

with all aspects of the file reference process.

In the batch Poisson model each user issues batches of opens to a particular file. In the model we consider, the number of opens in a batch has a geometric distribution, and the time between batches has an exponential distribution. The time between opens from the same batch can have any distribution, but for convenience we modeled it with an exponential distribution. However, the expected duration of a batch of opens (the time between opens multiplied by the expected number of opens in the batch) should be substantially smaller than the time between batches, so the coefficient of variation will be large. The model ignores that batches from different users will overlap, but under the conditions we have outlined the probability for such an event is small. The reference process for each user/file pair is independent of the process for all other user/file pairs. The superpositioned process for each file will also be a batch Poisson process where the distribution of the number of opens in a batch is a mixture of geometric distributions. The order users access a file is a Markov chain with transition probabilities that, except for a common normalizing factor, do not depend on the current state. Such a model was also used by Easton to describe references to a data base [Easton75].

The batch Poisson model is also a plausible description of user behavior. Users often adhere to the pattern of "open a file and do something to the file", "observe the results", "think", and if not "successful" repeat the cycle. We assume that the probability for success is constant, and that the cycle time is substantially shorter than the time between the startup of such tasks. The ensuing reference pattern is then a batch Poisson process with geometric batch sizes.

We believe a batch Poisson process with geometric batch size is a reasonable abstraction of a user's file references process. It is a simple model, which reproduces all the measured properties of the file reference process. It is also a plausible model of how users behave.

Chapter 4
File Migration in Distributed File Systems without Replication

Summary

This chapter analyzes file placement in distributed file systems that do not allow replication of files. We find that migration reduces the network traffic with up to 36% compared to static placement. Among the algorithms we analyze, those based on a batch Poisson model with geometric batch size have the best performance. These algorithms use the number of references, the number of changes in locality, and the fraction of a file accessed per open as the decision variables. The performance of these algorithms is insensitive to the estimation methods we use. The batch Poisson algorithms are decentralized, and they can be implemented so they only collect reference history for the current user. This may, however, increase the network traffic, but the algorithms become more robust to sudden changes in the reference pattern.

The last section evaluates the performance of the batch Poisson algorithms by analyzing the differences between the algorithms and the optimal look-ahead algorithm. The proposed algorithms have a reasonable performance with little room for improvement.

4.1. Introduction

In this chapter we compare the performance of file placement/migration algorithms for file systems that do not allow replication of the files. The system we investigate consists of several independent processors connected by a local area network. Each processor has its own local disks. The system has a common file system with the individual files stored at individual processors' own disks. The processors will have different access paths to the disks, since each disk is local to a particular processor, while the other processors can only access them through the network. Where a file is stored therefore affects the performance of the system. This chapter compares different placement/migration schemes using trace driven simulation. We find that the network traffic can be reduced if files are relocated (migrated) as the reference patterns change. For the optimal unrealizable algorithms, migration reduces the network traffic with 18% to 48% of that of the best static placement.

The processors (or nodes) in the system are connected by a high bandwidth, low latency network, and each processor can access all files regardless of where they are stored. Associated with each file is a *file host*, which is the processor controlling the disk where the file is stored. Similarly, associated with each job is a *job host*, which is the processor the job is running on. When a file reference has the same file and job host, the reference is *local*, otherwise it is *remote*.

There is no conceptual difference between remote and a local references. For a remote reference, the network and the file host operate as an extension of the job host's I/O channels. System-wise, remote references are costly in terms of wasted system resources. A remote reference implies opening and managing a communication path with a substantial overhead both for the job host and the file host. This was discussed in more detail in Chapter 1.

Remote references can be avoided by either moving the file to the job host (file migration), or by moving the job to the file host (process migration). Generally, the cost

of a single remote reference is less than the cost of either a file or process migration. Whether it is preferable to migrate or continue with remote references will depend on future references to that particular file.

In this chapter we concentrate on algorithms for file migration, and limit the scope of this chapter to single-copy file systems without replication. In addition, each user is permanently assigned to a processor, and all jobs originating from a user will run on the same processor.

The algorithms we analyze can be classified into five groups. Two of them represent the extreme points, static placement and demand migration. With demand migration, a file is moved to all nodes that reference it. The three remaining groups are algorithms that migrate a file only to some of the locations that use them. The first group is composed of algorithms based on a stack model of how users reference files. The second group consists of algorithms based on a Batch Poisson model. Both models were discussed in the previous chapter, and the batch Poisson model was recommended as the most appropriate one. The final group contains algorithms that try to estimate the best static location. Since this estimate will change, the files can be moved around.

The different algorithms are analyzed by trace driven simulation, with traces of file reference activity ranging in length from 8 to 30 days. We measure an algorithm's performance in the network traffic (in bytes) created by remote file references and file transfers. Preceding the simulation, we also recorded the reference pattern to all files and devised the file migration/placement scheme that in retrospect would have resulted in minimum network traffic. In addition to bounding the performance of the different algorithms, we also obtained a measure of the advantage of migration, which is not distorted by ineffective implementation.

This chapter has six sections. In the second section we discuss the simulation method in greater detail. The third section outlines a plausible file system organization, which is used as a background for the analysis. The calculation of the lower bounds on network traffic is described in the fourth section. We find that migration is advantageous compared to static placement, but few of the changes in locality (a different user accessing the file) should result in a file migration. Many files should not be moved at all, but these files did not have any common characteristics that we were able to identify. The algorithms themselves are described in the first part of Section 5, and their performances are compared in the last part of that section. In Section 5 we show that algorithms based on a batch Poisson model perform well, and the performance is insensitive to how the different parameters are estimated. The algorithms use the average fraction of a file accessed per open as the decision variable; if the fraction accessed is larger than a given limit, the file is moved to the node the user is on. The limit for migration changes with a user's reference pattern; it decreases with consecutive opens from the same user, and it increases with a change in locality. The analysis is repeated, but with several users placed at the same node. Two different strategies for placing users at the processors are analyzed, an allocation which provides an approximately even workload on the processors, and an allocation which places users from the same department at the same node. Only the latter substantially reduced the network traffic, but the relative ranking of the algorithms was unchanged. The last section analyzes the quality of the proposed algorithms and the potential for improving their performance. All definitions that are used in the chapter are displayed at the end in Table 4.13.

4.2. Method

We use trace driven simulation to evaluate the different algorithms. The traces of file system activity range in length from 8 to 30 days, and they are from Hughes Aircraft (Hughes), Amdahl Corp. (Amdahl), and the Stanford Linear Accelerator Center (SLAC). The traces and the procedure for generating them are described in Chapter 2. Since we use traces representing more than a week of activity from three different installations, we believe our simulations are representative of typical commercial installations.

4.2.1. Rules for Allocating Users to Nodes

In the system simulated, each user is permanently allocated to a processor (a node) in the system. All jobs originating from a particular user are routed to and run on the same processor. This section discusses the four schemes we use for placing users at nodes: one user per node, a nearly random allocation of users to 10 nodes, and all users from the same department allocated to the same node, either one department per node or all departments randomly allocated to 10 nodes.

In the first part of the simulation, each processor is dedicated to a single user. One user per processor is the appropriate allocation rule when the system is composed of workstations and/or personal computers. However, our results are also of general interest for systems with many users per processor, as long as no special effort has been made to locate users referencing the same files to the same processor. A typical example is a system where the users are allocated to provide an even load on the processors. Most files are shared by only a few users. The different users referencing a file will therefore most likely be placed at different nodes, provided there is a reasonable number of nodes in the system (>10 [1]). On a per file basis, it will then appear as if the users are placed at separate nodes. The network traffic will therefore be close to the one obtained in a system with one user per node. This claim will be justified by experimental data later in the chapter.

The simulation is also repeated with several users per node. Two different strategies for placing users at processors are analyzed; an allocation which provides an approximately even workload per processor, and an allocation which places users from the same department at the same processor. The former one did not substantially change the results; as previously explained, most users sharing a file will with this allocation rule be placed at separate nodes. There are several other strategies for placing users at nodes that can reduce the network traffic [Hartig75][Zadeh71]. Porcar's thesis [Porcar82] investigated several of these, including the mapping of users from the same department to the same node.

We do not seek to replicate his research, but we use the department mapping for two purposes: 1) It is an example of what can be obtained in terms of reduced network traffic by using some knowledge about the intended users. The mapping is practical, it does not require any measurements, and it is easy to administer. 2) By placing users from the same department to the same node, the reference patterns are changed. The reference process from a node to a file is the superposition of the processes from the users placed at

---

[1] To illustrate, if k users share a file in a system with n nodes, $k<n$, and the users are randomly allocated to the nodes, the probability two or more users are allocated to the same node is

$$1-\frac{n!}{(n-k)!}\cdot\left|\frac{1}{n}\right|^k$$

With 10 nodes and 4 users sharing a file, the probability that more than one user is placed at the same node is 0.50.

that node, and the results from Chapter 3, which are on a per user/file basis, are no longer valid. As an example, the number of consecutive opens can no longer be modeled by a geometric distribution. Instead, it is a mixture of several geometric distributions. Similarly, when users from the same department placed at the same node, the reference process need no longer be a renewal process. The stochastic properties of the reference process are changed, and the simulation provides us with a measure of the robustness of the various algorithms.

Only files referenced by at least two users during their lifetime (shared files) are included in the simulation. Files referenced by only one user need not generate any network traffic when they are stored at the node where the user is residing. This is a simplification, since files accessed by only one user during a trace may have been created by a different user before this trace started. In the simulation, all references to these files will appear as local ones, while in reality they may either have been remote or caused the file to be moved. We found that most of the files opened by a single user were created and scratched during the trace, so the error of excluding them should be minimal.

4.2.2. Performance Measure

We evaluate the algorithms by the network traffic created by remote file opens and file transfers. All network traffic is measured in the amount of bytes transferred. The network traffic was chosen for two reasons: 1) It is a direct measure of the events which degrade the system performance. 2) It is independent of any system parameters like network bandwidth, processor speed, and communication protocol. The results from the analysis are therefore not restricted to a particular system implementation. On the other hand, with network traffic as a measure, we are not able to consider the effect timing of the different events will have on system performance.

Even though we use a metric that is proportional to the number of bytes transferred, the metric will account for overhead from transferring a data block over the network. The sum of the network overhead for a remote open or file transfer is proportional to the number of blocks transferred, and for larger files this overhead will therefore also be proportional to the number of bytes transferred (with a fixed block size). However, the additional overhead of setting up a remote open or a file migration is ignored, since these types of overhead are highly system dependent.

We do not distinguish between network traffic caused by file transfers or by remote file opens, and we neglect the cost of updating the file catalogue when the file is moved. This cost depends on the implementation of the file system, and it is therefore ignored in this simulation. The impact such a cost could have on the results is discussed later in the chapter.

4.3. File System Organization

Our research is not about file organization, and a detailed description of the file system is not needed. The best design is a function of the actual hardware and the intended users. Our purpose is to outline a plausible structure that can serve as a background for the analysis.

4.3.1. Structure

The structure of the file system is similar to a centralized system. Each processor controls and administers the files stored at its disks. In addition, there are one or more name servers that know the location of all files. Each name server need only contain the storage location for a subset of the files. A natural way is to let each clearing house know

the location of all files from one user or department. Under such a scheme all nodes will know the clearing house they must query to find the storage location of a particular file. Similar schemes are used in many different systems (for example Grapevine [Birrel82][Birrel84] and QuickSilver [Cabrer87]). To reduce the number of queries to the clearing houses, each node has a cache with the storage locations of the most recently used files. The caches are not updated when a file is moved or erased, and will only contains hints of the correct storage location. However, since files are seldom moved, the hints will usually be correct.

4.3.2. Concurrency Control

The purpose of the concurrency control is to serialize all opens to a file. Only coarse grained locking schemes are considered. A scheme based on optimistic concurrency control is feasible (like in Amoeba [Mullen85]), but for simplicity we chose a locking scheme. A finer granularity, for example at the block level, is unnecessary, because for the systems we consider there is little need for several users to update different parts of the file simultaneously.

The various access locks to a file are managed by the node controlling the disk where the file is stored (the file host). Any node wishing to reference a file must first request and be granted an access lock by the file host. We consider two types of locks, multiple read lock and exclusive update lock. An exclusive update lock cannot be granted while other processes have a read or update lock, and a read lock cannot be granted if other processes have an exclusive update lock. For efficiency reasons, there must also be a timeout on the length a lock can be held to avoid a single user monopolizing a particular file. Schemes for handling failures of the file host are not needed, since the system cannot access the files stored at that file host even if a new lock manager could be appointed.

4.4. Lower Bounds on the Performance of Migration Algorithms

Prior to the simulation, we calculated the lower bounds for the network traffic. Two bounds are calculated, one for a static placement of the files, and one for a dynamic placement. With dynamic placement, files can be relocated as the reference patterns change.

We calculated the static and the dynamic bounds by first running the traces once to record the reference patterns to each file. We then devised the file placement that in retrospect would have resulted in the minimum network traffic (the optimal look-ahead placement). The calculation of the bounds serves three purposes: 1) The bounds serve as reference points for the simulation study, and they can be used to measure the quality of the proposed algorithms. (See [Smith78] for a similar example). 2) The analysis of the bounds provides hints of the properties migration algorithms should have. 3) The difference in performance between the static and dynamic look-ahead algorithms provides a measure of the advantage migration offers over static placement. It is a direct measure, since it is not distorted by ineffective implementation.

Since there are no storage constraints, each file can be analyzed independently. The static bound is found by placing the file at the node which has the largest I/O traffic (measured in bytes) to the file. The bound for dynamic placement is calculated by forward dynamic programming. For each file let

$V_n(i) =$ minimal network traffic to a file from the start of the trace up to the n-th reference, given the file is placed at node i after the n-th reference. The initial condition is $V_0(i) = 0$ for all i.

$B_n =$ is the I/O traffic (measured in bytes) associated with the n-th reference.

F_n is the file size after the n-th reference. $F_0 = 0$.

d_n is the identity of the node issuing the n-th reference.

The recursive formula for $V_n(i)$ is then:

$$V_n(i) = \min\left[\min_k\left[V_{n-1}(k)+F_{n-1}\right], V_{n-1}(i)\right]+B_n \cdot 1_{i \neq d_n}$$

If T is the last reference to the file, then $\min_i V_T(i)$ is the minimum network traffic for the file.

The network traffic is calculated with total knowledge of future references to all files. The traffic for the static look-ahead placement is therefore a lower bound on the network traffic for any static file placement algorithm, while the traffic for the dynamic look-ahead placement is the lower bound for any algorithm that allows file migration [Smith85].

4.4.1. Results

As previously described, each user is permanently allocated to one node. All jobs originating from a particular user are routed to and run on the same processor, and the results for the look-ahead algorithms will depend on the rule for allocating users to nodes. Several different allocation schemes are used, and the results are reported for each scheme.

4.4.1.1. One User per Node

The performance of the look-ahead placement with one user per node is summarized in Table 4.1. To make it easier to compare results for different installations, the network traffic for a particular group of files is expressed in percent of the total I/O traffic to that particular group of files. A network traffic larger than 100% means that the placement algorithm is creating more network traffic from remote I/O and migration than the total I/O traffic to the files. In the reminder of the chapter, the components of the network traffic, remote opens and file migrations, are also measured in percent of total I/O traffic to the files.

The differences between the static and the dynamic bounds vary from 7% to 27% of all I/O traffic. Translated into relative reduction of network traffic, the lower dynamic bound is from 18% to 48% lower than the static one. Migration therefore has a large potential for reducing network traffic.

To analyze the differences between files with different size and organization, we calculated the lower bounds on network traffic for large (> 1M byte) and small files, and for partitioned and sequential files. The results are displayed in Table 4.2 a, b, and c. The bounds for large files and partitioned files are similar to the overall bounds for each installation, since most of the I/O traffic is due to large files and to partitioned files. Migration is clearly beneficial for all types of files and for all sizes.

The bounds also provide useful hints of the properties a "successful" file placement algorithm should have. The dynamic bound is characterized by several measures of file movement, like the number of files that are moved and the fraction of changes in locality (a different user accessing the file) that should result in a file move. These and similar measures define the desired level of file movement for dynamic file placement algorithms. For all the installations, the network traffic due to file moves is roughly 30% of the total network traffic; the remainder is due to remote file accesses. With dynamic placement, files can be relocated to take advantage of changes in the access patterns. Yet a large fraction of the files (between 45% to 81%) remains stationary at the node that has the

Table 4.1: displays the results for the optimal look-ahead placement for the three installations. All types of network traffic for a group of files are measured in percent of the total I/O traffic to that particular group of files. The first row gives the optimal look-ahead placement for static placement, while the next 5 rows display the performance of the optimal look-ahead placement when files can be relocated or migrated.

	SLAC	Hughes	Amdahl
Lower Static Bound, the total network traffic in % of I/O traffic to the files	56.06%	39.17%	33.28%
Lower Dynamic Bound, the total network traffic in % of I/O traffic to the files	28.81%	32.18%	24.99%
Lower Dynamic Bound, the network traffic due to remote I/O in % of I/O traffic to the files	19.67%	23.47%	18.03%
% of the files that remain at the same location when the lower dynamic bound is calculated	44.50%	81.29%	79.84%
% of changes in locality that result in a file move for the dynamic placement	15.32%	11.10%	3.57%
number of file moves	1876	1594	1111
average number of file moves per file that is moved	7.9	1.5	2.2
number of shared files	427	5666	2485
number of opens to shared files	39296	51217	238182
number of changes in locality	12247	14361	31137
I/O traffic to shared files in 10^9 bytes	11.45	39.45	14.51

largest I/O traffic (measured in bytes) to the file. This is not necessarily the node that created the file, and with a realistic placement algorithm at least one or more moves may be necessary to reach that node. By realistic, we mean an algorithm without knowledge about future references. In addition, only a few (between 4 and 15%) of the changes in locality (a different user referencing the file) result in a file move. File migration algorithms should therefore let only a few changes in locality result in a file move, and a large fraction of the files should not be moved at all.

The usage patterns for sequential and partitioned files are different. A larger fraction of the sequential files remains stationary when the dynamic bounds are calculated. But the ones that are moved are relocated often; the fraction of changes in locality that result in a file move (7% to 26%) is higher for sequential files. The usage of sequential files therefore seems to be characterized by two patterns: one for which the file should remain stationary at one node, and one for which many of the changes in locality should result in a file move. Similar patterns are not apparent for partitioned files.

No clear conclusion can be drawn from the characteristics of the files that are not moved when the lower bounds are calculated. They do not have a particular organization or size. As previously explained, a larger fraction of sequential files remains stationary, but the fraction is high for all file types. For Hughes, these stationary files have a smaller

Table 4.2a: displays the performance of the look-ahead placement for the trace from SLAC. The results are with one user per processor. All types of network traffic for a group of files are measured in percent of the total I/O traffic to that particular group of files. The first row displays the result for static look-ahead placement, while the next 7 display the results for the dynamic look-ahead placement.

	Total	Partitioned files	Sequential files	Small files < 1Mb	Large files > 1Mb
Lower Static Bound, the total network traffic in % of I/O traffic to the files	56.06%	58.44%	33.05%	47.55%	57.82%
Lower Dynamic Bound, the total network traffic in % of I/O traffic to the files	28.81%	29.29%	25.25%	27.32%	29.12%
Lower Dynamic Bound, the network traffic due to remote I/O in % of I/O traffic to the files	19.67%	19.91%	18.06%	17.02%	20.22%
% of the files that remains at the same location when the lower dynamic bound is calculated	44.50%	36.14%	55.83%	43.62%	46.51%
% of changes in locality that results in a file move for the dynamic placement	15.32%	13.55%	26.64%	14.95%	15.95%
number of file moves	1876	1386	470	1155	721
average number of file moves per file that is moved	7.9	7.5	6.5	7.0	10.4
mean file size in Kb	1043	1182	838	not calculated	not calculated
mean file size in Kb for files that remain stationary for the **Lower Dynamic Bound**	1179	1289	1052		
number of files	427	249	163	292	129
number of changes in locality	12247	10231	1764	7726	4521
I/O traffic to the files in 10^9 bytes	11.5	10.5	0.9	2.0	9.5

mean file size than the overall files, while for SLAC and partitioned files at Amdahl, the mean file size is larger.

To summarize, the difference between static and dynamic look-ahead placement is equivalent to from 7% to 27% of the I/O traffic to shared files. Thus migration is beneficial for all types of files. A substantial part of the reduction is due to relocating

large and/or partitioned files. The analysis showed that not all files should be moved, and that only a few changes in locality should result in a file move. We are not able to characterize, either by file organization or file size, the files that should not be moved.

Table 4.2b: displays the performance of the look-ahead placement for the trace from Hughes. The results are with one user per processor. All types of network traffic for a group of files are measured in percent of the total I/O traffic to that particular group of files. The first row displays the result for static look-ahead placement, while the next 7 display the results for the dynamic look-ahead placement.

	Total	Partitioned files	Sequential files	Small files < 1Mb	Large files > 1Mb
Lower Static Bound, the total network traffic in % of I/O traffic to the files	39.17%	41.78%	46.71%	35.14%	39.79%
Lower Dynamic Bound, the total network traffic in % of I/O traffic to the files	32.18%	34.28%	39.87%	29.06%	32.66%
Lower Dynamic Bound, the network traffic due to remote I/O in % of I/O traffic to the files	23.47%	26.83%	23.70%	19.44%	24.09%
% of the files that remains at the same location when the lower dynamic bound is calculated	81.29%	80.37%	81.92%	81.82%	71.85%
% of changes in locality that results in a file move for the dynamic placement	11.10%	3.87%	18.75%	13.34%	6.02%
number of file moves	1594	267	1198	1329	265
average number of file moves per file that is moved	1.5	3.1	1.3	1.4	3.1
mean file size in Kb	292	1681	149	not calculated	not calculated
mean file size in Kb for files that remain stationary for the **Lower Dynamic Bound**	268	1611	130		
number of files	5666	433	5066	5364	302
number of changes in locality	14361	6899	6389	9960	4401
I/O traffic to the files in 10^9 bytes	13.6	8.6	3.1	1.8	11.8

Table 4.2c: displays the performance of the look-ahead placement for the trace from Amdahl. The results are with one user per processor. All types of network traffic for a group of files are measured in percent of the total I/O traffic to that particular group of files. The first row displays the result for static look-ahead placement, while the next 7 display the results for the dynamic look-ahead placement.

	Total	Partitioned files	Sequential files	Small files < 1Mb	Large files > 1Mb
Lower Static Bound. the total network traffic in % of I/O traffic to the files	33.28%	31.08%	33.30%	25.47%	34.23%
Lower Dynamic Bound. the total network traffic in % of I/O traffic to the files	24.99%	25.68%	28.32%	19.83%	25.61%
Lower Dynamic Bound. the network traffic due to remote I/O in % of I/O traffic to the files	18.03%	21.88%	25.20%	15.53%	18.33%
% of the files that remains at the same location when the lower dynamic bound is calculated	79.84%	76.62%	87.68%	80.22%	78.55%
% of changes in locality that results in a file move for the dynamic placement	3.57%	2.75%	7.35%	5.52%	1.98%
number of file moves	1111	694	198	772	339
average number of file moves per file that is moved	2.2	2.0	1.8	2.1	2.8
mean file size in Kb	1905	1902	885	not calculated	not calculated
mean file size in Kb for files that remain stationary for the **Lower Dynamic Bound**	1984	1912	879		
number of files	2485	1463	885	1821	564
number of changes in locality	31137	24277	2693	13988	17149
I/O traffic to the files in 10^9 bytes	34.0	17.3	2.2	3.7	30.4

4.4.1.2. The Results with Several Users Allocated to the Same Node

In this section we analyze the effect mapping several users to the same node will have on the results. We previously claimed that the performance of the various algorithms would be the same if we mapped several users to the same node, as long as we did not make any specific effort to map users sharing the same files to the same node. Most files

are only referenced by a few users. With a reasonable number of nodes in the system, it is likely that users sharing a file will be located at different nodes, and the results with one user per node will be representative.

This claim is confirmed by the two first columns of Table 4.3, which display the number of changes in locality (a different node accessing the file) and the lower bounds for different placement strategies. The first column tabulates the results with one user per node, the second with all users allocated to 10 nodes, with an average of 49 users per node at SLAC and 611 at Amdahl. Only the traces from SLAC and Amdahl are shown, since the users' department could not be identified for the trace from Hughes. The users were ordered according to their cpu usage during the trace period. They were then allocated to the 10 nodes in decreasing order by an elevator algorithm. Users with a heavy use of cpu time will therefore be placed at different nodes. This ensures a reasonable load balance (measured in cpu seconds), and an even load of users to each node.

Table 4.3: displays the performance of the look-ahead placement algorithms with different schemes for allocating users to nodes. The allocation schemes are explained in the text. Only for two of the traces, SLAC and Amdahl, could we implement all schemes. Only results for these two traces are included in the table.

		one user per node	users allocated to 10 nodes, sequentially ordered by cpu usage	users from the same department allocated to the same node, one node per department	the departments allocated to 10 nodes sequentially by their cpu usage
Lower dynamic bound	SLAC Amdahl	28.81% 24.99%	23.66% 23.67%	11.18% 11.63%	10.56% 11.36%
Lower Static Bound	SLAC Amdahl	56.06% 33.28%	48.85% 32.09%	22.53% 16.09%	21.36% 15.83%
Changes in locality	SLAC Amdahl	12247 31137	10794 28842	6816 14598	6529 13842

Allocating several users to the same node eliminates only a few changes of locality, i.e., a different node referencing the file. The number of changes in locality are only reduced by 12% at SLAC and 7% at Amdahl. The performance of the optimal look-ahead algorithm therefore remains nearly unchanged, as we claimed it would be. The results from the analysis with one user per node will therefore still hold, when several users are allocated to the same node, as long as no special effort has been made to allocate users referencing the same files to the same node.

However with different rules for allocating users to the same node, the reference patterns on a per node basis are substantially changed. The two last columns of Table 4.3 tabulate the lower bounds when users from the same department are placed at the same node. Only for SLAC and Amdahl could we identify the department a user belongs to. There are 61 departments active during the trace at SLAC and 83 departments active at Amdahl. Column three displays the lower bounds with one department per node, while column four is with the departments placed at 10 nodes (with the same location procedure used as for column two).

The department mapping reduces both the number of changes in locality and the lower bounds for network traffic. The number of changes in locality is reduced by 44% at SLAC and 53% at Amdahl, while the lower bounds are reduced $\approx$ 53% at SLAC and 51% at Amdahl. Even though we use more than 60 nodes, the number of changes in locality is much lower than when all users were placed at only 10 nodes in a roughly random procedure. Apparently a large part of the sharing of the files is between users from the same department. This is not unexpected, since users from the same department use the same programs, develop programs together, and share the same functions and organizational procedures. The optimal look-ahead placement therefore results in a much lower network traffic when we place users from the same department at the same node.

The department mapping reduces the number of changes in locality. In addition, the number of consecutive opens (runs) from the same node should increase when several users are mapped to the same node. With longer runs, it should be feasible to move a file more often; the cost of a file move can be amortized over a larger number of opens. However, the fraction of changes in locality that result in a file move remains nearly constant (see Table 4.4). The mean number of moves per file that is moved also remains the same. The migration patterns therefore seem unchanged, only the absolute number of file moves is reduced.

It does not seem to matter whether we locate one or more departments to the same node. For column four in Table 4.3, we repeated the location procedure used for column two (the departments were ordered by descending cpu usage and placed at 10 nodes with an elevator algorithm). The difference between column three and four is small (2.2% to 5.5%). The results obtained with one department per node will therefore be representative for the case when several department are placed at the same node, as long as the number of nodes is reasonable (e.g. > 10).

To summarize, the number of nodes is the system is less important than how users are placed at the nodes. With users from the same department placed at the same node the network traffic can be reduced substantially.

Table 4.4a: displays the performance of the look-ahead placement for the trace from SLAC. The results are with users from the same department are placed at the same node. All types of network traffic for a group of files are measured in percent of the total I/O traffic to that particular group of files. The first row displays the result for static look-ahead placement, while the next 7 display the results for the dynamic look-ahead placement.

	Total	Partitioned files	Sequential files	Small files < 1Mb	Large files > 1Mb
Lower Static Bound. the total network traffic in % of I/O traffic to the files	22.53%	24.30%	3.95%	18.02%	23.46%
Lower Dynamic Bound. the total network traffic in % of I/O traffic to the files	11.18%	12.00%	2.57%	10.97%	11.24%
Lower Dynamic Bound. the network traffic due to remote I/O in % of I/O traffic to the files	7.46%	8.02%	1.74%	7.81%	7.39%
% of the files that remains at the same location when the lower dynamic bound is calculated	75.88%	71.89%	82.21%	75.17%	77.52%
% of changes in locality that results in a file move for the dynamic lower bound	11.94%	9.81%	26.68%	10.38%	15.07%
number of files moves for the dynamic lower bound	814	588	210	472	342
mean number of file moves for files that are moved	7.9	8.4	7.2	6.4	11.8
number of files	427	249	163	298	129
number of changes in locality	6816	5996	787	4547	2269
Total I/O traffic to the files	11520 Mb	10526 Mb	862 Mb	1969 Mb	9551 Mb

Table 4.4b: displays the performance of the look-ahead placement for the trace from Amdahl. The results are with users from the same department are placed at the same node. All types of network traffic for a group of files are measured in percent of the total I/O traffic to that particular group of files. The first row displays the result for static look-ahead placement, while the next 7 display the results for the dynamic look-ahead placement.

	Total	Partitioned files	Sequential files	Small files < 1Mb	Large files > 1Mb
Lower Static Bound, the total network traffic in % of I/O traffic to the files	16.09%	11.32%	12.80%	11.42%	16.66%
Lower Dynamic Bound, the total network traffic in % of I/O traffic to the files	11.63%	9.77%	8.38%	8.57%	12.00%
Lower Dynamic Bound, the network traffic due to remote I/O in % of I/O traffic to the files	6.88%	8.11%	6.57%	7.03%	6.86%
% of the files that remains at the same location when the lower dynamic bound is calculated	89.05%	86.19%	94.46%	89.38%	87.94%
% of changes in locality that results in a file move for the dynamic lower bound	3.39%	2.44%	8.04%	5.80%	1.67%
number of files moves for the dynamic lower bound	495	304	77	353	142
mean number of file moves for files that are moved	1.8	1.5	1.6	1.7	2.1
number of files	2485	1463	885	1921	564
number of changes in locality	14598	12462	958	6090	8508
Total I/O traffic to the files	34038 Mb	17291 Mb	2234 Mb	3764 Mb	39363 Mb

Migration is still advantageous when users from one department are placed at the same node, and the behavior is similar to the one with one user per node. Only a few of the changes in locality should result in a file move, and most files should not be moved (76% at SLAC and 89% at Amdahl). A smaller fraction of sequential files should be moved, but those that should be moved are migrated more frequently, since the fraction of changes in locality that result in a file move is higher. Overall, the department mapping does not change the qualitative aspect of the optimal look-ahead placement. The optimal network traffic is, however, sharply reduced.

4.5. The Algorithms

We analyze five different groups of algorithms. Two of these, MRU and static placement represent the extreme cases of possible strategies. Under MRU (Most recent User) a file is always migrated when a different location starts using it. The other extreme is to never move the file, but to keep it stationary regardless of the location that is using the file. In Chapter 3 we showed that there can exist algorithms with a better performance. These algorithms should occasionally relocated some of the files. The remaining three groups are examples of strategies that will migrate only some of the files. Each group is based on different underlying models of how users reference files. These groups are: algorithms based on a Batch Poisson model, algorithms based on a stack model, and algorithms estimating the best possible static placement.

4.5.1. Detailed Description of the Algorithms

4.5.1.1. MRU Algorithms

This class consists of algorithms that move the file to the node currently using it (Most Recent User). If there is locality among the file references, this node is likely to continue using the file and the next references can be handled locally. One disadvantage with MRU is that the file will be moved each time there is a single reference from a different user. The cost of moving the file to and from the new location will be larger than the cost of single remote open, and it is only advantageous to move when there will be several consecutive opens from the same location. We found in the exploratory analysis that some nodes referenced a file only once, and to migrate a file to these nodes is a waste. An algorithm that did not migrate the file until there are two consecutive opens from the same location may therefore have a better performance. This algorithm was labeled DMRU. Such an algorithm avoids unnecessary file migrations to locations that will never reference the file again at the cost of always having a remote open before the file is migrated. The underlying assumption behind DMRU is similar to the assumption for MRU; the algorithm will only work if the locations that have two consecutive opens are likely to continue to reference the file. Both MRU and DMRU are evaluated, and the algorithms have the following features:

MRU In the Most Recently Used algorithm, a file is always moved to the node referencing it. All opens are therefore local ones. The algorithm needs no extra state information: whenever the file is referenced from a remote location, it is moved to that location.

DMRU The Delayed Most Recently Used algorithm operates similarly to the MRU, but it does not move a file until there are two consecutive opens from the same node. This algorithm needs one state variable per file to record the identity of the last node to access the file.

4.5.1.2. Static Placement

Only one static placement algorithm was analyzed (STAT0). It is the simplest possible static placement algorithm: Each file is placed at the first node to reference it. The simulation results are only approximations, since the first node (in the trace) to reference a file is not necessarily the creator. This algorithm combined with the MRU algorithms set the target for the remaining algorithms. Any well-designed algorithm must have a better performance than these two.

4.5.1.3. Algorithms Estimating the Best Static Placement

Since the network traffic is used as a metric, the best static placement is the node with the largest I/O traffic to the file. This node can, however, not be identified until the file is erased. A possible placement scheme is then to estimate the node that will have the largest I/O traffic. This estimate can change over time, and the files may be migrated to new locations. The cumulative I/O traffic to the file is used to estimate the node that will have the largest remaining I/O traffic to it, and the best static placement is estimated to be the node with the largest cumulative I/O traffic to the file so far.

This is an unstable scheme during the first few opens to the file. If these opens are from different locations, the estimate for the best static location may change several times. A version of the algorithm that avoided this unstable behavior was also included. The two different algorithms in this class are:

STAT1 The file is moved to the node with the largest cumulative I/O traffic so far. The placement can therefore change over time as the reference activity of the nodes changes. The algorithm may be unstable during the first few references to a file; within a few opens by different nodes the owner may change several times. The algorithm needs two state variables for each node that uses the file, one for the node id and one to accumulate the I/O traffic.

STAT2 This algorithm is a deviation of the previous algorithm. It was constructed to avoid the unstable behavior of STAT1 during the first few references; the file is placed at the first node to reference the file (STAT0) until 10 changes of locality are observed, after that it operates as STAT1. The algorithm needs the same number of state variables as STAT1.

4.5.1.4. Algorithms based on a Stack Model

The stack model is a extension of a model developed to explain locality in memory references [Spirn77]. An analysis of the model is found in Chapter 3. In the model, users (or nodes) are kept in order on a stack, with the most recent user kept at the top. The model does not distinguish between users. Instead, it assumes the stack position is the only variable that determines the future reference pattern from a particular user (or node).

The model assumes that with constant probability p_i the node in stack position i will reference the file next. Whether to migrate the file to the node currently referencing the file, i.e., the node in stack position 1, is a function of the number of consecutive opens from this node. The file should only be migrated if the advantage of local I/O for these references is larger than the migration cost. A conservative estimate for the cost of migrating the file is 2 * file size, i.e., the cost of moving the file to its new location plus the cost of moving the file back again to its current location. The expected number of consecutive opens from a node, i.e., the number of references made from stack position 1, is $\frac{1}{1-p_1}$. Let B_j be the number of bytes referenced per open by node j, F be the file size, and $f_j = \frac{B_j}{F}$ be the fraction of the file accessed per open from node j. The file should then be migrated to the node currently using it (node j) when the expected advantage of local I/O is larger than the cost of migrating the file. This condition can be expressed as: migrate when $E(B_j) \cdot \frac{1}{1-p_1} > 2 \times F$ or $E(f_j) > 2 \cdot (1-p_1)$.

The last condition depends only on the fraction accessed per open. When a file is referenced according to a stack model, all nodes have the same reference pattern, and the

nodes differ only in the fraction of a file accessed per open. It is therefore to be expected that the fraction accessed is the only decision variable that is needed. The migration condition can then be summarized as: the file should only be migrated to nodes that access more than a certain fraction ($lim = 2 \cdot (1-p_1)$) of the file per open.

The empirical values for p_1 found in the traces were 0.6775 at SLAC, 0.6040 at Hughes, and 0.8588 at Amdahl (see Table 3.6). This probability was translated into a limit (*lim*) for the fraction accessed before a file should be migrated. This is only a valid procedure when each node is dedicated to a single user. For the other location schemes the empirical stack probabilities were not measured, and the procedure will not result in the optimal placement under a stack model.

We evaluate two algorithms that are based on stack model of how nodes reference a file. They are:

STACKa — This algorithm migrates the file to the node referencing it only when a move flag is set to move. There are two conditions for setting the flag to move: 1) the current reference is from the same node as the last one, and 2) the last open accessed more than *lim* of the file. The algorithm is an adaptation of DMRU. We chose to set the flag to no-move each time there is a change in locality, since the correlation between the fraction accessed per open by different users is low. The fraction one user accesses is therefore not a good predictor of the fraction other users will access. (This is documented in the previous chapter.) STACKa needs two state variables for each file, one for the identity of the last node to reference it, and one for the flag.

STACKb — This algorithm is very similar to STACKa, but the move flag is **not** set to no-move each time there is a change in locality. The algorithm will therefore operate as MRU as long as the move flag is set, and the file will be moved to all nodes accessing it. The move flag is set to no-move only when an open accesses less than *lim* of a file, and the flag is cleared every time the opposite event takes place. The algorithm needs only one state variable, the move flag, per file.

4.5.1.5. Algorithms based on a Batch Poisson Model

These algorithms make migration decisions based on a batch Poisson model of how users reference files. In a batch Poisson model, the references from a user to a particular file are lumped together in batches. The batches are issued according to a Poisson process, and it is assumed that the time between opens from the same batch is substantially shorter than the time between batches. The number of opens in a batch has a geometric distribution. A batch Poisson model is designed to capture the tendency of users to reference a file in bursts of activity. Even though not all references are clustered, the model is appropriate as long as the time between the bursts of activity from a user is longer than the time between the individual opens. To simplify the discussion, we define a run as a set of consecutive opens from the same user to a particular file.

The storage cost is assumed to be identical for all nodes. The time a file spends at a particular location is then of no interest. For single copy file systems, only the order users reference files is of interest, and only the properties of this embedded chain are important. With a Batch Poisson model, the order users reference a file is a Markov chain, where the states are the identity of the users. The probability the jth user will open the file after an open from the ith user is given by the transition probability p_{ij} of the chain.

Under this model, the optimal piacement can be found using stochastic dynamic programming [Ross83]. To model the limited lifetime of a file, let us assume there is a constant probability, α, that the file will be erased when it is closed. This is an assumption used only to illustrate the stochastic dynamic programming formulation. Let $V(i,k)$ be the expected minimal cost of continuing with the optimal placement policy given that the file is placed at node i and that node k has just referenced the file.
Using the same definitions as in section 4.4 the stochastic dynamic formulation is then:

$$V(i,k)=\min\left[\left[B_k+\alpha\sum_j p_{k,j}V(i,j)\right],\left[F+\alpha\sum_j p_{k,j}V(k,j)\right]\right] \qquad 4.2$$

$$V(i,i)=\alpha\sum_j p_{i,j}V(i,j)$$

This stochastic dynamic programming formulation can be solved with standard techniques, like policy improvement [Ross83]. Theoretically, the optimal placement can be found, but it is not a practical approach, since, in the worst case, all possible solutions must be enumerated. More important, the approach can only work if the transition probabilities are known, which restrict the usefulness for any realistic file system. Instead, we chose to investigate simpler heuristics where we equate the advantage / disadvantage of migration over a short time horizon.

To explain the migration criterion we need the following definitions:

$E(CB_j)$ =	expected number of bytes to be transferred by consecutive opens (a run) from node j. The references are from one or more consecutive batches. Since we assume files are accessed according to a batch Poisson model with geometric batch sizes, the number of consecutive references will have a geometric distribution. $E(CB_j)$ is also the expected number of bytes that remain to be transferred by the same node before a different node starts using the file.
F =	file size
p_i =	is the probability a batch of opens will come from node i. If λ_j is the rate node j will issue batches of reference, then $p_i = \frac{\lambda_i}{\sum_j \lambda_j}$.

To simplify the explanation, let us assume that the file is currently placed at node i and that node j has just referenced the file. The decision we must make is whether to migrate the file to node j or not. The time horizon we use is the time until a node other than j starts referencing the file. The expected advantage of migrating the file to j is $E(CB_j)$. If the file will be moved directly back to where it came from, the cost of the migrating the file to and from the node is set to $2F$, since the file size rarely changes. If the file is never moved back, the cost is set to F. In the general formulation, the cost is set equal to $(1+\gamma)F$, where γF is the fraction of the cost from moving the file back to where it came from that must be amortized to node j. Generally, γ is a function of the current location of the file and the whole transaction matrix. The components of the migration decision are therefore the expected number of consecutive opens and the transition probabilities.

In the heuristics we analyze, γ is set to the probability that the next change in locality will be back to node i (that node i will the next node to reference the file). If another node references the file, j should not amortize any of the cost of moving the file back again to i. Let us define p'_{ji} as the probability node i will reference the file after node j. The decision criteria is then: migrate the file from node i to node j if $E(CB_j) > F(1+p'_{ji})$.

Under a batch Poisson model with geometric batch size, the transition probabilities are $p_{ij} = (1-k_i)p_j$, where p_j is the probability the batch is from node j and the mean batch size is $\frac{1}{1-k_i}$. This is assuming that batches from different nodes will not overlap. For a more thorough discussion we refer to Chapter 3. The probability node i will be the next node to open the file after the batches from node j is then $p'_{ji} = \frac{p_i}{1-p_j}$ if we disregard that batches can overlap.

4.5.1.5.1. Estimation Methods

We developed several versions of this algorithm with different methods for estimating $E(CB_j)$ and p'_{ji}. To simplify the discussion, let N_j be the number of opens to a particular file from node j. Let C be the number of changes in locality so far for the file, and C_j be the number of changes to node j. Let M be the number of times the file is moved and M_j the number of moves away from node j. The following estimations schemes are used:

Meth 1 *E(CB)* = The average of the previous runs of references from the node. During the first run, the expected remaining number of bytes in the run is estimated equal to the number of bytes transferred so far, since the run size has a geometric distribution.

Meth 2 *E(CB)* = The moving average of the previous runs, with a coefficient of 0.75. New estimate = old estimate * 0.75 + 0.25 * number of bytes observed for last run. The estimates for the first run is the same as in the previous method.

Meth 3 is a Bayesian estimate of $E(CB_j)$ based on a geometric batch size. The Bayesian approach was chosen for two reasons: a) It is an interesting approach in itself. b) It also incorporates some belief or knowledge about the various model parameters. Fewer observations are therefore needed before a decision can be made. During the preliminary analysis we discovered that part of the network traffic was remote references from nodes that the file eventually would be moved to. By migrating the files earlier some of this unnecessary network traffic could be avoided. With the prior distribution we use, the estimate for $E(CB_j)$ is larger than the one based on the average of previous values. Files would therefore tend to be migrated earlier with this estimation method, and the method could improve performance.

The number of consecutive opens in a batch Poisson model can consist of several batches when the same user is issuing the batches sequentially. For our purpose, the individual batches themselves are of less interest; the important aspect is the total number of consecutive opens. In the batch Poisson model with geometric batch size that we discussed in Chapter 3, the number of consecutive opens will have a geometric distribution with mean $\frac{1}{1-(k+(1-k)p_i)}$ (using the same notation as in Chapter 3). For our purpose, it is sufficient to estimate $r_i = k+(1-k)p_i$, the probability the next open will come from user i given the current open is from user i. Initially, we believe that r_j has a prior beta distribution given by the density function

$$F(R_j < x) = \int_0^x \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \times r_j^{\alpha-1}(1-r_j)^{\beta-1} dr_j$$

Our belief in the the shape of the distribution function is updated by the observed reference pattern. For each new consecutive refence (same locality), α is increased by one. When there is a change in locality, β is increased by one. After observing one more reference, the new density function for r_j will then either be a Beta(α+1,β) or a Beta(α,β+1). After observing N references (not including the current) and C changes in locality for a particular user/file pair, the expected number of consecutive references is:

$$\int_0^1 \left[\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \times r_j^{\alpha-1}(1-r_j)^{\beta-1} \times \frac{1}{1-r_j} \right] dr_j = \frac{N+\alpha_0+\beta_0-1}{C+\beta_0-1}$$

We initially chose $\alpha_0 = 1$ and $\beta_0 = 2$, which corresponds to an initial expectation of two consecutive references from each node. The density function for r_j is then $2\times(1-r_j)$. This implies $E(CB_j) = \frac{N_j+2}{C_j+1}$ E(bytes per access). The prior distribution is analyzed later in the chapter.

The second part of the batch Poisson algorithm is the estimate for p'_{ji}, the probability user i will open the file following an open from user j. This is the transition probability in the Markov chain that models the order different users will access a file. In such a chain $p'_{ii} = 0$. As previously mentioned $p'_{ji} = \frac{p_i}{1-p_j}$, where p_i is the probability a batch of reference will come from node i. Let $C_{j,i}$ be the number of changes of locality we have observed from node j to node i, while $C_{j.}$ is the total number of changes away from node j ($C_{j.} = \sum_k C_{j,k}$). One estimator for p'_{ji} is then $\frac{C_{j,i}}{C_{j.}}$. This estimator is not particularly good, since we need to record $C_{j,i}$ for all i and j. When only a few changes are observed, which is true for most files (see Chapter 2), the estimates are sensitive to the integer nature of C_{ij} and $C_{i.}$. This is especially a problem each time a new node (node k) starts using the file. Several changes in locality away from the node (node k) must be observed before the estimators settle down.

We therefore chose a simpler estimate based on the number of observed migrations. The underlying assumption is that $p'_{ji} \approx p'_{ki}$ for all k. All previous moves to or from node can then be utilized for the the estimate. Let M_i be the number of times the file has been moved to node i, and M the total number of times the file has been moved so far. An approximate estimator for p'_{ji} is then $\frac{M_i}{M}$. This is a more robust estimator, but during the first few moves it is sensitive to the integer nature of M. The first time a file has been migrated to node k, $p'_{k.}$ is set to 1, which means we are reluctant to leave it. This latter heuristic gave a slight performance improvement during the initial simulation study, which is why it is kept in the final study. In an initial study, we also used estimates based on the number of changes in locality (see previous paragraph). However, the network traffic was higher than when we used estimates based on the number of migrations. The estimators based on the number of migrations were therefore chosen for the final simulation.

Another potential model is a variation of the batch Poisson model with geometric batch size and uniform transition probabilities (λ_j). Let us assume that all users are issuing batches of opens at the same rate. The expected length of the batches will be different, so the expected number of opens per user will vary between the nodes using a file. This model was previously discussed in Chapter 3. The transition probabilities are then given by

$$p_{i,i} = k_i + (1-k_i)\frac{1}{n}$$

$$p_{i,j} = (1-k_i)\frac{1}{n}$$

where n is the number of nodes sharing the file and $\frac{1}{1-k_i}$ is the mean batch size for node i. This is assuming we can ignore the possibility that batches can overlap. The order users reference a file is then given by a Markov chain with transition probability $p'_{ji} = \frac{1}{n-1}$. This is the second estimation method that is analyzed.

The different estimates for p'_{ji} that we use are then:

$p'_{ji} = 0$ $\quad$ $p'_{ji} = 0$ implies a node will only access a file once.

$p'_{ji} = 1$ $\quad$ $p'_{ji} = 1$ implies a node will access the file repeatedly. If $E(CB) > 2 \cdot F$, which is equivalent to $p'_{ji} = 1$, it is always optimal to move the file to the node.

$p'_{ji} = \frac{M_j}{M}$ $\quad$ p'_{ji} is estimated only for nodes to which the file already has been moved to.

$p'_{ji} = \frac{1}{n-1}$ $\quad$ where n is the number of nodes using the file.

To illustrate the effect of the various approaches we compared 7 different algorithms. The first four use different methods for estimating p'_{ji}, while the estimate for $E(CB_j)$ remains the same. The last three use different methods for estimating $E(CB_j)$ while p'_{ji} is estimated with the same method. The algorithms are:

BAT0	$E(CB_j)$ is estimated using a Bayesian estimate (Meth 3). $p'_{ji} = \frac{1}{n-1}$.
BAT1	has the same estimate of $E(CB_j)$ as BAT0, but assumes $p'_{ji} = \frac{M_j}{M}$.
BAT2	has the same estimate of $E(CB_j)$, but assumes $p'_{ji} = 1$.
BAT3	has the same estimate of $E(CB_j)$, but assumes $p'_{ji} = 0$.
BAT4	has the same estimate as BAT1 of the p'_{ji}, but uses the average for $E(CB_j)$ (METH 1).
BAT5	has the same estimate of the p'_{ji} as BAT1, but uses the moving average for $E(CB_j)$ (METH 2).

4.6. Results

The performance of the different algorithms is shown in Table 4.5 The network and migration traffic are scaled with the total I/O traffic to shared files, so the different installations can be compared. The results are first reported with one user per processor, then with one department per processor.

4.6.1. One User per Processor

The performance of the various algorithms is displayed in Table 4.5. Two measures are used, the amount of network traffic (label "n%") and the number of file moves (label "nm"). The network traffic is measured in percent of total I/O traffic to shared files, so it is easier to compare the three installations. The same measures and the same labels are used in the reminder of the chapter.

The strategy of always moving the file to the processors using it (assuming the file is not already placed there) has the worst performance at all installations. To restrict the migration to only some of the processors reduces the network traffic substantially. Overall, the algorithms based on a batch Poisson process have the best performance, but for one installation (Amdahl) the gain is small compared to the static placement at the first processor to reference the file.

4.6.1.1. Results for MRU Algorithms

The MRU algorithm moves the file to the user referencing it, unless it has already been moved there. As already explained in chapter 3, such a strategy is wasteful, since few processors or nodes will reference the file often enough to justify a move. This was also confirmed by the calculation of the optimal look-ahead placement, where only a few percent (from 3% to 15%) of the changes in locality (a different processor referencing the file) should result in a file move. For sequential files, the MRU algorithms has a much better performance. Adding the restriction that the file cannot be moved before there are two consecutive opens from the same processor (DMRU) substantially reduces the network traffic and the number of file moves. The latter is reduced from 55% to 71%, and a large number of the changes in locality must be due to a single reference from a different user. Except for some extreme reference patterns, it will never be optimal to move a file to a node that issue only a single reference, since the cost of moving the file back again to where it came from will be larger than the benefit. DMRU's performance is much better for sequential files than for partitioned, index sequential, and direct files.

4.6.1.2. Static Placement

The static placement algorithm STAT0, which places the file at the first node to reference it, was included only for comparison. It has better performance than MRU, but it is not as good as for the dynamic algorithms. Yet the difference is small for the trace from Amdahl, where the difference between STAT0 and the best dynamic algorithms is $\approx$11% or 5.2% of the I/O traffic to shared files. For this installation, static placement is an acceptable strategy.

4.6.1.3. Estimating the Best Static Placement

The optimal static look-ahead placement is at the node with the largest I/O traffic to the file. The simple strategy of placing the file at the first node to access the file (STAT0) may not be the optimal static placement. The difference in performance between the optimal static look-ahead placement and STAT0 is equivalent to 14% (SLAC), 13% (Hughes), and 20% (Amdahl) of the I/O traffic to shared files. However, the optimal

Table 4.5a: The table displays the results for the various policies for the trace from SLAC. Two measurements are displayed, the network traffic (label "n%") and the number of file moves (label "nm"). The network traffic is measured in percent of all I/O traffic to shared files. The results are obtained with one user per processor.

Policy	Measure	All files	Sequential files	Partitioned files	Small (<1 Mbytes) files	Large (>1 Mbytes) files
MRU	n%	125.5	57.2	131.1	117.5	127.1
DMRU	n%	70.0	41.7	72.5	54.8	73.1
STAT0	n%	71.0	51.9	72.5	66.3	71.9
STAT1	n%	61.6	42.1	63.6	51.7	63.6
STAT2	n%	65.1	51.2	66.2	59.3	66.3
STACKa	n%	56.3	43.8	57.2	44.6	58.7
STACKb	n%	69.6	43.4	71.8	44.4	74.8
BAT0	n%	44.9	38.3	45.6	39.6	46.1
BAT1	n%	46.1	37.2	46.9	40.0	47.3
BAT2	n%	46.7	37.5	47.6	41.1	47.8
BAT3	n%	46.0	39.0	46.8	39.6	47.3
BAT4	n%	46.7	40.2	47.5	41.1	47.9
BAT5	n%	46.2	40.1	46.9	40.7	47.3
MRU	nm	12247	1764	10231	7726	4521
DMRU	nm	4162	767	3307	2763	1399
STAT0	nm	0	0	0	0	0
STAT1	nm	337	103	220	219	118
STAT2	nm	162	48	108	108	54
STACKa	nm	1328	464	846	1002	326
STACKb	nm	3208	1153	2021	2262	946
BAT0	nm	3026	968	2034	2126	900
BAT1	nm	2494	854	1616	1754	740
BAT2	nm	1802	583	1203	1210	592
BAT3	nm	3380	1075	2280	2362	1018
BAT4	nm	2651	829	1800	1834	817
BAT5	nm	2681	823	1836	1868	813

static placement is a look-ahead strategy, since it cannot be determined until the file is erased. The two algorithms STAT1 and STAT2 try to estimate the best static placement and move the file to this location. They use the cumulative I/O traffic from a node, and since this changes over time, the estimated best location will also change and the files can be moved. STAT1 has a lower network traffic than STAT0 for all installations, for all file types, and for both small and large files. The only exception is large index sequential and direct files at Amdahl where the network traffic is increased. The difference between STAT1 and the optimal static look-ahead placement is equivalent to 5.5% (SLAC), 7.65% (Hughes), and 19.8% (Amdahl) of the I/O traffic to shared files.

The estimate of the node with the largest I/O traffic is not always stable. Especially during the few first opens will the estimate be unstable, because the node with the largest

Table 4.5b: The table displays the results for the various policies for the trace from Hughes. Two measurements are displayed, the network traffic (label "n%") and the number of file moves (label "nm"). The network traffic is measured in percent of all I/O traffic to shared files. The results are obtained with one user per processor.

Policy	Measure	All files	Sequential files	Partitioned files	Small (<1 Mbytes) files	Large (>1 Mbytes) files
MRU	n%	195.9	61.1	277.0	108.2	209.3
DMRU	n%	93.5	54.3	123.1	50.9	100.0
STAT0	n%	51.8	56.4	57.6	46.9	52.6
STAT1	n%	46.8	53.1	50.7	41.5	47.6
STAT2	n%	49.7	56.6	54.0	45.6	50.4
STACKa	n%	45.0	50.1	49.3	40.1	45.7
STACKb	n%	51.8	49.9	60.1	38.7	53.8
BAT0	n%	43.5	49.1	47.4	36.8	44.6
BAT1	n%	44.9	49.5	49.5	37.5	46.0
BAT2	n%	45.0	49.4	49.6	37.7	46.1
BAT3	n%	44.9	49.0	48.0	37.0	46.2
BAT4	n%	44.7	49.6	47.4	37.9	45.7
BAT5	n%	44.7	49.6	47.3	38.0	45.7
MRU	nm	14361	6389	6899	9960	4401
DMRU	nm	4086	1452	2160	2525	1561
STAT0	nm	0	0	0	0	0
STAT1	nm	740	461	204	612	128
STAT2	nm	125	41	74	74	51
STACKa	nm	1020	722	215	828	192
STACKb	nm	3972	2963	837	3176	796
BAT0	nm	1712	1118	462	1366	346
BAT1	nm	1279	929	242	1071	208
BAT2	nm	1080	763	209	910	170
BAT3	nm	2323	1447	649	1806	517
BAT4	nm	1307	767	365	1006	301
BAT5	nm	1329	765	380	1018	311

I/O traffic can change several times. To counter this unstability STAT2 added the restriction that no file should be moved until there had been 10 changes in locality. This is an effective strategy only for partitioned and large files at Amdahl. For all the other installations and file types the network traffic increases. The restriction of 10 changes in locality was arbitrarily chosen, so the performance can most likely be improved.

Table 4.5c: The table displays the results for the various policies for the trace from Amdahl. Two measurements are displayed, the network traffic (label "n%") and the number of file moves (label "nm"). The network traffic is measured in percent of all I/O traffic to shared files. The results are obtained with one user per processor.

Policy	Measure	All files	Sequential files	Partitioned files	Small (<1 Mbytes) files	Large (>1 Mbytes) files
MRU	n%	865.8	75.5	1049.6	140.5	953.6
DMRU	n%	424.3	45.6	545.1	77.4	466.3
STAT0	n%	53.3	41.5	54.5	39.0	55.0
STAT1	n%	53.4	39.1	57.9	31.5	56.0
STAT2	n%	49.6	40.9	53.6	36.6	51.2
STACKa	n%	48.2	38.6	50.1	33.3	50.0
STACKb	n%	70.3	55.4	68.2	40.4	73.9
BAT0	n%	50.1	37.8	49.7	30.2	52.5
BAT1	n%	52.0	37.9	51.4	31.1	54.5
BAT2	n%	51.1	37.9	51.3	31.0	53.5
BAT3	n%	49.4	37.3	49.6	29.5	51.7
BAT4	n%	49.4	35.9	50.4	31.5	51.5
BAT5	n%	49.6	36.0	50.5	31.8	51.8
MRU	nm	31137	2693	25277	13988	17149
DMRU	nm	13976	816	11731	6071	7905
STAT0	nm	0	0	0	0	0
STAT1	nm	1039	197	737	623	416
STAT2	nm	353	25	291	178	175
STACKa	nm	1726	477	919	1241	485
STACKb	nm	4692	1578	2424	3411	1281
BAT0	nm	1350	357	656	947	403
BAT1	nm	1038	282	501	708	330
BAT2	nm	883	264	436	627	256
BAT3	nm	2031	556	1057	1465	566
BAT4	nm	1364	255	798	934	430
BAT5	nm	1431	284	837	983	448

4.6.1.4. Results for Stack Algorithms

Two stack algorithms were analyzed; one used the fraction accessed per user/file pair (STACKa), while the other used the fraction accessed per file (STACKb). The former has the best performance, except for sequential files at SLAC and Hughes where the performances were approximately equal. This illustrates the importance of differentiating between the users. The fraction accessed by one user is not a good predictor for the fraction other users will access from the same file. STACKa is therefore the preferred stack algorithm.

It has a better performance than the previous STAT1 algorithm, although the difference is less than 10% (or equivalent to 5.3% of all I/O traffic to shared files). The difference was even smaller for sequential files, where STAT1 actually has a better performance for the trace from SLAC (equivalent to 1.2% of the I/O traffic to shared files). This was unexpected, since STAT1 only tries to place the files at the best static location.

4.6.1.5. Batch Poisson Algorithms

These algorithms had the best overall performance. The reduction in network traffic compared to static placement (STAT0) is equivalent to from 4 % to 25% of all I/O traffic to shared files. However as expected, the performance is considerably worse than the optimal look-ahead performance. The difference is equivalent to from 13% to 24% of the I/O traffic to shared files. The batch Poisson algorithms only have a better performance than the optimal static look-ahead placement for the trace from SLAC. The quality of the algorithms is discussed further later in the chapter.

The various batch Poisson algorithms have very similar network traffic although they estimate the parameters differently. For two of the installations, SLAC and Hughes, BAT0 has the best performance, while BAT3 has the best at Amdahl. BAT0 assumes all users issue batches of opens at the same rate, while BAT3 ignores the rate of issuing batches of opens. The criterion for moving a file away from node i to node j is move if $E(CB_j)-(1+p'_{ji})F > 0$. $E(CB_j)$ is the expected number of bytes transferred per run of consecutive opens from node j. F is the file size, and p'_{ji} is the probability node i will reference the file next. BAT3 assumes $p'_{ji} = 0.0$. The algorithms BAT0, BAT1, BAT2, and BAT3 use the same estimate for $E(CB_j)$, but different estimates of p'_{ji}. BAT0 assumes all users have the same reference rate, and $p'_{ji} = \frac{1}{n-1}$, where n is the number of users sharing the file. BAT1 uses a dynamic estimate based on the number of file moves; while BAT2 fixes p'_{ji} to 1, and BAT3 fixes p'_{ji} to 0. Regardless of the value chosen, the network traffic remains nearly constant ($\Delta \approx$ 1% to 3%, see Table 5.3). The estimate for p'_{ji} therefore has little impact on the network traffic of the various algorithms. Simply fixing it to 1.0 seems to be a reasonable solution that provides a low network traffic with the fewest possible number of file moves.

This simplifies the implementation of the algorithm. The estimate for p'_{ji} depends on either the number of nodes sharing the file (BAT0) or the number of file moves so far (BAT1). This implies that the node making the migration decision must have knowledge of either the total reference pattern to the file or all previous migration decisions. Although such information easily can be kept at the node that handles the concurrency control to the file, the algorithm is simplified if p'_{ji} is fixed to a constant. The algorithm is then completely decentralized. Each node can make its own migration decision based on only its own reference pattern to the file (the average fraction accessed and the average number of consecutive opens from the node), and any exchange or coordination of information between the nodes is avoided.

The network traffic displays a similar insensitivity to the estimate for $E(CB_j)$. The algorithms BAT3, BAT4, and BAT5 estimate $E(CB_j)$ differently, but they all set $p'_{ji} = 0.0$. Although the estimates for $E(CB_j)$ converge, they differ during the first few runs observed. The migration decisions will therefore be different for some user/file pairs. The three algorithms have nearly identical performance (see table 5.3), and the largest difference is a 3% difference for sequential files at Amdahl. The network traffic is therefore insensitive to the estimation method for $E(CB_j)$.

However, this does not mean that the migration decisions are insensitive to how the parameters are estimated. The number of file moves vary between the various algorithms implying different migration decisions. The two extreme points are BAT2 and BAT3, where the difference in number of file moves ranges from 87% to 130%. The difference in network traffic is at most 1.4%. The additional moves beyond those made by BAT2 must therefore increase the migration traffic by as much as they save in added local I/O. The network traffic is therefore insensitive to whether these file moves are made or not.

4.6.2. The Batch Poisson Model Compared to a Stack Model

The two models can be compared by the difference in performance between the algorithms based on them. The comparison is made between BAT0 and STACKa, since these are the algorithms with the best overall performance for each model. (For a description of the two models we refer to the previous chapter or the previous section.)

The results are inconclusive. The difference between STACKa and BAT0 is 25% at SLAC, -4% at Amdahl, and 0.2% at Hughes. The difference at Amdahl is due to the handling of large index sequential and direct files. For sequential and partitioned files at Amdahl, BAT0's performance is 2% and 4% lower than the one for STACKa. To summarize, algorithms based on a batch Poisson model have an equal or better performance than algorithms based on a stack model. This supports our claim in the previous chapter that the batch Poisson model with geometric batch size is a better abstraction of how users reference files than a stack model.

4.6.3. The Results Compared to Other Simulation Studies

Porcar used two of the same traces, SLAC and Hughes, for a trace driven simulation in a cluster with 2, 4, and 8 nodes [Porcar82]. He compared 4 single copy algorithms: MRU, STAT0 (static allocation), the static lower bound, and an algorithm based on a two state Markov model (DYNOPT). DYNOPT uses two states to model references to a file; state 1 is the first node to access the file; state 2 represents all other nodes. A file is moved from state i (i=1,2) to state j (j=2,1) if $B \cdot \left\lfloor \frac{1}{1-p_{jj}} \right\rfloor > 2F$ [2],

where

B = expected number of bytes transferred per open

F = expected file size

p_{jj} = probability next access will be from state j given the previous access also came from state j.

Until 10 transitions are observed, the policy operates as MRU, which is also used for transitions between nodes in state 2. p_{11} and p_{22} are the unweighted average of the 5 last transitions.

Porcar simulated clusters with 2, 4, and 8 nodes. His main consideration for allocating users to a node was to reduce the number of changes in locality for the files and to increase the length of each locality phase. The algorithm based on the two state Markov model had the best performance [Porcar82]. For SLAC, DYNOPT's performance was close to the ones for MRU and STAT0 [Porcar82]. For Hughes, MRU's performance was

[2] This is an adaptation of Porcar's algorithm. His original version also includes a term for storage cost. Since we assume the storage cost is the same, the terms can be ignored.

substantially worse (more than 2 times the network traffic of the others). This was mostly due to the movement of a few large files. [Porcar82].

Even though we did not simulate Porcar's DYNOPT algorithm, we are able to obtain a good estimate of its performance. It is dominated by the algorithms based on a batch Poisson model. In DYNOPT, $B \cdot \frac{1}{1-p_{jj}}$, which is the estimate of the number of bytes transferred per sequence of consecutive opens, is approximately the estimate for the number of bytes transferred per batch in algorithms BAT3. Porcar used the unweighted average of the 5 last transitions, while the BAT3 uses a Bayesian estimate. DYNOPT collapses all nodes except the first one into one super-node, while BAT3 estimates all parameters for each user/file pair. The former will therefore overestimate the benefits of moving a file away from the first node. Porcar used $2 \times F$ as the cost of moving, which is the same value used by BAT3. As we have already seen, the network traffic is insensitive whether the Bayesian or the average estimation method was used, so the difference in estimation method between BAT3 and DYNOPT should therefore not affect the comparison.

In the case where only two nodes share a file, the two policies should have the same performance. However, DYNOPT uses the policy MRU while the parameters are estimated (the 10 first transitions) and for the transitions between the nodes in state 2. DYNOPT's network traffic will therefore be a mixture of the network traffic for MRU and BAT3, and it will be substantially worse than the one for BAT3 since MRU has such a high network traffic.

As previously mentioned, the ranking of the algorithms also implies a ranking of the models for how files are referenced. The batch Poisson model is therefore a more suitable model of how files are referenced compared to a two state Markov model, since the algorithms based on this model have a better performance.

4.6.4. Detailed Analysis of the Batch Poisson Algorithms

The algorithms based on a batch Poison model estimate several parameters for each node that references a file, and a large amount of statistics must be collected and maintained for the algorithms to function. Most likely, part of this information is unnecessary; in some cases the node will never reference the file again, or the accuracy of the estimates is of a higher quality than what is needed. In this section we establish the relationship between the amount of reference history that is collected and the performance of the algorithms.

A modified version of the BAT1 algorithm was used to establish this relationship. The various statistics were were collected and stored in a fixed size stack, where each element contained all statistics for one location. The statics are only kept for the n last nodes to access the file, and whenever the stack overflows, the statistics for the n+1st last node are erased. With stack size 1, all statistics are erased whenever a different node accesses the file, and only statistics for the current node are kept. With stack size equal to ∞, we keep the statistics for all nodes referencing the file. A fixed stack size limits the number of nodes that statistics are collected for, and the database of a file's reference history will have a limited size. It is then easier to implement the algorithms. The relationship between amount of reference history collected and the performance is important for the implementation of the algorithms, since algorithms with fixed stack size are easier to implement.

Keeping statistics from previous runs (sets of consecutive opens) from a particular node has two advantages:

1) All estimates needed for the migration decision will already exist for nodes that already have referenced the file. The migration decision can therefore be made at the first new reference from the node. Otherwise, some references would have to be remote until the various parameters could be estimated.
2) The accuracy of the estimates increases with the stack size. A large stack will contain statistics based on all previous references, while with a small stack the estimates will are based on only a part of the references.

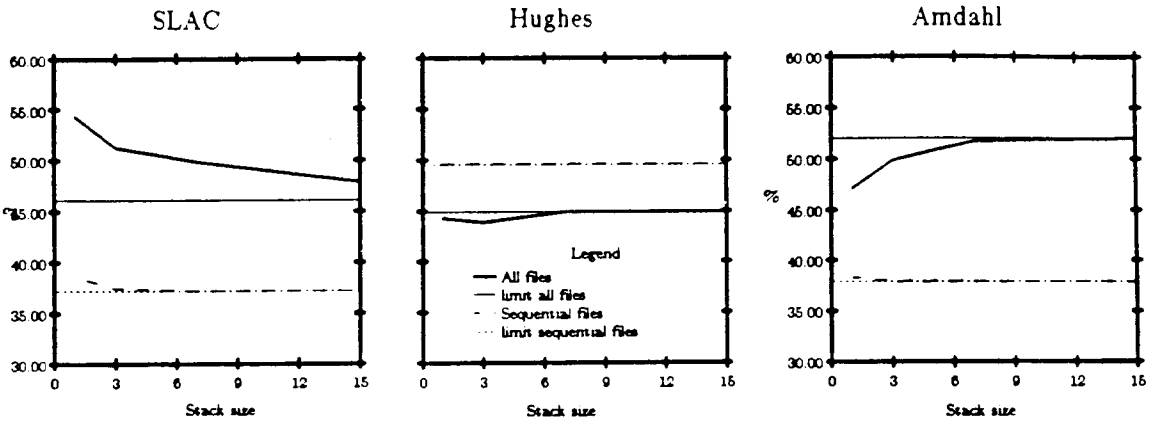


Figure 4.1a: The figure displays the network traffic (measured in % of I/O traffic to shared files) as a function of the number of nodes the various parameters are estimated for. The measurements and the procedure are explained in the text. The measurements are made with one user per node. The curves labeled "limit ..." display the network traffic with stack size = ∞.

BAT1's performance with different stack sizes is displayed in Figure 4.1. The results are inconclusive. Only at SLAC will keeping any history reduce the network traffic significantly, and most of the gain is from stack size 1 to size 3. At the two other installations, the network traffic is constant or increases. The pattern is similar for the number of file moves. The trace from SLAC is the only one where the number of file moves increases with the stack size.

The first time a node references a file, all algorithms must estimate the node's reference characteristics before a migration decision is made, and the reference is made as a remote one. By keeping reference history, such remote traffic can be avoided, but obviously, it can only be of help if the node references the file again later. SLAC has the largest average number of file moves per file for the three installations. The mean number of file moves per file is 5.8 for the SLAC trace, 0.4 for the Amdahl trace, and 0.2 for the Hughes trace. The numbers are similar for sequential files. Intuitively, one would expect the advantage for a bigger stack to be lager for SLAC, since files in this trace are moved more often. Figures 4.1a and b also imply that for Hughes there must be fewer repeated runs of opens (consecutive opens from the same node) from nodes the file should be migrated to, since both the network traffic and the number of moves remain nearly constant with the stack size. Otherwise, one or both of them should be affected by the stack

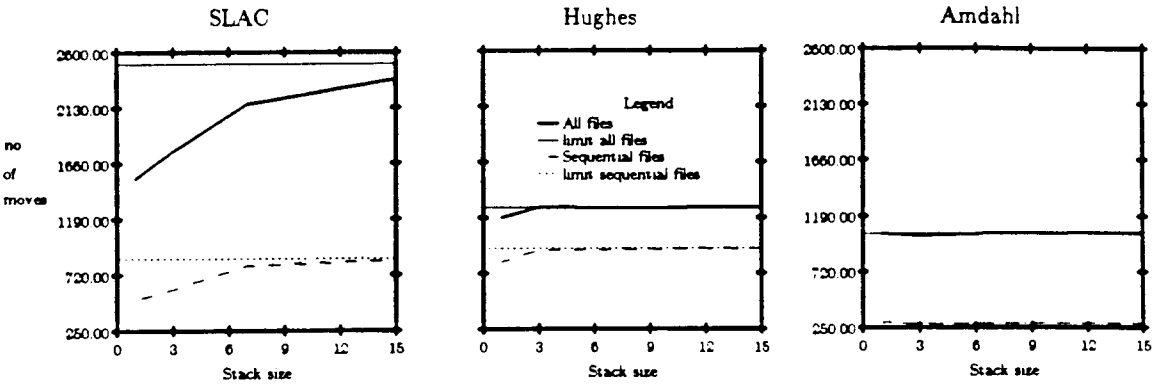


Figure 4.1b: The figure displays the number of file moves made by the BAT policy as a function of the number of nodes the various parameters are estimated for. The measurements and the procedure are explained in the text. The measurements are made with one user per node. The curves labeled "limit ..." display the number of file moves with stack size = ∞.

size. To summarize, a limited stack for parameter estimates will reduce the cost of implementing the policy, while it has a small impact on the performance.

For analyzing the effect the accuracy of the estimate of $E(CB_j)$ has on the results, a different approach was needed. To remind the reader, files are migrated to node j from node i when $E(CB_j) > (1+p'_{ji})$. Varying the stack size affects both the estimate for $E(CB_j)$ and for p'_{ji}. The estimate for p'_{ji} depends on the number of file moves observed, and it is a direct function of the stack size and $E(CB_j)$. The effect the stack size has on the estimate for $E(CB_j)$ may be amplified or counteracted by the additional changes to p'_{ji}. To analyze the effect for $E(CB_j)$ only, BAT2 must be compared to BAT1 with stack size 1. Both use the same estimate method for $E(CB_j)$, but the estimate for p'_{ji} is set to 1.0. The difference between them is a direct measure the advantage a more accurate estimate fore $E(CB_j)$ will have on the performance. The results are displayed in Table 4.6.

The results are comparable to the results in Figure 4.1. For SLAC, BAT2 has a better performance, which is the expected behavior. By keeping history of previous references a migration decision can be made immediately, and some of the remote opens while $E(CB_j)$ is estimated are avoided.

A different effect is seen at the two other installations. The algorithm with the stack size 1 has a lower or equal network traffic, and more important, the number of file moves is higher. This means that for some of the runs $E(CB_j)$ is higher with stack size 1 than when the whole reference history is considered. To move the files for these runs also lowers the network traffic. Such a behavior will typically occur when there are several short runs combined with several longer ones, or a mixture of geometric distributions with a large difference between their means. As we will show, for such a distribution, it is not optimal to base the migration decision on the average of previous set of consecutive opens. Instead the decision should be based on the properties of the current set of consecutive opens.

Let us assume the run size has a distribution which is a mixture of two geometric distributions, one with mean $\frac{1}{1-r_1}$, and one with mean $\frac{1}{1-r_2}$, where $r_2 > r_1$. With probability p the run size will be drawn from the first geometric distribution. The condition for moving the file is $p \times \frac{1}{1-r_1} + (1-p) \times \frac{1}{1-r_2} > (1+\gamma) \times f$, where $f = \frac{B}{F}$ is the mean fraction accessed per open and γ is a penalty parameter for a file move. Regardless whether the gain of moving is positive or not, it may be beneficial to wait with the migration decision until at least one more consecutive open is observed. If it is likely the next reference will come from a different node, it is possible to avoid an unnecessary file move. The condition for waiting for one more open is then:

$$\left[p \times (1-r_1) + (1-p) \times (1-r_2)\right] \times (1+\gamma) F > B$$

or

$$p \times (1-r_1) + (1-p) \times (1-r_2) > \frac{1}{(1+\gamma) f}$$

As one more consecutive open is observed, the probability the run size is drawn from the first geometric distribution is $\hat{p} = \frac{p \cdot r_1}{p r_1 + (1-p) r_2}$. After a new consecutive open has been observed, the decision to wait must then be reevaluated with the new value $\hat{p}$ replacing p in the calculation. Provided the second distribution has a large enough mean, the file will eventually be migrated.

To illustrate the argument, we use the extreme situation where the number of adjacent references from a node has a distribution which is an even mixture ($p=0.5$) of two geometric distributions, one with a small mean (about 1) and one with a large mean (>10). The expected gain of always migration is then positive if $f > 0.4$, but an algorithm which waits with the migration decision until two consecutive reference has been observed, will always have a better performance than an algorithm which makes the decision after the first reference. By waiting for two consecutive references, it likely that the remaining number of adjacent reference will be large (that the number is drawn from the second distribution). By this strategy, the algorithms avoids migrating the file when there is an even chance the number of adjacent references will be small (about 1). This is done at the cost of only one remote open.

To base the decision on the average of the previous runs can therefore result in non-optimal decisions. With stack size 1 all migration decisions are based on the current set of adjacent references. The algorithm is therefore well suited for run size distributions which are mixtures of geometric distributions. This may explain why the network traffic increases for sequential and partitioned files at Amdahl as we retain more reference history, i.e., a large stack size.

4.6.5. An Analysis of the Prior Distribution for the Bayesian Approach

The Bayesian approach uses an initial belief (a prior distribution) as the starting point for the estimate. As the system is observed, the prior distribution is updated. This section analyzes the relationship between the prior distribution and the performance of the algorithms. With the prior distributions we have used, the batch Poisson algorithms are optimistic algorithms that are more inclined to migrate a file compared to the ones that use the estimate based on the average of observed values. This effect is most evident during the first few opens from a file, and, as we will discuss later in the chapter, it is a desired feature. Let F be the file size, R be the number of references, B be the I/O traffic

Table 4.6: The table displays the performance for the different Bayesian estimation schemes. The two performance measures are the network traffic (label "n%") and the number of file moves (label "nm"). The results are obtained with one user per processor.

Policy	measure-ment	SLAC		Hughes		Amdahl	
		All files	Sequential files	All files	Sequential files	All files	Sequential files
BAT2	n% nm	46.7 1802	37.5 583	45.0 1080	49.4 763	51.1 883	37.9 264
With stack size 1							
α_0=1 β_0=2	n% nm	54.3 1533	38.6 513	44.3 1186	49.6 810	47.1 1042	38.3 294
α_0=2 β_0=2	n% nm	52.8 1728	38.5 534	44.5 1367	49.3 900	46.7 1227	38.2 353
α_0=0 β_0=1	n% nm	60.2 628	40.3 196	46.5 409	52.1 271	49.0 485	36.1 97

so far from the current node, and C be the number of changes in locality, the migration criterion is: move if $\frac{B}{R} \times \frac{R+\alpha_0+\beta_0-1}{C+\beta_0-1} > (1+p_i) \times F$. The factor $\frac{R+\alpha_0+\beta_0-1}{C+\beta_0-1}$ is due to the Bayesian approach, and it will only differ from the estimate based on the average when R and C are small. We assume that for each node/file pair j the number of consecutive references has a geometric distribution with mean $\frac{1}{1-r_j}$, where r_j is unique for each node/file pair. We therefore associate a distribution function with r_j. As an initial guess we believe r_j has a Beta distribution with density function

$$\frac{\Gamma(\alpha_0+\beta_0)}{\Gamma(\alpha_0)\Gamma(\beta_0)} \times r_j^{\alpha_0-1}(1-r_j)^{\beta_0-1}$$

As we observe the reference patterns, our belief in the shape of the distribution function for r_j changes. The distribution will still be a Beta distribution, but with different parameters α and β. For each additional consecutive reference, α is increased by one, and for each change in locality, β is increased by one. The effect of the prior distribution is therefore largest during the first few references, and the algorithms with stack size 1 is used for the analysis. Each time there is a change in locality, the estimates start all over, so the effect of the prior distribution is amplified. For comparison, the performance with stack size ∞ is also included.

Three different prior distributions are analyzed:
1) $\alpha_0 = 1$ and $\beta_0 = 2$ which corresponds to an initial expectation that there will be two consecutive opens from the same node. The density function for r_j is then $2 \times (1-r_j)$ and it is shown in Figure 4.2a. The values chosen therefore corresponds to a belief that most nodes will only have few consecutive references, with 25% of the nodes with an average

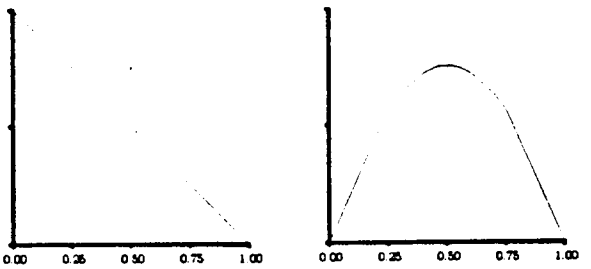
larger than 2 and with only 6% with and average larger than 4. This is a reasonable assumption compared to the empirical distributions in Table 4.7, which display the empirical distributions for user/file pairs with more than 6 sets of consecutive opens. These are the most active user/file pairs. We do not have the distribution for all user/file pairs, but the median number of opens per user is less than 4 opens, so the initial prior distribution seems to be appropriate.

Table 4.7: The table displays the empirical distribution for the number of consecutive opens per user/file pair. Only user/file pair with 6 or more sets of consecutive opens are included.

Average no of consecutive opens	SLAC	Hughes	Amdahl
1-2	58.3%	61.4%	39.3%
2-4	26.3%	24.2%	30.3%
4-	15.5%	14.3%	30.5%

2) $\alpha_0 = 2$ and $\beta_0 = 2$ which corresponds to an initial expectation that there will be three consecutive opens from the same node. The density function, $f(r_j)$, is now $6 \times r_j(1-r_j)$, and it is shown in Figure 4.2b. This set of values reflects the belief that more than 50% of the nodes will have mean run sizes larger than 2, but that only $\frac{5}{32} \approx 15\%$ will have a mean larger than 4. This prior distribution is closer to the empirical distributions for user with more than 6 sets of consecutive opens found in Table 4.7.
3) $\alpha_0 = 0$ and $\beta_0 = 1$ does not correspond to a prior distribution, since the Beta distribution is only defined for α and β larger than 0. With these values, the estimate for the number of consecutive opens is equal to the number observed so far, which is the estimator based on a renewal approach.

The results are summarized in Table 4.6. The Bayesian approach is clearly beneficial compared to the renewal approach ($\alpha_0 = 0$, $\beta_0 = 1$). This is under the assumption that no previous history is kept, so the benefit of using a prior distribution is amplified. Sequential files at Amdahl is the exception, where the amount of network traffic is reduced with an estimator based on the renewal approach.



α_0=1 β_0=2 Figure 4.2a

α_0=2 β_0=2 Figure 4.2b

The Bayesian approach is therefore beneficial, since the locations a node should be moved to can be identified "faster". Some remote I/Os are avoided, but at the cost of doing some unnecessary file moves. For all installation the net effect is a reduction of the network traffic.

The default prior distribution has $\alpha_0 = 1$ and $\beta_0 = 2$, which corresponds to the belief that the expected number of consecutive opens is two per node. If we increase α_0 to 2, the initial expectation for the run size is changed to three consecutive opens. Under this setting, we believe that more than 50% of the nodes will have mean run sizes larger than 2, but that only $\frac{5}{32}$ will have a mean larger than 4. This is a prior distribution that is closer to the empirical distributions for users with more than 6 sets of consecutive opens. This prior distribution seems to be more appropriate, since the network traffic is improved at both SLAC and Amdahl, while it remains constant for Hughes. In hindsight, the improvement in performance should have been expected, since the benefit of migration is largest for nodes that repeatedly reference files with several changes in locality. The prior distribution should therefore reflect the distribution found among these nodes and not the distribution for the "average" node.

To conclude, the Bayesian approach results in a lower network traffic than the estimator based on the renewal approach, where the expected number of remaining consecutive opens is equal to the number observed so far. This is of special importance if no reference history is retained each time there is a change in locality. The choice of the initial prior distribution affects the result, and it seems that the tail behavior is an important feature.

4.6.6. The Stack Algorithms Revisited

The batch Poisson algorithms can be interpreted as algorithms based on the fraction accessed per open. They migrate a file if $E(CB_j) > (1+\gamma) \times F$, where $E(CB_j)$ is the expected amount of I/O from node j before a different node will reference the file, F is the file size, and γ is a parameter between 0 and 1. For BAT1, $E(CB_j)$ is $B \times \frac{R+2}{R \times (C+1)}$, where B is the I/O traffic observed so far, R is the number of references so far from the current node, and C is the number of changes in locality. The migration condition can

then be written as:

$$\frac{B}{R \times F} > \frac{(1+\gamma) \times (C+1)}{R+2},$$

where $\frac{B}{R \times F}$ is the mean fraction accessed per open from the current node. The algorithms can therefore be viewed as using the average fraction accessed per open by a user as the decision variable: if the fraction is larger than a given limit, the file is migrated to the node the user is on. However, the limit for migration varies with the reference pattern for each user/file pair. For the algorithms with stack size 1, γ is equal to 1, and $C = 0$. The migration condition for the algorithm BAT1 with stack size 1 is then:

$\alpha_0 = 1\ \beta_0 = 2$ migrate if the average fraction accessed $> \frac{2}{R+2}$

$\alpha_0 = 0\ \beta_0 = 1$ migrate if the average fraction accessed $> \frac{2}{R}$

The migration criterion will therefore decrease as the number of consecutive references from the current node increases. One would expect that the two versions of BAT1 with stack size 1 would have a lower network traffic than STACKa; they use the average fraction accessed since the last change in locality, while STACKa uses the largest fraction accessed. (STACKa moves whenever the fraction accessed is larger than *lim*.) Their migration criterion is also decreasing as the number of consecutive references is increasing, while STACKa uses a constant criterion. STACKa will therefore never move a file to users with many consecutive references that transferred a small fraction of the file per open. The conjecture is only plausible as long as the distribution for the number of consecutive opens has a constant or decreasing hazard rate, like a geometric distribution. Otherwise, the decreasing migration criterion would have a tendency to migrate files that were likely to be referenced by a different node in the near future.

The algorithm STACKa is dominated by BAT1 with stack size 1, but only for α_0=1 *or* 2 and β_0=2. This is true even for sequential files, although the differences are smaller. With α_0=0 and β_0=1, the BAT1 algorithm waits too long before it migrates a file, and its performance is worse than the STACKa algorithm. As an example, a file will be migrated to a user that accesses a fraction equal to *lim* (the criterion for the stack algorithm) after one reference by the STACKa algorithm, while it will take 4 references for SLAC, 3 for Hughes, and 7 for Amdahl. The corresponding values for *lim* are given in section 4.4. This is the most likely explanation of the relative poor performance of the BAT1 algorithm with α_0=0 and β_0=1.

To summarize, for algorithms based on the fraction accessed, a dynamic criterion is a better decision criterion compared to a static one. However, the migration criterion must not be so high that many consecutive opens must be observed before the file is migrated. The cost of the remote opens will then make the algorithm ineffective. This section illustrates how the batch Poisson algorithms can be viewed as algorithms based on the average fraction accessed, but with a dynamic migration criterion.

4.6.7. The Effect a Cost for Updating the File Directory will have on the Results

We have based our analysis on a zero cost of updating the file directory each time there is a file migration. Our results, at least for class C algorithms, will not be affected as long as the cost of updating the directory is small compared to the cost of migrating the file. If we assume the cost of updating the directory is U units of network traffic, the migration criterion will be, move when $E(CB_j) > (1+\gamma)\times(F+U)$. The term $(1+\gamma)U$ reflects the cost of updating the file directory. This condition can be rewritten as: move when $E(CB_j) > (1+\gamma+\frac{U}{F}\times(1+\gamma))\times F$. Section 4.6.1.5 analyzed the performance with $U = 0$ and γ ranging from 0 to 1. The network traffic remained nearly constant regardless of the value for γ. With $U \neq 0$, the network traffic will be similar [3], as long as $\gamma+\frac{U}{F}\times(1+\gamma)$ is less than 1, or $\frac{U}{F} < 1$ and $\gamma = 0$. The migration condition will then be within the ranges we already have analyzed in Section 4.6.1.5. In addition, the cost of updating the file directory will only affect the migration decision for small files, and the total network traffic is insensitive to whether small files are migrated or not (see Table 4.5).

4.6.8. The Results with Users from the Same Department Located to the Same Node

The simulation was repeated with all users from the same department allocated to the same node, and the performance of the various algorithms with this user allocation is displayed in Table 4.8. This allocation rule is an example of the performance that can be obtained by using some knowledge about the intended users. The mapping is practical, it does not require any measurements, and it is easy to administer. This allocation also provides an important measure of the robustness of the proposed algorithms. The reference patterns to the files have been analyzed only on a per user basis. When several users are placed at the same node, their reference processes are superpositioned to create a new process, which has other properties than the one for the individual processes. The most appropriate model is still a batch Poisson model, but the batch size should be modeled by a distribution which is a mixture of geometric distributions.

All the policies are based on the properties found on a per user basis, and they are not adapted to the reference pattern on a per department basis. The performance under this user allocation is therefore a measure of the robustness of the algorithms, since the reference pattern will be different from those for which the algorithms were designed.

4.6.8.1. Results

The results are similar to the ones obtained when each user was placed at a separate node. The network traffic is substantially reduced, but the relative difference between the algorithms remains nearly the same. However, the absolute difference between some of the algorithms is reduced, and the smallest difference is found for sequential files at SLAC, where the difference in network traffic between the best and the worst algorithm is 34%. Our results therefore seem to be fairly robust to changes in the reference patterns.

The results for the MRU algorithms are virtually unchanged. To always migrate a file when there is a change in locality (MRU) still has the worst performance. If the

[3] The network traffic will be equal to the nearly constant network traffic for the batch Poisson algorithms plus the additional update cost for the directory.

Table 4.8a: The table displays the results for the various policies for the trace from SLAC. Two measurements are displayed, the network traffic (label "n%") and the number of file moves (label "nm"). The network traffic for a particular group of shared files (like sequential files) is measured in percent of all I/O traffic to that group of shared files, and it is obtained with all users from the same department allocated to the same node.

Policy	Measure	All files	Sequential files	Partitioned files	Small (<1 Mbytes) files	Large (>1 Mbytes) files
MRU	n%	65.7	5.1	71.5	78.9	63.0
DMRU	n%	34.1	4.7	36.8	27.6	35.4
STAT0	n%	29.4	5.0	31.8	25.0	30.4
STAT1	n%	24.6	4.6	26.5	19.5	25.6
STAT2	n%	25.5	4.8	27.5	21.8	26.3
STACKa	n%	23.1	4.1	24.9	17.8	24.2
STACKb	n%	36.0	3.8	39.1	20.9	39.1
BAT0	n%	19.3	4.0	20.8	17.0	19.8
BAT1	n%	19.8	3.8	21.3	17.8	20.2
BAT2	n%	20.0	3.9	21.5	17.9	20.4
BAT3	n%	19.2	3.9	20.7	17.2	19.7
BAT4	n%	19.8	4.0	21.3	18.2	20.1
BAT5	n%	20.5	4.0	22.1	17.8	21.1
MRU	nm	6816	787	5996	4547	2269
DMRU	nm	2029	364	1645	1322	707
STAT0	nm	0	0	0	0	0
STAT1	nm	114	28	85	73	41
STAT2	nm	57	11	46	35	22
STACKa	nm	576	234	324	465	111
STACKb	nm	1602	583	992	1167	435
BAT0	nm	1713	545	1142	1219	494
BAT1	nm	1603	515	1062	1148	455
BAT2	nm	1147	415	714	829	318
BAT3	nm	1979	571	1381	1354	625
BAT4	nm	1719	479	1216	1159	560
BAT5	nm	1662	485	1153	1116	546

migration is restricted only to nodes with two consecutive opens (DMRU), the performance increases, as it did with one user per node. The relative performance of static placement has improved, and it has become a reasonable placement rule. For the trace from Amdahl, it has nearly the best performance. This is more a result of the "bad" performance of the other algorithms for this installation, since there is a difference between the optimal dynamic and static look-ahead algorithms.

The algorithms (STAT1 and STAT2), which try to place the file at the best static placement, have a reasonable performance. The relationship between them is the same; at SLAC, STAT1 has the lowest network traffic, while STAT2 has the lowest (of the two) at Amdahl. To remind the reader, STAT1 moves the file to the node with the largest

cumulative I/O traffic to the file, while STAT2 waits until there has been 10 changes in locality before it starts moving the file. The node with the largest cumulative I/O traffic to the file must therefore still change quite frequently at Amdahl.

Table 4.8b: The table displays the results for the various policies for the trace from Amdahl. Two measurements are displayed, the network traffic (label "n%") and the number of file moves (label "nm"). The network traffic for a particular group of shared files (like sequential files) is measured in percent of all I/O traffic to that group of shared files, and it is obtained with all users from the same department allocated to the same node.

Policy	Measure	All files	Sequential files	Partitioned files	Small (<1 Mbytes) files	Large (>1 Mbytes) files
MRU	n%	494.5	31.8	691.6	54.5	547.8
DMRU	n%	250.1	18.7	345.8	30.2	276.7
STAT0	n%	27.3	16.1	27.6	17.0	28.5
STAT1	n%	28.4	16.2	26.1	14.0	30.2
STAT2	n%	24.4	16.0	23.4	15.8	25.5
STACKa	n%	26.0	13.1	26.1	13.2	27.5
STACKb	n%	34.9	22.0	30.9	16.6	37.1
BAT0	n%	28.2	11.8	27.1	12.9	30.1
BAT1	n%	29.3	11.9	27.2	12.9	31.3
BAT2	n%	28.9	11.9	27.2	12.8	30.8
BAT3	n%	27.6	12.6	25.7	12.6	29.4
BAT4	n%	27.6	11.8	26.1	12.8	29.4
BAT5	n%	27.7	11.8	26.1	12.8	29.4
MRU	nm	14598	958	12462	6090	8508
DMRU	nm	6211	349	5206	2611	3600
STAT0	nm	0	0	0	0	0
STAT1	nm	403	62	297	233	170
STAT2	nm	106	5	85	56	50
STACKa	nm	742	181	357	531	211
STACKb	nm	1977	498	1114	1456	521
BAT0	nm	641	150	267	449	192
BAT1	nm	542	143	209	369	173
BAT2	nm	411	112	182	276	135
BAT3	nm	1001	224	486	697	304
BAT4	nm	727	129	355	495	232
BAT5	nm	684	135	329	448	236

For the other algorithms, the performance is similar to the one observed with one user per node. The network traffic for the batch Poisson algorithms remains insensitive to the estimation method, although the number of migrations varies significantly. The largest difference in network traffic is 5%, and the lowest network traffic is obtained with p'_{ji}=0, which is the probability the file will return to its current location i after node j has finished referencing the file. This indicates that either there are few repeated set of consecutive opens or they are far apart.

Table 4.9: The table displays the performance for the different Bayesian estimation schemes. Two measurements are displayed, the network traffic (label "n%") and the number of file moves (label "nm"). The network traffic is measured in percent of all I/O traffic to shared files, and it is obtained with all users from the same department allocated to the same node.

Site	Policy	Measure	All files	Sequential files	Partitioned files	Small (<1 Mbytes) files	Large (>1 Mbytes) files
Slac	BAT2	n%	20.0	3.9	21.5	17.9	20.4
	BAT2	nm	1147	415	714	829	318
	BAT1 with stack size 1						
	$\alpha_0=1\beta_0=2$	n%	22.7	4.2	24.5	17.7	23.8
		nm	697	260	420	536	161
	$\alpha_0=2\beta_0=2$	n%	23.3	4.1	25.1	17.8	24.4
		nm	780	268	494	589	191
	$\alpha_0=0\beta_0=1$	n%	24.9	4.8	26.8	18.6	26.2
		nm	305	104	196	232	73
Amdahl	BAT2	n%	28.9	11.9	27.2	12.8	30.8
	BAT2	nm	411	112	182	276	135
	BAT1 with stack size 1						
	$\alpha_0=1\beta_0=2$	n%	25.3	12.0	20.2	12.1	26.9
		nm	470	122	211	316	154
	$\alpha_0=2\beta_0=2$	n%	25.2	12.8	20.2	12.2	26.7
		nm	536	139	254	368	168
	$\alpha_0=0\beta_0=1$	n%	27.1	12.1	20.1	12.3	28.9
		nm	221	44	121	156	65

With one department per node, a batch Poisson model is still appropriate, as long as the distribution for the batch size is a mixture of geometric distributions. If the distributions in the mixture have large difference between their means, the decision criterion should not be based on statistics from the whole reference history. Instead, only statistics from the current set of consecutive opens should be used. This was discussed in more detail in the previous sections. However, the relationship between using the whole reference history for estimating $E(CB_j)$ (BAT2) or only the current set (BAT1 with stack size 1) remains the same; BAT2 has lower network traffic at SLAC, while the opposite is true for Amdahl (see Table 4.9). Since the network traffic decreases with increasing stack size at SLAC, the distribution for the batch size must be a mixture of geometric distributions with relatively similar means. Otherwise, the network traffic should have increased with increasing stack size (see section 4.6.4).

The relationship between the performance and stack size is similar to the one with one user per node; the network traffic decreases with increasing stack size at SLAC, while it increases at Amdahl. With one department per node, the number of nodes sharing a file is smaller. The benefit of having larger stacks should then also be smaller. It is

therefore surprising that we need a fairly large stack size, 7, at SLAC before the network traffic approaches the one with stack size ∞.

The various prior distributions have nearly the same performance, and the one with the best performance has $\alpha_0 = 1$ and $\beta_0 = 2$. As previously mentioned, the batch Poisson algorithm with stack size 1 can be viewed as an algorithm that uses the fraction accessed per open as the decision variable. The stack algorithms also use the fraction accessed as the decision variable, but the migration criterion is constant, while the batch Poisson algorithms use a dynamic criterion. The latter should therefore have a better performance, which they also do; For both installations BAT1 with stack size 1 dominates STACKa.

The stack algorithms STACKa and STACKb are no longer the optimal algorithms under a stack model, since we did not measure the stack probability under the department mapping. Instead, the stack probabilities with one user per node were used. The results can therefore not be used to determine whether a stack model is the most appropriate model under the department mapping. The difference between the network traffic for STACKa and STACKb underlines the importance of differentiating between the users; the fraction accessed by one user is not a good predictor of the fraction other users will access.

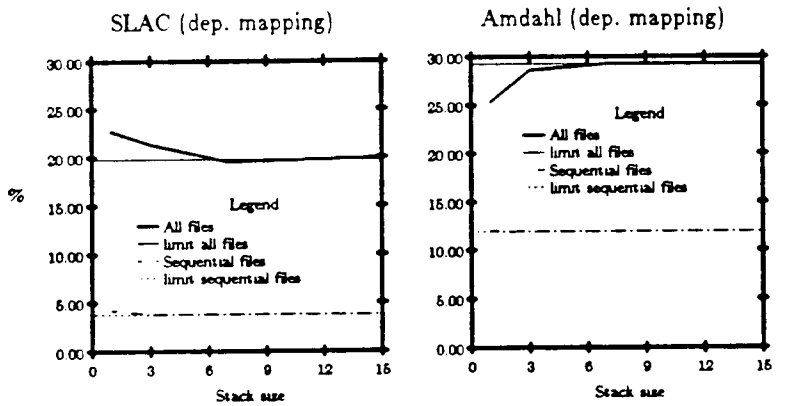


Figure 4.3a: The figure displays the network traffic (measured in % of I/O traffic to shared files) as a function of the number of nodes the various parameters are estimated for. The measurements and the procedure are explained in the text. The results are obtained with all users from the same department allocated to the same node. The curves labeled "limit ..." display the network traffic with stack size = ∞.

Placing users from the same department at the same node does not seem to change the reference pattern in a way that affects the behavior of some of the algorithms. For example, the fraction of changes in locality that results in a file move remains nearly the same for the batch Poisson algorithms. With one department per node the number of changes in locality is reduced by 45% at SLAC and 53% at Amdahl, while the number of file moves for BAT0 is reduced by 44% at SLAC and 52% at Amdahl. However, the reference pattern is changed, since the number of file moves for STAT1, which moves the file to the node with the largest cumulative I/O traffic, is reduced by a larger percentage

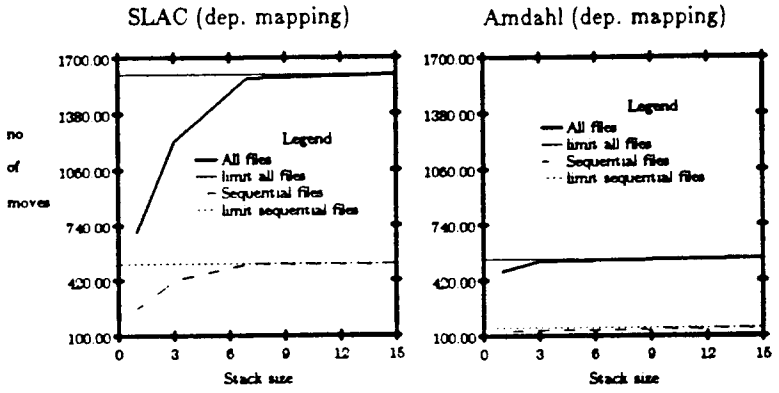


Figure 4.3b: The figure displays the number of file moves made a the BAT policy as a function of the number of nodes the various parameters are estimated for. The measurements and the procedure are explained in the text. The results are obtained with all users from the same department allocated to the same node. The curves labeled "limit ..." display the number of file moves with stack size = ∞.

(67% at SLAC and 61% at Amdahl).

To summarize, the results of the analysis are not changed when users from the same department are placed at the same node. The relative ranking of the various algorithms remains the same, but the difference in network traffic between some of them is negligible. The proposed algorithms are robust to the changes in the reference pattern caused by placing one department per node, and they will most likely provide good performance for other installations.

4.6.9. A Final Note on the Results for Sequential Files

For sequential files opened in overwrite mode, we made the conservative estimate that the potential file transfer cost is equal to the file size. If the file actually was overwritten, the potential transfer cost is 0.0, since the copy at the old location can be discarded and a new version can be created locally. However, the traces do not record the actual mode, just the intended mode of the open. A file can therefore be opened in overwrite mode, even though it can be read or read and updated. Under these conditions, the potential file transfer cost is equal to the file size. We therefore chose a conservative approach and set the potential file transfer cost equal to the file size regardless of the mode.

This conservative assumption does not change the results. Table 4.10 and 4.11 display the results for the optimal look-ahead and various algorithms when the transfer cost is set to 0.0 for sequential files opened in overwrite mode. The change is largest for the trace from SLAC, where the network traffic is drastically reduced. At the two other traces there are only small changes. However, the relative performances of the algorithms remain the same, except for the batch Poisson and the stack algorithms at SLAC. They have nearly the same performance.

Table 4.10: The table displays the performance of the optimal look-ahead placement for sequential files. The results are obtained assuming all opens in overwrite mode erase the file before it is opened.

	one user per node			one department per node	
	SLAC	Hughes	Amdahl	SLAC	Amdahl
Lower Static Bound, the total network traffic in % of I/O traffic to the files	33.05	46.71	33.3	3.95	12.8
Lower Dynamic Bound, the total network traffic in % of I/O traffic to the files	5.95	36.65	26.27	1.67	7.94
% of the files that remains at the same location when the lower dynamic bound is calculated	28.83	78.96	77.85	74.85	90.62
% of changes in locality that results in a file move for the dynamic placement	49.55	25.09	18.6	36.72	16.18
number of file moves	874	1603	501	289	155
average number of file moves per file that is moved	7.5	1.5	2.6	7.0	1.9

As expected, the number of file moves is increased, both for the look-ahead algorithms and the realizable algorithms. Since the cost of a file move is reduced, the difference between the static and dynamic solutions is also larger. This is seen both for the look-ahead and the dynamic algorithms.

To summarize, our results are not affected by the conservative assumption that for sequential files opened in overwrite mode, the file transfer cost is equal to the file size. This assumption only underestimates the advantage of moving. It does not change any of the conclusions.

4.7. The Quality of the Proposed Algorithms

So far only the relative performance of the algorithms has been analyzed. The batch Poisson algorithms have the best overall performance, but these algorithms are based on a simple model, and there may well exist better algorithms. The interesting question is how much better. To measure the quality or the room for improvement we use the difference between the performance of the optimal look-ahead algorithms and the proposed algorithms.

The optimal look-ahead algorithms have full knowledge of the future references, and they will always have a better performance than any realizable algorithm. There are three sources for this difference:
1) The realizable algorithms may be based on a simplified or a completely wrong model of how files are referenced.

Table 4.11: The table displays the performance of the various algorithms for sequential files. The results are obtained assuming all opens in overwrite mode erase the file before it is opened. Two measurements are displayed, the network traffic (label "n%") and the number of file moves (label "nm").

Algo-rithm	Measure	one user per node			one department per node	
		SLAC	Hughes	Amdahl	SLAC	Amdahl
MRU	n%	18.1	57.0	70.0	3.3	30.6
	nm	1764	6389	2693	787	958
DMRU	n%	25.3	51.9	43.6	3.7	18.3
	nm	767	1452	816	364	349
STAT0	n%	51.9	56.4	41.5	5.0	16.1
	nm	0	0	0	0	0
STAT1	n%	36.2	52.4	38.4	4.2	16.1
	nm	103	461	197	28	62
STAT2	n%	50.4	56.6	40.9	4.8	16.0
	nm	48	41	25	11	5
STACKa	n%	9.0	43.7	33.9	2.5	11.9
	nm	890	1096	689	304	230
STACKb	n%	11.1	45.0	50.5	2.3	20.7
	nm	1345	3228	1707	596	539
BAT0	n%	8.5	43.2	33.9	2.5	11.0
	nm	1260	1445	618	578	216
BAT1	n%	8.5	43.5	33.9	2.3	11.0
	nm	1171	1272	560	550	214
BAT2	n%	8.8	43.3	34.0	2.4	11.0
	nm	956	1110	534	450	180
BAT3	n%	9.2	43.2	34.1	2.4	11.8
	nm	1306	1698	774	601	279
BAT4	n%	9.4	43.6	32.0	2.6	11.0
	nm	1125	1160	531	523	198
BAT5	n%	9.4	43.6	32.0	2.5	11.0
	nm	1121	1161	557	529	204
BAT1 Stack size 1	n%	8.7	43.4	34.1	2.6	11.1
	nm	918	1151	540	330	184
α_0=2 β_0=2	n%	8.7	43.2	34.0	2.5	12.0
	nm	929	1214	585	338	198

2) The realizable algorithms may use the correct model, but with incomplete or wrong parameter estimates. The parameters are estimated from the reference history, and they will therefore most likely differ from the "correct" ones.
3) The correct stochastic decisions are necessarily not the correct decisions given full knowledge of future references. For example, let the number of consecutive opens from a node have a geometric distribution with mean 2. Further, let us assume that each open transfers 49.99% of the file and that the cost of a file transfer is equal to the file size. The best stochastic decision is then not to migrate the file, since the expected advantage is smaller than the transfer cost. However, one quarter of the runs (sets of consecutive opens) will contain three or more opens, and a look-ahead algorithm will migrate the file for these runs. In the simplified case used to illustrate our argument, the look-ahead algorithm has a 38% lower network traffic than the best stochastic algorithm. Let the file size be used as a measuring unit. The best stochastic algorithm will handle all opens remotely, and the expected network traffic per run is $0.4999 * 2 \approx 1$. The cost of the optimal look-ahead is $0.4999 \sum_{i=1}^{i=\infty} \frac{i}{2^i} + 1 \sum_{i=3}^{\infty} \frac{1}{2^i} = \frac{5}{8}$.

This section analyzes the difference between the proposed algorithms and the optimal look-ahead algorithms in more detail. Only the batch Poisson algorithms are discussed, since these had the best overall performance. Whenever an algorithm is compared with a look-ahead algorithm, the difference is measured in percent of the performance of the optimal look-ahead algorithm.

The differences between static placement at the first node and the optimal static look-ahead placement are between 27% (at SLAC) and 60% (at Amdahl) (measured in percent of the performance of the optimal look-ahead). The difference is similar for sequential files, 25% (at Hughes) to 56% (at SLAC). Placing the file at the first node to access it is the simplest possible static placement, and the difference to the optimal should be large. The difference is less when users from the same department are placed at the same node (12% at SLAC and 55% at Amdahl).

The difference between the optimal dynamic look-ahead placement and the batch Poisson algorithms is larger than the difference between STAT0 and the optimal static look-ahead. This is not significant, since the difference between a static algorithm and the static look-ahead algorithm is unrelated to the differences between migration algorithms and the optimal dynamic look-ahead placement. The differences between BAT0 and the optimal dynamic look-ahead placement range from 35% (at Hughes) to 100% (at Amdahl). For sequential files, the differences are from 15% (at Hughes) to 51% (at Amdahl). The number of file moves are higher than the number for the optimal look-ahead algorithms. However, this difference does not affect the quality of the algorithms, since other batch Poisson algorithms (BAT2) have a lower number of file moves but with the same or higher difference in performance. Placing users from the same department at the same node increases the difference, so it is 72% at SLAC and 143% at Amdahl. The same numbers for sequential files are 54% and 23%. As previously discussed, this mapping changes the reference patterns on a node level, and the batch Poisson model with geometric batch size is no longer the appropriate model. This is reflected in the larger difference for the algorithms based on the this model. At Amdahl, the algorithm that used no reference history (BAT1 with stack size 1) had a better performance. The difference for this algorithm is 88% when there is one user per node and 118% when users from the same department are placed at the same node. The same numbers for sequential files are 35% and 43%. Except for the results from Hughes, the initial assessment of the quality for the batch Poisson algorithms is poor; there seems to be a large room for improvement. This initial assessment is, however, not correct.

The batch Poisson algorithms move a file from node i to node j when $E(CB_j) > (1+\gamma)F$. $E(CB_j)$ is the estimate of the advantage of the move, while $(1+\gamma)F$ is the estimated cost. Initially $E(CB_j)$ is set to 0.0, so at least the first time node j uses the file, several opens must be handled remotely before the file is migrated (before the estimated benefit is larger than the estimated cost). This procedure was chosen, since the algorithms that always migrated a file when there was a change in locality had a poor performance. The remote opens are "wasted" network traffic for the nodes that the file will be migrated to; had the file been migrated immediately when there was a change in locality the remote opens could have been avoided without any increase in network traffic. A look-ahead algorithm will avoid such "wasted" network traffic, since the algorithm knows when there is a change of locality whether it is beneficial to migrate the file or not. The migrations are therefore typically done when there is a change in locality. It is necessary to prefix the explanation with "typically", since artificial examples can be constructed where it is optimal to move the file first after several remote opens. For example, if the file size is reduced in the middle of a set of consecutive opens from a node, it may be advantageous to let the first opens be remote so only the smaller version of the file must be migrated. The initial implementation of the look-ahead calculation included a bug that forced migrations only to occur when there was a change in locality. Removing this bug hardly changed the results. The "wasted" network traffic can be viewed as the cost batch Poisson algorithms pay for estimating parameters; until the advantage of migrating can be estimated, the file will not be moved. For a look-ahead algorithm, the benefit is known immediately when there is a change in locality.

This wasted network traffic was measured for each algorithm. Formally, it is defined as the traffic due to consecutive remote opens from nodes the file will be migrated to. A prerequisite is that the file is migrated immediately after the last remote open; if there are any opens from other locations between the remote opens and the migration, the remote traffic is not considered wasted. As an example, under this definition STAT0 has no wasted network traffic.

For BAT0 the "wasted" network traffic is equivalent to 4.3% at SLAC, 4.1% at Hughes, and 6.9% at Amdahl of the total I/O traffic to shared files. The total difference between BAT0 and the look-ahead placement is equivalent to 16.1% (SLAC), 11.3% (Hughes), and 25.1% (Amdahl) of the I/O traffic to shared files. The "wasted" traffic therefore represents more than 26% of the difference between BAT0 and the optimal look-ahead algorithm. With the Bayesian estimation method, files are migrated faster (fewer remote opens), and the "wasted" network traffic is reduced. This effect was partly the motivation behind the approach. To get a true measure of cost of the non-optimal migration decisions made by the BAT0 algorithm, it is necessary to subtract the "wasted" network traffic from the difference in performance, since it represents a cost of estimating various parameters that the look-ahead algorithm avoids.

Subtracting this "wasted" network traffic substantially reduces the difference between the batch Poisson algorithms and the optimal look-ahead algorithm. If BAT1 with stack size 1 is used for the comparison for the trace from Amdahl, the difference range from 22% (at Hughes) to 47% (at Amdahl) with one user per node and 57% and 47% for one department per node. For sequential files the difference is less than 17% regardless of installation and rule for mapping users to nodes. These numbers measure directly the difference due to the migration decisions, and we claim the differences are reasonable.

Previously we argued that any realizable algorithm will always have worse performance than the optimal look-ahead algorithm. Table 4.12 exemplifies the differences that

Table 4.12: The table displays the difference between the best stochastic algorithm and the optimal look-ahead algorithm for a model where the number of consecutive opens from a node has a geometric distribution with parameter p. The cost of migrating the file is set to 1* file size, and the fraction accessed per open is a constant equal to f. The difference is expressed in percentage of the performance for the optimal look-ahead.

Probability p	Fraction accessed per open								
	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
0.05	0.0	0.0	0.0	0.1	0.3	1.9	3.1	4.0	4.7
0.10	0.0	0.0	0.1	0.6	1.0	4.2	6.5	8.4	9.9
0.15	0.0	0.0	0.2	1.3	2.3	7.0	10.5	13.4	9.3
0.20	0.0	0.0	0.6	2.5	4.2	10.3	15.1	19.0	8.7
0.25	0.0	0.1	1.2	4.1	6.7	14.3	20.4	17.6	8.1
0.30	0.0	0.2	2.1	6.2	9.9	19.0	26.6	16.3	7.5
0.35	0.0	0.5	3.5	9.0	14.0	24.7	24.2	14.9	7.0
0.40	0.0	1.0	5.4	12.6	19.0	31.6	22.0	13.6	6.4
0.45	0.0	1.9	8.0	17.2	25.4	28.2	19.8	12.4	5.8
0.50	0.1	3.2	11.6	23.1	33.3	25.0	17.6	11.1	5.3
0.55	0.3	5.3	16.5	30.6	29.0	22.0	15.6	9.9	4.7
0.60	0.6	8.4	23.0	40.4	25.0	19.0	13.6	8.7	4.2
0.65	1.4	13.1	32.0	34.3	21.2	16.3	11.7	7.5	3.6
0.70	2.9	20.2	44.7	28.5	17.6	13.6	9.9	6.4	3.1
0.75	6.0	31.1	35.9	23.1	14.3	11.1	8.1	5.3	2.6
0.80	12.0	48.7	27.7	17.9	11.1	8.7	6.4	4.2	2.0
0.85	24.5	34.8	20.0	13.1	8.1	6.4	4.7	3.1	1.5
0.90	53.5	22.1	12.9	8.5	5.3	4.2	3.1	2.0	1.0
0.95	24.6	10.5	6.2	4.1	2.6	2.0	1.5	1.0	0.5

can be expected between the best stochastic algorithm and the optimal look-ahead algorithm when the number of consecutive opens has a geometric distribution with parameter p. The calculation assumes that the cost of migrating a file is equal to the file size. The difference is given in percent of the performance of the optimal look-ahead algorithm for opens that access a fraction f per open with probability p of continuing to reference the file. In the ranges that are similar to the ones typically found in the trace we analyzed, $0.3<f<0.7$ and $0.35<p<0.75$, the difference is from 3% to 44%, but most of the differences are in the 20% range. The observed differences are therefore similar to the ones that must be expected between the best stochastic decision and the optimal look-ahead algorithm. This is a clear indication that there is little to be gained from trying to improve the migration decisions made by the batch Poisson algorithms.

To summarize, in this section we have analyzed the difference in the performance between the proposed algorithms and the optimal look-ahead algorithms. We believe only a slight improvement of the migration decisions is possible, so the quality of this algorithm is satisfactory. The differences are within what must be expected between the best stochastic algorithm and a look-ahead algorithm.

The batch Poisson algorithms need several remote opens from a node before a file can be migrated to the node for the first time. These opens are needed to estimate the benefit of migrating the file, and they can be considered a waste for nodes the file will be migrated to. They also represent a large part of the difference between the batch Poisson algorithms and the optimal look-ahead algorithms. However, a file should only be migrated to a few of the nodes using it, and the benefit of migrating must be estimated first. The alternative is to migrate the file to all nodes, which substantially increases the network traffic. A significant improvement in the network traffic is therefore only likely if

the nodes that the file should be migrated to could be identified without observing the reference pattern. This is an area of future research, but such a scheme must depend on the file content, the identity of the user and the installation.

4.8. Conclusion

In this chapter we have investigated the performance of several different file placement algorithms in a distributed system. We found that file migration will reduce the network traffic compared to static placement. The algorithm that assumes a geometric distribution for the number of consecutive reference from the same user has the lowest network traffic. That algorithm uses the average fraction of a file accessed per open as the decision variable; if the fraction is larger than a given limit, the file is moved to the processor the user is on. The limit is unique for each user/file pair, and it is modified as the reference pattern changes; a consecutive reference from the same user decreases the limit, and a change in locality increases the limit. The algorithm is based on an Bayesian approach for estimating the distribution of the number of consecutive references from the same user. It will therefore adapt itself to the different reference pattern for the various files and users. However, the algorithm needs to collect several statistics for each node accessing a file. The algorithm is, however, decentralized. Each node collects its own reference history, and all migration decisions are made individually by each node based on the data it has collected. A version of the algorithm that maintains no statistics from previous runs (set of consecutive opens) is easy to implement. For one of the traces, SLAC, the network traffic is increased, while it is decreased for the one of the others, Amdahl. For the third one, the results remain the same.

Using some knowledge about the intended users when we place users at nodes, it is possible to reduce the network traffic substantially. If users from the same department are allocated to the same node, the lower bounds are reduced by more than 54%. The relative ranking of the algorithms is unchanged, but the difference in network traffic between static and dynamic algorithms is reduced.

We recommend the algorithm that does not retain any statistics from previous runs (BAT2/BAT1 with stack size 1) for two reasons:
1) It is easy to implement, and it does not require a large data structure.
2) It is more robust to different reference patterns, since the algorithm bases the migration decision only on the properties of the current set of consecutive references. In the chapter we showed that this is beneficial if the distribution for the number of consecutive opens is a mixture of geometric distributions.

We feel that the algorithm's robustness is worth the increase in network traffic. However, for some installations the cost in terms of increased network traffic may be high (like for the trace from SLAC).

Table 4.13: The table displays the definitions used to explain the various algorithms.

$V_n(i)$	minimal network traffic to a file from the start of the trace up to the n-th reference, given the file is placed at node i after the n-th reference.
B_n	is the I/O traffic (measured in bytes) associated with the n-th reference.
F_n	is the file size after the n-th reference.
d_n	is the identity of the node issuing the n-th reference.
$E(CB_j)$	Expected number of bytes transferred by consecutive opens from node j
f	fraction accessed per open $= B/F$
α	discounting factor
p_j	probability a batch of opens is from node j
k_j	probability the next open from node j is from the same batch as the current one
λ_j	rate node j is issuing batches of opens
p_{kj}	probability an open (to the same file) from node j will follow an open from node k
r_j	shorthand notation for $p_{jj} = k_j + (1-k_j)p_j$ probability an open (to the same file) from node j will follow an open from node j
p'_{kj}	probability an open (to the same file) from node j will follow an open from node k given the open is not from node j
γ	amortizing factor
C_{ij}	number of changes in locality from node i to node j.
M_i	number of file migrations from node i
R_i	number of file references from node i
U	the cost of updating a directory
Cost	
c_r	Cost of a remote access measured per byte transferred
c_u	Cost of updating a copy measured per byte transferred
c_t	Cost of creating a copy measured per byte transferred

Chapter 5
File Placement in Distributed File Systems with Replication

Summary

This chapter analyzes file placement in distributed file systems that allow replication. The specific questions we attempt to answer are:
1) What are the additional advantages of replication compared to migration alone?
2) What are the best algorithms for determining when and where to replicate a file?
3) What is the advantage of updating versus invalidating a copy each time the file is updated?
4) What effect do different rules for allocating users to nodes have on the results of the analysis?

We find that replication can potentially reduce the network traffic by 50% compared to migration alone. The cost is a doubling of the time-averaged file system size for shared files. For making good placement decisions, the average reference rate, the estimated number of consecutive updates, and the time a node has referenced the file are sufficient characterization of a file's reference process.

Whether to invalidate or update copies depends on the installation and the rule for allocating users to nodes. File systems should therefore implement both strategies. For the algorithms we propose, the difference between updating and invalidating is minimal (3 and 5% of the performance) for two of the traces. For the last trace, Amdahl, we found a difference ranging from 22% to 30%, depending on the rule for mapping users to nodes. Invalidation is the preferred scheme for sequential files, since these files are usually overwritten when they are updated.

The last part of the chapter investigates the quality of the proposed algorithms. We claim that the algorithms have a satisfactory performance, and to support the claim we analyze the difference between the actual simulation results and the lower bounds on the system performance. The difference in performance due to "wrong" decisions are within the range that must be expected between decisions made under uncertainty about future references and those made with certainty. The proposed algorithms must estimate several parameters. Before these parameter values are estimated, all references are handled remotely, and this traffic is the source for most of the difference in performance. However, demand copying is not a suitable solution, because most files should not be copied.

5.1. Introduction

In this chapter we analyze file placement algorithms for file systems that allow replication of files. Replication of files offers three advantages: 1) If there is an extended locality in the references to a file, i.e., several nodes are accessing a file randomly during an interval, it is possible to reduce the network traffic by replicating the file at the active nodes in the extended locality. 2) The availability of a file increases with the number of copies. 3) It is possible to improve the load balancing, since several locations can service a file reference, and bottlenecks can be avoided.

Algorithms that allow replication of files should have better performance than algorithms restricted to only migration of files, since migration of a file can be viewed as a special case of replication; the file is copied to a new location and the old copy is erased. In addition, algorithms that can replicate files have a wider spectrum of possible decisions,

and their performance should be better. One of the goals of this chapter is to identify this additional advantage.

The advantages of replication are only obtained at the cost of a more complex file system. The file system becomes larger, and retrieval of files is more complex when several copies exist. However, the major problem with replicated files is to maintain mutual consistency between the different copies of a file; an update to the file must be migrated to all copies of the file before these copies can be used. For our purpose the details of the implementation are not important.

A consistent state of the copies can also be guaranteed by erasing (invalidating) all additional copies when the file is updated. This approach was chosen in the distributed file systems ITC [Satyan85] and Swallow [Svobod81]. Part of this chapter analyzes the performance difference between updating copies and invalidating them.

Replication affects the concurrency control scheme, which prevents updates from one user from changing the view another user has of the file. Concurrency control is often implemented using locks such that when one user is updating a section of a file, other users can neither read nor write that section. This is more difficult in a distributed system with replication where the actions of the different nodes must be coordinated. Partitioning of the network poses an additional problem, since the partitions are no longer aware of file references in the other partitions.

In Section 2 we outline a plausible file system organization. Section 3 discusses the cost structure, while the next one summarizes the method we used. Section 5 formulates the best possible replication strategies given full knowledge of future references (look-ahead strategies). Section 6 describes the frame work of the file placement policies and the different simulation experiments we conducted. The results for the optimal look-ahead placement are discussed in Section 7, while the trace driven simulation is analyzed in Section 8. This last section also discusses the quality of the proposed algorithms. All definitions that are used in the chapter are displayed at the end in Table 5.12.

5.2. Organization of File System with Replication

In order to justify our assumptions and cost structure, it is necessary to describe a plausible file system organization that allows replication. Our research is not about file organization, and a detailed description of file management mechanisms will not be needed. The best implementation of these mechanisms depends on the system and the application. Except for maintaining consistency, all algorithms that allow replication require the same mechanisms. We are not considering specific existing hardware, and the specific details of the implementation are less important, because we cannot judge the actual cost. Our goal is to illustrate the improvement in performance owing to replication, both with updating and invalidating copies. Whether replication should be used must be a judgement between the expected improvement in performance versus the increased complexity of the file system.

5.2.1. Structure

The file system with replication has the same structure as without replication. Each node has a directory containing information about all files stored at that particular node. In addition, there must be one or more name servers that know the placement of all files. To reduce the requests to the name server, each node has a cache containing the storage locations of the most recently requested files. The cache is not updated when files are moved, so the cache only provides hints of the correct storage location.

5.2.2. Concurrency Control

The concurrency control scheme is the same as for file systems that do not allow replication. Locks at the granularity of a file are used to coordinate opens from different users. We consider two types of locks : multiple read locks and exclusive update locks. All updates to a file are serialized, i.e., two users cannot update the file simultaneously. For efficiency reasons, there is a timeout on the length a lock can be held to avoid a single user monopolizing a particular file.

5.2.2.1. Lock Management

A simple two-phase locking scheme [Bernst81] should be sufficient. All references to the copies are coordinated / administrated by a single entity called the administrative node (AN). The AN is responsible for granting locks to a file, and for setting up the connection between the node requesting access (RN) and the node storing a copy of the file (SN). The concept of an administrative node, a requesting node, and a storing node is identical to the one used in LOCUS [Walker83]. The AN for a particular file is found through the information stored at the clearing house.

For efficiency reasons, a third type of lock, an exclusive access lock, should also be used. Without an exclusive access lock, all nodes are forced to request a lock from the AN each time they open the file. This increases the overhead of an open. Nodes with an exclusive access lock on a file need not coordinate their references to that particular file with any other nodes. The lock signals that the node owning it is the only one interested in the file, and it is unnecessary to request a lock each time the node opens the file. The overhead of referencing the file is thereby sharply reduced. This is an advantage for files that are not shared, which are the majority of all files. The nodes using these files can be granted exclusive access locks to them, and all lock management can be handled locally. The exclusive access continues until the lock expires or is called back.

The exclusive access lock is different from an exclusive update lock. The exclusive update lock only guarantees that the node holding the lock is the sole current user of the file. Each time the node opens the file, it must still obtain a lock from the AN. Holding the exclusive access lock frees the holder from coordinating any of its opens to the file with any other node. It can continue to reference the file until the lock is recalled or expires. An exclusive access lock can therefore be viewed as a lock one level higher than read and update locks.

If a node does not have an exclusive access lock, it must request a lock from the AN. The AN will recall any exclusive access lock (at most one) before any lock can be granted. The lock scheme we have described is similar to the ones used in ITC [Satyan85] and Quicksilver [Cabrer87].

5.2.3. Administration of Copies

In addition to granting locks to a file, the AN is responsible for maintaining consistency between all copies of a file and for setting up a connection between a requesting node (RN) and one of the nodes that has a copy of the file (SN). The placement algorithms considered in this chapter are used by the administrative node (AN) to determine when and where files should be replicated.

The administrative nodes can be organized in many different ways ranging from a static AN common for all files, to a dynamic changing AN for each file. The latter is the most flexible one and the least susceptible to congestion and node failure. However, the directory of the file system, which contains the AN for each file, must be updated each time a new AN is appointed. For our analysis, we assume each file has its own

administrative node, which can change as the reference patterns change. The third exclusive access lock is then no longer needed. The placement of the AN will serve as an implied exclusive access lock, since all lock requests from the AN itself can be handled locally. The administrative node should therefore be the node which will reference the file most often (regardless of mode). A maximum number of lock requests will then be local ones.

We will not discuss the mechanisms for appointing a new AN in case of node failure or network partitioning. These mechanisms are known, and they affect all file placement algorithms in the same way.

5.2.4. Migrating of Updates

All updates to a file must be sent to the Administrative node (AN), since it is responsible for maintaining consistency between the copies. We do not allow any temporary inconsistency, so, as long as a file is replicated, all copies must contain the latest version of the file. Maintaining consistency can either be achieved by updating the copies or by invalidating all remaining copies. Updating of copies is either done by pushing or by pulling the updates to the copies. With pushing, the administrative node sends all updates to the nodes that have a copy. The AN will mark those nodes that acknowledge the update as containing a valid version of the file. With pulling, the nodes themselves are responsible for obtaining the updates from the node with the most recent version. The AN only sends a message stating the file is changed and the time the file was changed. A node can therefore not be restarted after a crash before it has asked for and received all update messages for the files stored at its disks. This is the scheme used in LOCUS [Walker83].

The scheme for invalidating is similar. The AN marks all additional copies as invalid when the file is updated. The invalidation messages should also be sent to the nodes that store the invalid copies, so the nodes can erase them. Alternatively, each node can also be responsible for detecting when a copy is invalid. This can be enforced by requiring each node to request the time stamp of the latest version of the file when it opens the file. Only when a copy's time stamp is the same as the file's time stamp, can the copy be used, else it is invalid.

Clearly these protocols must be augmented to handle failures by the AN. However, these mechanisms are common to all the file placement algorithms that allow replication, and they will therefore not affect the relative performance of the algorithms. These failures also occur infrequently, and an effective implementation should minimize their effects on the performance. A detailed description is not needed for evaluating different file placement algorithms.

5.2.5. Master Copy

In a replicated file system, copies will be created and erased at any time by the file placement algorithm, without any interaction with users. To guarantee that a copy of a file always exists somewhere in the system (at least until erased by a user), one of the copies is designated the master copy. It has a protected state, and cannot be erased by the placement algorithm unless it first appoints a different master copy. This is equivalent to moving a master copy from node to node as the reference pattern to the file changes.

As previously mentioned, the master copy should be placed at the node that coordinates all accesses (AN). A remote reference will then only involve two nodes (RN and AN), instead of three. The performance is further improved if the AN is the node that references the file most often (regardless of mode), since theses references then can also be handled locally.

5.3. Cost Structure

We use the same cost structure as in the previous chapter. In addition, a file storage cost is included to reflect the increased use of storage space due to replication. The costs are not necessarily measured in dollars. Instead, they are meant as weights used to compare the different actions that can be taken by the algorithms. As with the single copy algorithms, we assume the weights are proportional to the number of bytes transferred or stored. The cost of erasing a copy is ignored. The cost elements are :

c_r Cost of a remote access measured per byte transferred

c_u Cost of updating a copy measured per byte transferred

c_t Cost of creating a copy measured per byte transferred

c_s Cost of storing a file measured per byte stored per time unit

The first three costs c_u , c_t , c_r are system dependent, while c_s depends on the value an installation puts on storage compared to network traffic. For convenience c_u , c_t , c_r are unitless, and c_s is measured in $\frac{1}{time\ units}$. This unit can be viewed as a measure of the willingness to store data to avoid network traffic.

5.3.1. The Cost Values Used

Many different cost values are possible, but we only investigate a few. In an actual implementation of the proposed algorithms, the cost values should be different and reflect the situation at the individual installation.

The same cost structure is used both by the algorithms to make the decision when to create or erase a copy, and to evaluate the performance of the algorithms. The performance is measured in a two element vector consisting of the total network traffic and the time-averaged file system size. All types of network traffic have the same weight, i.e., $c_u = c_t = c_r$. The administrative overhead of migrating updates to copies is disregarded. Our assessment of the update cost is therefore optimistic, since we neither account for the overhead nor the reduced performance owing to more complex file system.

In file systems that allow replication, it is possible to make a trade off between file system size and network traffic. In order to generate a spectrum of possible combinations of network traffic and file system size, we use an artificial storage cost, c_s. It reflects the willingness to trade storage space for network traffic. We arbitrarily chose three different values for c_s to generate possible combinations of file system size and network traffic. The three values for c_s are set so 1 byte of network traffic equals 1) 1 byte stored for 8 hours, 2) 1 byte stored for 48 hours, and 3) 1 byte stored for the duration of the trace. With the latter value it is beneficial to replicate all bytes accessed only in read mode at all nodes accessing them. These are not "correct" values, but chosen to illustrate a reasonable spectrum of the different trade offs between network traffic and file system size.

5.4. Method

We use trace driven simulation to compare the different algorithms, as we did in Chapter 4. In the simulation, each user is permanently allocated to a "home" processor. All jobs submitted by a user run on its home processor. The allocation scheme affects the simulation results, since it determines whether users sharing the same file will be placed on the same node. The simulation results are therefore reported separately for each allocation, as was done in chapter 4. The next section describes the two schemes we used.

5.4.1. Rules for Allocating Users to Nodes

In chapter 4 we found that the number of nodes in the system had little influence on the performance, while the rule for allocating users to nodes had a much greater influence on the performance. We investigate the same two allocation rules here: a) one user per node, and b) all users from the same department to the same node.

One user per node is a reasonable model for a system consisting of workstations and personal computers. It is also a reasonable approximation for a large (> 10 nodes) system, with random allocation of users to the nodes (see previous chapter).

Allocating users from the same department to the same node is an example of what can be gained by utilizing some knowledge about the intended users of the system. Most of the sharing is expected to be along organizational lines. With the department mapping, most of the users sharing a file will be at the same node, and the network traffic between different nodes is expected to be lower.

With one department per node, the reference patterns on a node level are different from those with one user per node. All the algorithms are designed based on the observed reference pattern for each user. The results with one department per node will therefore provide a measure of the robustness of the algorithms (see previous chapter).

5.5. The Optimal Look-ahead Policy

One of the goals of the thesis is to analyze the advantage of replication compared to migration or static placement. The optimal look-ahead algorithm is used as a tool for this comparison. This algorithm makes the placement decision with full knowledge of future references to the file. It is not a realizable algorithm, but it gives the best possible placement. It therefore provides a bound on the performance of any realizable algorithms. In addition, it can also be used to measure the advantage of replication compared to migration or static placement. The difference between the three optimal look-ahead algorithms measures the relative advantage of these strategies without any distortions from ineffective implementation of the realizable algorithms. The optimal look-ahead policy also provides hints of the properties a good algorithm should have, and it can determine the algorithms that should be investigated.

We do not consider any incremental copy schemes, where only parts of the file are copied or migrated, since the traces cannot be used for such schemes. The individual I/O operations are not traced, and the individual disk blocks cannot be identified. Policies that transfer only particular blocks and not the whole file can therefore not be considered.

Many users transfer only a fraction of the file per open. One alternative we did not implement was transferring the remaining parts of the file after a remote open. The cost of creating a local copy would then be lower, since the parts that had been transferred during the remote open would not have to be transferred when the file is transferred. In the analysis, the cost for creating a copy, c_l is equal to 1 (the cost of remote I/O). Setting it to a value smaller than 1 would be equivalent to a policy that transferred the remaining

part of the file, since all the policies we analyze use the product $c_t F$; the results are the same whether we analyze policies that transfer less than the file size, F, when they create a copy, or policies that have a transfer cost, c_t, less than 1. The policies that transfer the remaining unreferenced parts of file after an open can therefore be considered as a special cases of migration policies with different cost values.

5.5.1. Formulation of the Optimal Look-ahead Policy

We assume there are no storage constraints in any of the nodes in the cluster. Each file can then be evaluated independently, and the optimal look-ahead policy can be formulated as a forward dynamic programming equation. For the formulation, the following definitions are needed:

b_n^u	Amount of update I/O traffic by the n'th access to the file
b_n^r	Amount of read I/O traffic by the n'th access to the file
b_n	Amount of I/O traffic by the n'th access to the file $b_n = b_n^r + b_n^r$
T_n	time between the n-1'st and n'th access
F_n	File size after the n-1'st access
i_n	Identity of the node issuing the n'th access
a_n	mode of the n'th access. If the access is an update, $a_n = 1$, otherwise $a_n = 0$
N	Total number of accesses to the file
m	Total number of nodes accessing the file
I	Subset of the m nodes that have a copy of the file
$\|I\|$	Number of elements in set I. $\|I\| > 1$, since I is never an empty set.
$V_n(I)$	The minimal cost up to the n'th access when I is the state at the end of the n'th access

The initial condition is given by $V_0(I) = 0$ for all $|I| = 1$, and $V_0(I) = \infty$ for all $|I| \neq 1$. The optimal solution is $\min_I V_N(I)$.

If we disregard the duration of an open, the recursive formulation of $V_n(I)$ is :

$$V_n(I) = \min_J \Big[V_{n-1}(J) + c_t \cdot F_n \cdot |I \bigcup J - J| + c_s T_n F_n |J| + 1_{i_n \notin I} \cdot c_r b_n + a_n c_u b_n^u \cdot [|I| - 1 + 1_{i_n \notin I}] \Big] \quad (5.1)$$

Explanation of the various elements in Equation 5.1

$1_{i_n \notin I}$	is 1 if the node issuing the n'th open does not have a copy, otherwise it is 0
$c_t \cdot F_n \cdot \mid I \bigcup J - J \mid$	Cost of creating copies at the nodes that did not have a copy after the n-1'st reference
$c_s T_n F_n \mid J \mid$	the storage cost for the copies from the n-1'st reference to the beginning of the n'th reference
$1_{i_n \notin I} \cdot c_r b_n{}^r$	The cost of a remote read reference
$a_n c_u b_n{}^u \cdot [\mid I \mid -1+1_{i_n \notin I}]$	The cost of updating all other copies of the file. If the reference is a local one $\mid I \mid -1$ other copies must be updated.

With m nodes accessing a file, there are $(2^m - 1)$ different states. For each open to the file, equation 5.1 must be evaluated for each of the $(2^m - 1)$ states. Owing to the computational complexity, the formulation can then only be used for files shared by a few users. The next section discusses an alternative formulation that can be used for all files regardless of the number of users sharing the file.

5.5.2. Optimal Look-ahead with Fixed Master Copy

In the previous formulation, the state of all nodes in the cluster must be known each time a copy is erased. Otherwise, it is possible that the file could be erased at all nodes. However, if a node can assume that a copy exists somewhere (the master copy), each node can create and erase its copy independently of the status of the other nodes. The optimal placement can then be decomposed into m independent placement problems, one for each node. The computational work with this formulation is equivalent to a state space with $(2 \cdot m)$ states. This formulation requires a two stage calculation; the first stage calculates the optimal placement for each node assuming the master copy is placed at another node. After the optimal decision for each node is known, the second stage places the master copy to minimize the total cost for the system. The following recursive forward dynamic programming formulation is used to find the optimal location :

$V_n{}^k(j)$ the minimum cost from the first reference to the n'th reference given that the state at node k is j after the n'th reference. $j = 1$ signals that the node has a copy, otherwise the node does not have a copy

$$V_n{}^k(0) = \min\left[\left[V_{n-1}{}^k(0) + 1_{i_n = k} \cdot c_r b_n\right], \left[V_{n-1}{}^k(1) + c_s F_n T_n + 1_{i_n = k} a_n c_u b_n{}^u\right]\right] \quad (5.2a)$$

$$V_n{}^k(1) = \min\left[\left[V_{n-1}{}^k(0) + c_t F_n\right], \left[V_{n-1}{}^k(1) + c_s F_n T_n\right]\right] + a_n c_u b_n{}^u \quad (5.2b)$$

The initial conditions are $V_0^k(0) = 0$ and $V_0^k(1) = \infty$. The term $a_n c_u b_n{}^u$ accounts for the updates either to the master copy if $i_n = k$ or to the copy itself when $i_n \neq k$.

The cost for the optimal file placement at node i after N opens is then $\min\left[V_N^i(0)\ ,\ V_N^i(1)\right]$, given that the master copy is permanently placed at another node. The cost of keeping the master copy at this "other" node is only the storage cost (MC), $MC = \sum_{i=1}^{N} c_s T_n F_n$, since any updates or remote reads to the master copy are included in the cost for the node that made the updates or reads. If the master copy is permanently placed at node k , the minimal total cost (C) is:

$$C = \sum_{i \neq k} \left[\min\left[V_N^i(0)\ ,\ V_N^i(1)\right]\right] + MC$$

which is equal to

$$C = \sum_{i=1}^{m} \left[\min\left[V_N^i(0)\ ,\ V_N^i(1)\right]\right] + MC - \left[\min\left[V_N^k(0)\ ,\ V_N^k(1)\right]\right] \qquad (5.2c)$$

The two first terms of equation 5.2c are constant, so C is minimized if the master copy is placed at the node with the largest cost, i.e., at node k when

$$k = arg\ \max_i \left[\min\left[V_N^i(0)\ ,\ V_N^i(1)\right]\right]$$

This procedure is used to find the optimal look-ahead policy given that a master copy has to reside at a node for the duration of the trace (or until the file is erased by the user).

5.6. File Placement Policies

In file systems that allow replication of the files, a dynamic file placement algorithm must make two decisions, 1) when and where should a copy be created, and 2) when should a copy be erased. The decision to update is an implicit part of the decision to erase, since each time the file is updated all copies that are not erased must be updated. In addition, the algorithm must decide where to place the master copy.

This section analyzes in detail the conditions for creating / updating / erasing a copy and formulates several placement algorithms based on these conditions. The algorithms are based on the mean reference rate (measured in bytes per time unit), the expected number of consecutive opens in update mode, and the time a node has referenced the file.

5.6.1. Conditions for Creating / Updating / Erasing a File Copy

We believe a reasonable approach is to decompose the problem to one node at a time. This is not the optimal approach, since the optimal placement of the master copy can only be found by solving the placement problem for all nodes simultaneously. For each open, the desired state for each node is calculated, assuming the master copy is placed at another node. Then the best placement of the master copy is selected and the feasible states are implemented. (It is not feasible to erase a copy that has been appointed the master copy.) With this scheme, the decisions for each node are independent of all other nodes in the system, and the computational complexity is linear to the system size.

To discuss the conditions for creating / erasing / updating replicated files, we need the following definitions:

$E(B_i^r)$	Expected number of bytes transferred in read mode by node i per open
$E(B_i^u)$	Expected number of bytes transferred in update mode by node i per open in update mode
$E[R_i(T)]$	Expected number of read references during an interval T from node i
$E[U_i(T)]$	Expected number of update references during an interval T from node i
$E(T_i)$	Expected time between read references from node i
$E(T_i^u)$	Expected time between update references from node i
$\lambda_i^r = \frac{1}{E(T_i)}$	reference rate measured in accesses per time unit
$\lambda_i^u = \frac{1}{E(T_i^u)}$	update reference rate measured in accesses per time unit
$r_i = \frac{E(B_i)}{E(T_i)}$	reference rate measured in bytes per time unit
$r_i^u = \frac{E(B_i^u)}{E(T_i^u)}$	update reference rate measured in bytes per time unit
$!=i$	short hand notation for all other nodes except node i
$E(F)$	Expected file size

The underlying idea is that a copy with a lifetime T should only be created if the expected advantage of local I/O to it is larger than the cost of creating and maintaining it. A copy will not have an infinite lifetime, and the advantage and the cost of having a local copy must be compared over its lifetime, T. The maintenance cost consists of two parts, the storage cost and the cost of updating the copy whenever the file is updated. The storage cost is proportional to the file size and the lifetime of the copy ($c_s\ T\ E(F)$). The update cost is set proportional to the expected number of bytes that must be updated. During an interval, T, the expected number of updates to the file from node k is $E[U_k(T)]$, so the expected number of updates that must be propagated to a copy at node i is $\sum_{j \neq i} \left[E[U_j(T)] \right]$. The expected updating cost can then be expressed as: $c_u \sum_{j \neq i} \left[E(B_j^u) E[U_j(T)] \right]$. The cost of creating a copy is set proportional to the number of bytes that is transferred (the file size) or $c_t F$. The final element of the cost equation is the expected advantage of local I/O to the copy. During an interval T, the expected number of bytes read from node i is $E(B_i^r)E(R_i(T))$, and the advantage of local I/O is set

proportional to this traffic. The update traffic from node i to the file does not affect the advantage of creating a local copy at node i, since we assume a master copy is placed at another node: even if a local copy is created, the master copy must still be updated each time the file is updated by node i. The condition for creating a copy can then be expressed as: create a copy if the expected advantage of local I/O during the copy's lifetime, T, is larger than the cost of creating and maintaining it, or

$$c_r E(B_i^r) E[R_i(T)] > c_u \sum_{j \neq i} \left[E(B_j^u) E[U_j(T)] \right] + c_t F + c_s TE(F) \qquad (5.3a)$$

Using the same type of argument, the condition for erasing a copy that will remain erased for T time units is:

$$c_r E(B_i^r) E[R_i(T)] < c_u \sum_{j \neq i} \left[E(B_j^u) E[U_j^u(T)] \right] + c_t F + c_s TE(F) \qquad (5.3b)$$

These equations assume that on a per node basis, the expected number of bytes transferred per open $E(B_i^r)$ is independent of or equal to the number transferred by previous opens.

The epochs when the conditions are evaluated influence the expectations $E[R_i(T)]$ and $E[U_j(T)]$, since in most cases,

$$E[R_i(T) \mid \textit{time since last accesses} = 0] \neq E[R_i(T) \mid \textit{time since last accesses} = t]$$

The coefficient of variation of the time between references is larger than 1, and the expected number of references in an interval T typically decreases with the time since last access. As T approaches ∞, $E[R_i(T)]/T$ will approach $1/E(T_i)$.

The condition for creating a copy at node i (eq 5.3.a) needs to be evaluated only when node i references the file. If the copy is created at node i at any other time, there is a potential for incurring unnecessary network traffic; node i may not reference the file again, or the copy may have to be updated before node i references the file again. In both cases the network traffic is reduced by waiting to create the local copy until node i opens the file. The only advantage of creating a copy "prematurely" is a reduction in the access delay for node i, but delay is not a part of our performance metric.

The condition for erasing a copy at node i (eq 5.3b) must be evaluated for all copies each time the file is updated, since a policy always has the choice between updating or erasing the copy. Equation 5.3b must also be used to evaluated for how long a copy can remain unreferenced before it should be erased. The conditions for erasing a copy will therefore be evaluated at any time, while the condition for creating a copy at a node will only be evaluated whenever that node opens the file.

In the special case of buffer management, equation 5.3a can be simplified. The transfer size will be equal to the expected number of bytes transferred per open and per update, since the individual disk blocks cannot be partially read or written. In the simple case where $c_t = c_r = c_u$ and $c_s = 0$, equation 5.3a can be simplified into: create a copy if $E[R_i(T)] > \sum_{j \neq i} \left[E[U_j^u(T)] \right] + 1$. The best policy is then to always create a local copy when a disk block is referenced and to erase it whenever the block is updated. $E[R_i(T)]$ is then > 1, while $E[U_j(T)]$ is equal to 0, so demand replication will be the best possible policy. However, this simple policy is only the best one when the following conditions are true: the storage cost can be ignored, all types of network traffic have the same cost, and the individual disk blocks can be replicated. In the general case, such a simple policy is

not the best one, and later in the chapter we show that demand copying of files is not a suitable policy.

5.6.1.1. The Condition for Creating a File Copy for File Systems with Invalidation

Equation 5.3a can also be used to evaluate when copies should be created even if they are invalidated instead of being updated. With invalidation, none of the additional copies are updated and the expected amount of update traffic per copy is 0.0. The time horizon T should, however, be different, since when copies are invalidated instead of being updated, their expected lifetime is shorter.

5.6.2. Placement Algorithms with File Replication

This section describes how equations 5.3a and b can be implemented as file placement algorithms. Such an implementation must be based on some assumption about the stochastic characteristics of the reference process.

Chapter 3 investigated some of the properties of the file reference process. We found that the number of bytes transferred per open can be modeled by a constant for each user/file pair. For most of the user/file pairs, the number of bytes transferred is either constant or independent of previous reference history. The time between references for a user/file pair may be characterized by a renewal process with a high coefficient of variation. In Chapters 3 and 4, we concluded that a batch Poisson model with a geometric batch size is appropriate for a single copy file system. For this model, the references from a user to a file arrive in batches, and the time between the batches has an exponential distribution. For simplicity, the time between opens in the same batch is also modeled by an exponential distribution, but with a substantially smaller mean. Further, we assumed that the number of opens per batch had a geometric distribution. The same starting point is used in the discussion of what constitutes a sufficient characterization for file placement algorithms in file systems that allow replication.

The condition for creating a copy at a node (equation 5.3a) should only be evaluated when the node opens the file. Creating the copy at any other time will only result in a potential performance degradation. The copy may not be used for a while, or it may have to be updated before the node references it. In either case, the performance is improved by waiting to create the copy until the node opens the file. The only potential advantage of creating the copy before it is opened is a reduction in the access delay, but delay is not a part of our performance measure. If a user references a file according to a renewal process, equation 5.3a will only be evaluated just before the renewal epochs of the process. The mean time between references is then a sufficient characterization for equation 5.3a, and $E[R_i(T)] = 1 + \frac{T}{E(T_i)}$, where $\frac{T}{E(T_i)}$ is the expected number of opens in the next T time units after the node has opened the file.

The amount of update traffic in the interval T is more difficult to predict. The updates to a file are the superposition of each user's updating process to the file. If we assume there is a constant probability an open will update the file, and that each user references the file according to a batch Poisson process, the updates to the file will also be according to a batch Poisson process. The underlying assumption is that all users sharing a file are interested in using the file during the whole trace period. Few users do that, and instead they will reference the file according to batch Poisson process for only a part of the trace period. The model of the updates to the file therefore also depends on the characteristics of the processes that determine when a user will start accessing a file and when the user will loose interest in the file. The two latter processes are not well

understood and are not characterized. Instead, we chose the simplifying assumption that the expected amount of update traffic in an interval T in equation 5.3a is $T \cdot \sum_{j \neq i} \frac{E(B_j^u)}{E(T_j^u)}$. This implies that the update traffic is overestimated, since not all users will continue to update the file during the whole trace period.

The condition for creating a copy (eq. 5.3a) can then be rewritten as:

$$c_r E(B_i^r)[1+T/E(T_i)] > c_u \cdot T \cdot \sum_{j \neq i} \left[\frac{E(B_j^u)}{E[T_j^u]} \right] + c_t F + c_s T E(F) \qquad (5.4a)$$

In stead of basing the condition for creating a copy on the expected number of opens in an interval T, the condition can be expressed in terms of the expected reference rates per byte ($r_i = \frac{E(B_i^r)}{E(T_i)}$). By dividing both sides by T and rearranging the terms, the conditions for creating a copy becomes: created a copy at node i when

$$c_r \cdot r_i(1+E(T_i)/T) - c_u \cdot \sum_{j \neq i} r_j^u - c_s \cdot E(F) - c_t \cdot \frac{F}{T} > 0 \qquad (5.4b)$$

Typically, $\frac{E(T_i)}{T} < 1$ for files that are replicated, since a copy should only be created for nodes that will reference the file several times during the copy's lifetime. During the initial analysis, we found that the placement algorithms had a tendency to create too many copies. We therefore used the slightly stronger condition for creating a copy : create when

$$c_r \cdot r_i - c_u \cdot \sum_{j \neq i} r_j^u - c_s \cdot E(F) - c_t \cdot \frac{F}{T} > 0 \qquad (5.4c)$$

This equation is just a restatement of our original condition for creating a copy, but with a different time scale; a copy should only be created if the expected advantage of local I/O per time unit is larger than sum of the expected storage cost, the expected cost of updating the copy per time unit, and the amortized cost of creating the copy (measured per time unit).

Equation 5.4c compares the long-term expected advantage of creating a copy against the expected cost. Even if it is beneficial to create a copy for the long-term benefit, it can be advantageous to erase it for a shorter time period. The long-term cost of updating a file is expressed as constant rates, yet updates will most likely occur in batches. In Chapter 3 we concluded that users tend to reference a file in batches of opens, with little overlap between batches from different users. Updates will therefore most likely also arrive in batches. For the duration of the batch of updates, it can be advantageous to erase all other copies of the file and only update the master copy. At the end of the batch with updates, the copies can be recreated at the nodes that will reference the file again. Equation 5.3b can still be used, but T should be set to the expected duration of the batch. The expected amount of update traffic will be the expected number of bytes in the batch of updates, and the expected number of read opens should be set to 0.0, since batches from different users will seldom overlap ($E[R_i(T)] = 0$). The condition for temporarily erasing a copy can then be expressed as : erase the copy if the expected cost of updating the copy in the near future is larger than the cost of recreating the copy.

Equation 5.3b can also be used to identify copies that should be erased when there are no references to them. As long as we associate a cost with storage, there will exist situations where the expected cost of storing a copy until the next reference is larger than the cost of recreating the copy. In Chapters 2 and 3 we found that the time between references had a very large coefficient of variation. We expect that the probability of a reference in the next t time units will decrease with the time since last reference. The best policy is then to erase a copy at node i if the copy has not been referenced within the last say d time units. To calculate d, we must know the exact distribution of the time between references both from node i and for the update process to the file.

We do not believe this is an important performance issue. The disk space is seldom so scarce that copies that are still used must be erased to provide space for other copies. In addition, a separate mechanism is needed to erase copies from nodes that no longer use them. Unfortunately, we have not identified a model for this process, and instead use the working set criterion : erase all copies that have not been accessed in the last d^* time units. d^* is set to the ratio between the transfer cost and the storage cost. Such a value for d^* minimizes the maximum cost of the wrong decision. If a copy is erased and the node later starts referencing the file again, the maximum cost is the cost of recreating the copy at the node, $c_t F$. On the other hand, if a copy that is not used remains at a node for d time units, the cost is $c_s F\, d$. The maximum cost of the wrong decision is then minimized when $d = \frac{c_t}{c_s}$. This criterion will also erase copies that are temporarily not used, and it partially incorporates the idea discussed in the previous paragraph.

In this sub-section we have argued that the mean reference rate plus a simple batch model of the updates is a satisfactory probabilistic characterization of the reference process. The conditions for creating / erasing / updating copies are only evaluated at epochs where this a simple characterization is sufficient.

5.6.3. Formulation of Policy

The framework of the policies we will consider is already given by equations 5.4c and 5.3b, and the probabilistic characterization of the reference process. However, there are three areas of the policy that must be investigated further: 1) the best estimation method of the reference rate $\frac{E(B_i^r)}{E(T_i)}$ and $\frac{E(B_i^u)}{E(T_i^u)}$, 2) The best time horizon T to use in the conditions for creating and erasing a copy, and 3) invalidating of copies versus updating.

In this subsection we discuss these three areas and the rules for managing the master copy. At the end the simulation experiments are summarized.

5.6.3.1. Estimation of the Reference Rates

We propose three different methods for estimating the reference rates $\frac{E(B_i)}{E(T_i)}$; an average rate, a normalized rate, and a discounted rate. The two latter methods are designed to detect and react to changes in the reference pattern. The average rate is also supplemented with a decision criterion that accounts for the uncertainty of the estimate. For convenience we use two different definitions of the reference rate, either λ, which is the reference rate measured in number of references per time unit, or r, which is the reference rate measured in bytes per time unit. To discuss the various methods we need the following definitions:

$!=i$	is a short hand notation for all nodes except node i
$t_i^{r,j}$	time of the j'th read reference from node i
$t_i^{u,j}$	time of the j'th update reference from node i
$b_i^{r,j}$	number of bytes transferred by the j'th read reference from node i.
$b_i^{u,j}$	number of bytes transferred by the j'th reference in update mode from node i.
$n_i(S)$	number of references from node i to the file up to epoch S
α	discount rate
d	maximum time a copy can remain unreferenced before it is erased

5.6.3.2. The Average Rate

At time S, the average read rate, $r_{i,a}^r$, from node i and the update rate to a copy at node i, $r_{!=i,a}^u$, are given by the following equations:

$$r_{i,a}^r(S) = \frac{\sum_{j=1}^{n_i(S)} b_i^{r,j}}{S - t_i^{r,1}} \qquad r_{!=i,a}^u(S) = \frac{\sum_{k\neq i}\sum_{j=1}^{n_k(S)} b_k^{u,j}}{S - \min_{k\neq i}(t_k^{u,1})} \tag{5.5}$$

The read transfer rate is only evaluated when node i references the file, and the estimate for r_a^r can change substantially, especially during the first few references. The inter-reference times $t_i^{j+1} - t_i^j$ are expected to vary, because the inter-reference-time distribution has a high coefficient of variation.

The condition for creating copies (equation 5.4) assumes there is no uncertainty associated with the estimates for the reference rates. Even though equation 5.4c may indicate that it is advantageous to create a copy, the expected gain of doing so may be negative when the estimates for the reference rates are uncertain. The decision criterion for creating a copy should therefore be augmented by an additional condition to account for the uncertainty of the estimates. In the initial study, the algorithms created more copies than the optimal look-ahead algorithm did. Our main concern is therefore to avoid creating copies based on an uncertain estimate. The idea is that unless the expected gain of greating a copy is large, a copy should not be created as long as the estimate of the read rate is uncertain. We are less concerned with the uncertainty associated with the update rate for two reasons: 1) Overestimating the update rate will only reduced the number of copies created, and we do not view this as a problem. 2) Whenever the file is updated, equation 5.3b is used to evaluate whether the additional copies should be erased. If a file suddenly is updated frequently during a short interval, most additional copies will be erased. This counteracts most effects of underestimating the update rate in equation 5.4c.

To account for the uncertainty of the estimate for the read rate for node i, we formulated a simple decision problem over a time horizon of two references, j and $j+1$. The

decision is whether to create a copy at node i, when equation 5.4c indicates it is advantageous to do so. Over a time horizon of two opens we consider four possible outcomes:

1) If a copy is created and equation 5.4c evaluated for the j+1st reference remains positive, creating the copy was the right decision, and no loss is incurred.

2) If a copy is created and equation 5.4c evaluated for the j+1st reference indicates that a copy should **not** have been created, the loss is set equal to the cost of creating a copy minus the advantage of local access for the jth reference.

3) If a copy is not created and equation 5.4c evaluated for the j+1st reference indicates that a copy should **not** have been created, not creating the copy was the right decision, and no loss is incurred.

4) If a copy is not created and equation 5.4c evaluated for the j+1st reference indicates that a copy should have been created, the loss is set equal to the cost of handling the jth reference remotely.

These decision outcomes result in the following decision/loss matrix:

Decision for the j'th reference | gain of creating a copy > 0			
Decision		Create Copy	Do not Create a Copy
Expected gain for creating a copy for the j+1'st reference	>0	0	$c_r E(B_i^r)$
	<0	$c_t F - c_r E(B_i^r)$	0

The expected gain of creating a copy for the j+1'st reference is a function of the expected gain for the j'th reference and the stability of the estimate for the reference rate. The expected gain of creating a copy at node i can be written as

$$c_r \cdot E(B_i^r) \cdot \lambda_r^i - c_u \sum_{j \neq i} E(B_j^u) \lambda_u^j - \frac{c_t F}{T} - c_s F \tag{5.5}$$

$\lambda_i^r = \dfrac{k-1}{t_i^{r,k} - t_i^{r,1}}$ is defined as the reference rate in read mode, while λ^{u_j} is defined as the reference rate in update mode for node j ($\lambda_r^i \cdot E(B_i^r) = r_i$). As previously mentioned, the update rate λ_i^u is assumed to remain constant. If the expected gain of creating a copy is larger than 0, the following condition must hold for λ_r^i

$$\lambda^{r_i} > \frac{1}{c_r \cdot E(B_i^r)} \cdot \left[c_u \sum_{j \neq i} E(B_j^u) \lambda^{u_j} + \frac{c_t F}{T} + c_s F \right] = C \tag{5.6}$$

If the time between the j'th and the j+1'st reference is x, the potential gain of creating a copy for the j+1'st reference will be negative if

$$\frac{j}{t_i^{r,j} - t_i^{r,1} + x} < C \implies x > \frac{j}{C} - (t_i^{r,j} - t_i^{r,1}) = L \tag{5.7}$$

In our decision model, the expected loss of creating a copy for the j'th reference is $P(X>L)\left[c_t F - c_r E(B_i^r)\right]$, while the expected loss of not creating a copy is $P(X<L) c_r E(B_i^r)$. A copy should therefore be created only if:

$$P(X>L)\left[c_tF-c_rE(B_i^r)\right] < P(X<L)\,c_rE(B_i^r) \quad or \quad P(X>L)\,c_tF < c_rE(B_i^r) \tag{5.8}$$

To pursue this analysis further, it is necessary to make some assumptions about the distribution of X, the time between references. To make the analysis tractable, we assume X has an exponential distribution with parameter μ^i. To account for the uncertainty of the estimate of μ^i, we assume μ^i has a prior gamma (θ_0,k_0) distribution, with $\theta_0 = t_2^i - t_1^i$ and $k_0 = 1$. The expected value for μ^i is then the estimated rate λ^i. The gamma distribution was chosen mainly for mathematical simplicity. If μ^i has a prior gamma (θ_j,k_j) distribution and the inter-reference time, X_j, has an exponential distribution, the posterior distribution for μ^i is still a gamma distribution with new parameters $\theta_{j+1}=\theta_j+x_j$ and $k_{j+1}=k_j+1$. The two assumptions of 1) exponential distributed inter-reference times with parameter μ, and 2) a prior gamma distribution are only used to assess the risk of the uncertainty of the estimated reference rate. With these simple assumptions, it is possible to get a a rough estimate of $P(X>L)$.

With the given assumptions it can be shown that

$$P(X > L) = \left[\frac{\theta_{j-2}}{\theta_{j-2}+L}\right]^{k_{j-2}} \tag{5.9}$$

A copy should therefore be created at node i for reference j if the gain of doing so is positive and

$$P(X > L) = \left[\frac{\theta_{j-2}}{\theta_{j-2}+L}\right]^{k_{j-2}} \cdot c_tF < c_r \cdot E(B_i^r) \tag{5.10}$$

We believe this approach reflects the risk of creating a copy based on an uncertain estimate.

5.6.3.3. Normalized Rate

In file systems where storage space is limited, copies will only exist for brief periods while a node is actively using a file. The average rate is then no longer useful, since it reflects the average reference rate from the time the node started accessing the file. The normalized rate on the other hand is defined to reflect only the reference rate during the potential lifetime of a copy. As already mentioned, it is necessary to have a condition that erases copies not referenced in the last d^* time units. The reference rate should therefore reflect only the rate to the copy during its potential lifetime, and not the rate while the node is active. We defined the normalized rate as:

$$r_{i,n} = \frac{\sum_{j}^{n_i(S)} b_i^{r,j}}{\sum_{j=1}^{n_i(S)-1} min(t_i^{r,j+1}-t_i^{r,j}\,,\,d^*)+\,S-t_i^{r,n}}$$

and

$$r_{!=i}^{u} = \frac{\sum_{k\neq i}\sum_{j=I_k} b_k^{u,j}}{\sum_{j=1}^{n_i(S)-1} min(t_i^{r,j+1}-t_i^{r,j}\,,\,d^*)+\,S-t_i^{r,n}}$$

The set I_k includes only those updates that will update the file while a copy potentially can exist at node i. Mathematically this constraint can be expressed as the set of all

indexes l such that

$$t_i^{r,j} < t_k^{u,l} < t_i^{r,j} + d^*$$

for some j.

5.6.3.4. Discounted Rate

The last method is motivated by the belief that the reference process may not be renewal, but may change slowly over time. All observations should then not be given the same weight in the calculation of the reference rate; instead the more recent ones should be given greater weight. To do this, we order into one set all read references from node i and all update references from the nodes j such that $j \neq i$. Let t^j be the epochs of this ordered set, $b^{r,j}$ and $b^{u,j}$ the number of bytes transferred in read and update mode. The discounted rates are then defined as

$$r_{i,d} = \frac{\sum_{j}^{n(S)} b^{r,j} \cdot \alpha^{n(S)-j}}{\sum_{j=1}^{n(S)-1} (t^{j+1} - t^j)\alpha^{(n(S)-j)} + S - t^{n(S)}}$$

and

$$r_{i,d}^u = \frac{\sum_{j}^{n(S)} b^{u,j} \cdot \alpha^{n(S)-j}}{\sum_{j=1}^{n(S)-1} (t^{j+1} - t^j)\alpha^{(n(S)-j)} + S - t^{n(S)}}$$

The ad hoc value of $\alpha = 0.95$ was used for two reasons. Any higher values for α would put nearly equal weights on all observations, while a smaller value for α would put only a small weight on older observations. The latter would be suitable if the changes in the reference rate were abrupt. The reference process would then be quite different from a renewal process, and the conditions for creating and erasing file would no longer be valid. With $\alpha = 0.95$ the 1st reference has only 60% of the weight of the 10'th reference, and the discounted rate will reflect slow changes in the reference process.

5.6.4. Time Horizon

A time horizon, T, is used both by the condition for creating a copy (eq 5.4c) and by the condition for erasing (eq 5.3b). A copy is erased either when the file is updated or when the copy has not be referenced for a while. In the former case, T is set equal to the expected duration of a batch of updates, while in the latter, T is set to the ratio between the transfer cost and the storage cost. An analysis of the time horizon, T, is only needed by the condition for creating a copy (eq 5.4c).

Initially, we thought the term $\frac{c_t}{T}$ could be disregarded in equation 5.4c, since we conjectured the expected lifetime of a copy, T, would be long. This assumption neglected the effect updating a file would have on the lifetime of a copy. As we previously explained, it is feasible that a copy can be erased temporarily when the file is updated frequently. Setting the expected lifetime to ∞ also overlooked the effect on the lifetime from erasing all copies that have not been referenced in the last d^* time units.

However, in the next section we show that the algorithms that used a long time horizon only had a moderate performance. The value of T used in equation 5.4 should reflect

the potential time a copy will be active, but we have not characterized the process that determines how long a node will continue to use the file. A simple model is to assume with a constant, but unknown, probability each open will be the last one from the node. This probability can be different for each user/file pair. For each open, the expected remaining time until the last open from the node is then equal to the expected time the node has referenced the file. This makes the time since the first reference from the node to the file (the age) a reasonable estimate for the remaining time the node will reference the file, and the time horizon T in equation 5.4c should be set equal to this time (the age of the reference process). Using the age of a node's reference process implies we believe a node that has referenced a file for a while is likely to reference the file again.

The expected life of a copy also depends on the distribution for the inter-reference time and the distribution for the time between updates. If the inter-reference time is larger than d^*, the copy will be erased. The condition given in equation 5.3b will also erase the copy if the expected cost of updating it is larger than the expected cost of recreating the copy. The effect these two events will have on the lifetime of a copy is not incorporated into the criterion for creating a copy. It would require a more detailed description of the file reference process. For algorithms that can update a copy, we believe few copies are erased and later recreated, and the consequence of not modeling this effect should be small.

However, it cannot be overlooked for the algorithms that invalidates all copies when the file is updated. The expected lifetime of a copy should then be set to the minimum of the expected time to next update of the file and the expected time the node will remain interested in the file.

5.6.5. Invalidating versus Updating a Copy

There is an overhead associated with algorithms that update all existing copies. In addition to the updates themselves, several messages must be exchanged between the node with the master copy and the other nodes to administrate the update. One alternative is to invalidate all additional copies each time the file is updated, and instead only to update the master copy. We expect this alternative will have higher network traffic than algorithms which can update a copy. However, the file system is simpler, since by definition all copies are consistent; each time the file is updated there will exist only one copy.

We therefore investigate the potential increase in network traffic caused by invalidating copies when a file is updated. Both the increase for the look-ahead algorithms and and for the different algorithms are analyzed. Another issue, which we only briefly discuss, is the added availability obtained by updating copies instead of invalidating them.

5.6.6. Selection of the Master Copy

We now discuss the selection of the master copy. A master copy cannot be erased until either the file is erased or another master copy is selected. For efficiency reasons, the node with the master copy should also coordinate all opens to the file (AN). In a previous section we argued that the AN should be the node with the highest reference rate to the file, so the largest number of open and lock requests can be handled locally.

As we found in Chapter 4, the algorithm that placed the file at the node with the largest cumulative I/O traffic had reasonable performance, even though it initially moved the file around unnecessarily. This suggests placing the master copy at the node with the highest reference rate (measured in bytes). Some additional rules are needed to avoid the potential instability during the first few references to the file.

As the main rule, a master copy can only be placed at a node that already has a copy. The unstable behavior previously mentioned is then avoided; the master copy will only move to a node that already has a copy, and any instability will not result in any network traffic. By default, the first node to reference a file has the master copy.

With these rules, a master copy will remain at the first node, unless a copy is created at another node. The first node is not necessarily the best static placement, and for files that are not copied, the master copy can remain in a non-optimal location. Initially, we considered a rule that could move the master copy to a new location even when the file was not replicated; if the master copy would have been erased had another copy of the file existed somewhere in the system, the rule would move the master copy to the node referencing the file. Given that the placement algorithm would have liked to erase the copy from its current position (if that had been feasible), the node currently referencing the file seemed a reasonable placement. The trace driven simulation did not support this conjecture, and the net result was an increase in network traffic. The rule, however, decreased the network traffic for sequential files, but only for the shorter values of d^*. The overall effect was an increase in the network traffic, and the rule was excluded from the final simulation study.

5.6.7. Description of the Policies Simulated

The various algorithms in the simulation are determined by the different estimation methods and assumptions we already have discussed. In particular, the simulation study is used to find the effect on performance of, 1) the three different methods for estimating reference rates, 2) the method that accounts for the uncertainty of the estimate, 3) the time horizon T that should be used in the condition for creating a copy (eq 5.4c), and 4) updating of copies versus invalidating.

In addition, we studied the effect of demand copying. With demand copying, a copy is always created (if necessary) at the node currently using the file. The current node is also assigned the master copy of the file. Demand copy algorithms are attractive, since they do not require any estimation of the reference rates. They are beneficial when most users reference the whole file or fraction of the file repeatedly, without any updates to the file. Three different demand copy algorithms were studied:
a) A demand copy algorithm where all copies accessed in the last d^* time units will be updated.
b) A demand copy algorithm where all copies are invalidated upon updates.
c) A demand copy algorithm which uses the same rule for updating copies as all the other algorithms we analyze.

The simulation study encompass four groups of algorithms, demand copy algorithms, algorithms using average rate , algorithms using normalized rate, and algorithms using the discounted rate. All algorithms use one common rule for erasing copies : copies that have not been accessed in the last d^* time units are erased. Three different values for d^* are used. The various algorithms in the simulation study are summarized in Table 5.1. The table displays the features of the different algorithms, the method used to estimate the reference rate, and the name used to identify the algorithm in the rest of the chapter.

5.7. Results for the Optimal Look-ahead Algorithm

We calculate the optimal look-ahead policy for three reasons:
1) To find the lower bounds on the performance of the algorithms in the trace driven simulation.
2) To provide hints of the desired features an algorithm should have.

Table 5.1: A summary of all the algorithms that are simulated. There are four groups of algorithms, demand copy algorithms, algorithms using average rate, algorithms using normalized rate, and algorithms using discounted rate. For all algorithms, copies that have not been accessed in the last d^* time units are erased

Group	name	description
Demand Copy algorithms	Update	All copies are updated. (dem1)
	Inv	When a file is updated all copies except the master copy are erased. (dem2)
	Selec. Inval.	The expected amount of updates are estimated based on a batch model. Only those copies where the expected update cost is less than the cost to recreate the copy will be updated (dem3)
Average Rate	Average	Equation 5.4 is used to determine when and where to create a copy. Updates are handled by the same way as for dem3 (av1)
	W Uncertainty	The algorithm av1 is augmented by the procedure outlined in section 5.5.2 for handling uncertainty of estimate (av2)
	W Cost	The previous algorithm is augmented by the cost of creating a copy in equation 5.4. This is discussed in section 5.5.4 (av3)
	W Invalid.	Same as algorithm av1 but with invalidation (av4)
Normalized Rate	Norm	The algorithm used equation 5.4 to determine when and where copies should be created. Only copies with an expected update cost smaller than the recreation cost are updated. The algorithm is the same as av1 but with a normalized rate. (norm1)
	W Invalid.	All copies except the master copy are invalidated when the file is updated. This algorithm is equivalent to av4 (norm2)
	W Cost W Inv	This is equivalent to av3 but with a normalized rate and invalidation. (norm3)
Discounted Rate	Discounted	This algorithm is equivalent to algorithms av1 and norm1 but with the discounted reference rate

3) To measure the potential advantage of replication versus migration or static placement of a single copy.

The optimal look-ahead policies are well suited to measure the potential advantage of replication, since the differences between the various optimal look-ahead policies are not distorted by ineffective implementation. They are therefore a direct measure of the advantage a particular scheme may offer.

This section is organized along the same lines. First the advantages of updating a copy versus invalidating are discussed. The look-ahead algorithms that allow updating will have a better performance that those limited to invalidation, since they may propagate the bytes that have been changed instead of always invalidating the additional copies. When each user is placed at a separate node, the difference between updating and invalidating the copies is marginal. If users from the same department are mapped to the same node, the difference between updating and invalidating increases. For one of the traces invalidating had at least 18% worse performance than updating.

The next sub-section discusses the advantage of replicating files versus migrating. For two of the traces, replication reduces the network traffic substantially (up to $\approx$50%), but at the cost of nearly a doubling of the file system. Finally, the features of the optimal look-ahead policy are discussed in more detail. We found that only a few files should ever be replicated.

As already discussed, the optimal look-ahead policy with replication can only be calculated if the master copy is fixed at one location. It is therefore necessary to judge the impact this constraint has on the optimal look-ahead policy.

5.7.1. Results with One User per Node

5.7.1.1. The Optimal Look-ahead Policy with Fixed Master Copy versus Optimal Look-ahead Policy.

Due to the computational complexity, we are only able to find the optimal look-ahead policy for files accessed by only a few users (up to three users). For the remaining files, an optimal look-ahead policy can only be found if a master copy is permanently placed at a node. The results for these files will therefore deviate from the "true" optimal look-ahead policy. In Table 5.3 we compare the difference in performance caused by the added constraint of a fixed master copy.

Table 5.2 Description of symbols

Symbol	Meaning
nt	network traffic measured in Giga bytes (10^6 K bytes)
n	network traffic in % of total I/O traffic
r	remote I/O traffic in % of I/O traffic
c	copy traffic in % of I/O traffic
u	update traffic in % of I/O traffic
sp	space time product in fraction of space time product for a single copy file system
s	update traffic to copies that are erased before they are accessed
e	remote traffic to nodes that eventually will have a copy
nc	number of copies
nfc	number of files that are replicated

For the files shared by at most three users, the difference in network traffic between the true optimal look-ahead and the optimal look-ahead with a fixed master copy is between 7% and 22%, which must be characterized as substantial. The difference between the two look-ahead algorithms is most likely smaller when they are compared for all files. The files shared by at most three nodes represent a substantial fraction of all the update traffic, and the cost for forcing the master copy to a node is larger if the file is frequently updated. The master copy is always updated, and if it cannot be moved to the node updating the file, the updates must be sent over the network. For the true optimal

look-ahead policy, which permits master copy migration, the network traffic can usually be avoided by moving the master copy to the node updating the file. For the trace from SLAC about 66% of all the update traffic is to files shared by at most three users, while for Huges it is 80%, and for Amdahl about 30%. For the traces from SLAC and Hughes, the difference for all files between the true optimal look-ahead, and the optimal with a fixed master copy will therefore most likely be smaller; the files that we compare (files shared by at most three users) represent a majority of the update traffic, and, as we have described, it is the handling of the update traffic that increases the network traffic for the optimal look-ahead algorithm with a fixed master copy.

Most of the files shared by at most three users are also sequential files, and they are often updated by overwriting the old contents. With a fixed master copy, the optimal action is to handle these overwrites as remote I/O when they are not issued by the node with the master copy. For the optimal look-ahead policy, these overwrites will usually result in a file move; repeated read references from the node can then be handled locally. This is reflected in the larger number of copies created by the true optimal look-ahead policy.

The difference between the optimal look-ahead with and without a fixed master copy is larger when the additional copies cannot be updated (i.e., with invalidation). With a fixed master copy, any updates from nodes other than the master node will result in remote traffic, since only the master copy can be updated.

The added constraint of a fixed master copy decreases the performance of the optimal look-ahead policy. However, it is the only feasible method for calculating the optimal look-ahead policy for all files. The reference pattern with a high fraction of updates is not as prevalent for files shared by more than three nodes, and the difference in performance may be smaller for these files. We therefore conclude it is acceptable to use the optimal look-ahead policy with a fixed master copy as a reasonable approximation of the optimal look-ahead policy. For simplicity, we will in the reminder of the chapter refer to this policy as the optimal look-ahead policy without the suffix "with a fixed master copy". Instead, we added the abbreviation WFMC (With Fixed Master Copy) to remind the reader whenever we refer to the optimal look-ahead policy with a fixed master copy.

Table 5.3a: Performance of the optimal look-ahead algorithm for the trace from SLAC with and without the added constraint of a fixed master copy. The results for the algorithm which always invalidate all additional copies when the file is updated are also included. Due to the computational complexity, only the results for files accessed by less than 4 users are included.

	Optimal lookahead with Fixed Master Copy for files accessed by less than 4 nodes		Optimal lookahead for files accessed by less than 4 nodes	
		With Invalidation		With Invalidation
$c_s = \frac{1}{8h}$				
nt	0.37	0.39	0.30	0.30
n	16.7	17.4	13.5	13.5
r	14.1	15.1	8.0	8.0
u	0.0	0.0	0.0	0.0
c	2.6	2.3	5.5	5.5
sp	1.0	1.0	1.0	1.0
nc	121	100	215	215
nfc	82	65	148	148
$c_s = \frac{1}{48h}$				
nt	0.35	0.37	0.29	0.29
n	15.6	16.5	13.0	13.0
r	13.3	14.3	7.8	7.8
u	0.1	0.0	0.0	0.0
c	2.2	2.1	5.2	5.2
sp	1.0	1.0	1.0	1.0
nc	116	99	205	205
nfc	92	74	148	148
$c_s = \frac{1}{length\ of\ trace}$				
nt	0.32	0.35	0.27	0.27
n	14.5	15.4	11.9	12.0
r	11.8	13.0	6.4	6.4
u	0.1	0.0	0.1	0.0
c	2.6	2.4	5.4	5.5
sp	1.1	1.1	1.1	1.1
nc	115	101	201	203
nfc	96	79	153	152

Table 5.3b: Performance of the optimal look-ahead algorithm for the trace from Hughes with and without the added constraint of a fixed master copy. The results for the algorithm which always invalidate all additional copies when the file is updated are also included. Due to the computational complexity, only the results for files accessed by less than 4 users are included.

	Optimal lookahead with Fixed Master Copy for files accessed by less than 4 nodes		Optimal lookahead for files accessed by less than 4 nodes	
		With Invalidation		With Invalidation
$c_s = \frac{1}{8h}$				
nt	1.25	1.27	1.13	1.13
n	16.3	16.5	14.6	14.7
r	13.1	13.9	10.6	10.7
u	0.6	0.0	0.0	0.0
c	2.6	2.6	4.0	4.0
sp	1.0	1.0	1.0	1.0
nc	332	284	638	641
nfc	239	185	468	464
$c_s = \frac{1}{48h}$				
nt	1.18	1.22	1.08	1.09
n	15.4	15.8	13.9	14.1
r	12.6	13.4	10.2	10.3
u	0.7	0.0	0.3	0.0
c	2.1	2.4	3.5	3.8
sp	1.0	1.0	1.0	1.0
nc	347	284	655	664
nfc	288	228	516	512
$c_s = \frac{1}{length\ of\ trace}$				
nt	1.16	1.20	1.06	1.08
n	15.0	15.6	13.8	14.0
r	12.3	13.3	10.0	10.1
u	0.8	0.0	0.4	0.0
c	1.9	2.3	3.3	3.9
sp	1.0	1.0	1.0	1.0
nc	355	296	701	714
nfc	304	242	576	571

Table 5.3c: Performance of the optimal look-ahead algorithm for the trace from Amdahl with and without the added constraint of a fixed master copy. The results for the algorithm which always invalidate all additional copies when the file is updated are also included. Due to the computational complexity, only the results for files accessed by less than 4 users are included.

	Optimal lookahead with Fixed Master Copy for files accessed by less than 4 nodes		Optimal lookahead for files accessed by less than 4 nodes	
		With Invalidation		With Invalidation
$c_s = \frac{1}{8h}$				
nt	1.37	1.38	1.20	1.20
n	13.0	13.2	11.4	11.4
r	11.3	11.8	9.3	9.4
u	0.2	0.0	0.0	0.0
c	1.5	1.4	2.1	2.1
sp	1.0	1.0	1.0	1.0
nc	203	175	324	318
nfc	136	113	236	231
$c_s = \frac{1}{48h}$				
nt	1.28	1.31	1.18	1.18
n	12.2	12.5	11.2	11.2
r	10.3	11.0	8.9	8.9
u	0.3	0.0	0.0	0.0
c	1.6	1.5	2.3	2.3
sp	1.0	1.0	1.0	1.0
nc	210	199	328	327
nfc	159	132	247	241
$c_s = \frac{1}{length\ of\ trace}$				
nt	1.13	1.17	1.06	1.07
n	10.8	11.2	10.1	10.2
r	8.2	9.4	7.6	7.8
u	0.5	0.0	0.0	0.0
c	2.0	1.8	2.5	2.4
sp	1.0	1.0	1.0	1.0
nc	231	222	341	338
nfc	192	162	277	265

5.7.1.2. Invalidation versus Updating

The performance of the optimal look-ahead policies (WFMC) is displayed in Figure 5.1. The optimal policy (WFMC) for all files is displayed in the left figure, while the optimal policy (WFMC) for sequential files is displayed in the right figure. A more detailed description of the performance is contained in Table 5.4. To illustrate different combinations of optimal (WFMC) network traffic and file system size, three different storage cost values are used.

From Figure 5.1 it is apparent that the difference between updating and invalidating can be small. The difference in network traffic is at most 16%, and it is decreasing with increasing storage cost. For the optimal look-ahead policy (WFMC) only a minute part of the network traffic is due to updates to copies (see Table 5.4), even though the update traffic constitutes from 9 to 30% of the total I/O traffic. This means that the optimal look-ahead policy (WFMC) only migrates a small fraction of file updates to the copies, and the copies are instead invalidated when the file is updated. This is reflected in the average number of copies of a file per node, given that the node has at least one copy during the trace. At the lowest storage cost, these averages are 1.20, 1.08, and 1.60 for the traces from SLAC, Hughes, and Amdahl. This can only be due to repeated invalidation / recreation of the copies.

Invalidation should be the preferred scheme for sequential files. For one of the traces, SLAC, we found that the optimal policy (WFMC) would never update copies of sequential files. Whenever a sequential file was updated, the additional copies were always erased. For the other two traces, the update traffic was insignificant. A large number of the updates to a sequential file overwrite the file, and it is better to invalidate and recreate the copy at the next reference from the node.

The total I/O traffic to shared files is dominated by the I/O traffic to partitioned files, and for partitioned files we expected invalidation to be inefficient. These files consist of several members, and the updates will usually be to individual members. Potentially, it can be beneficial to update the individual copies instead of invalidating them. In Table 5.4, this is reflected by higher network traffic for the policy with invalidation. However, partitioned files are not updated frequently enough to make a decisive increase in network traffic with invalidation. For the true optimal look-ahead policy (Table 5.3) the difference between invalidating and updating is at most 1%. Some of the difference we observe between updating and invalidating for the policy with a fixed master copy is therefore most likely due to the artificial constraint of a fixed master copy. Allowing the master copy to migrate is likely to reduce the difference between updating and invalidating.

We conclude that there is only a small potential gain (at most 16% for our traces) from updating copies instead of invalidating them. Whether copies are invalidated or updated will not influence the availability of the file system; the time average file system size remains nearly constant.

5.7.1.3. Potential Advantages of Replication

Replication versus migration of a single copy is advantageous for only two of the traces, SLAC and Hughes. However, replication is clearly advantageous compared to static placement of a single copy. Further, we found that replication is not advantageous for sequential files at all; migration of a single copy has a lower network traffic than replication with a fixed master copy.

Replicating files does not imply a large increase of the file system size. In the case where the storage cost is set to essentially zero, the largest increase in the time average file system size was 92%. This resulted in a 50% reduction in the network traffic compared to migration of a single copy (at SLAC). It should also be emphasized that we are only concerned with the file system size of shared files.

With the lowest storage cost, replication reduces the network traffic (compared to migration) by 50% for the SLAC trace, 22% for the Hughes trace, while it increases the network traffic by 1.6% for the Amdahl trace. The latter is due to the constraint of a fixed master copy, since the performance with replication should always be better than with migration alone; a policy with replication can behave as a migration policy by always

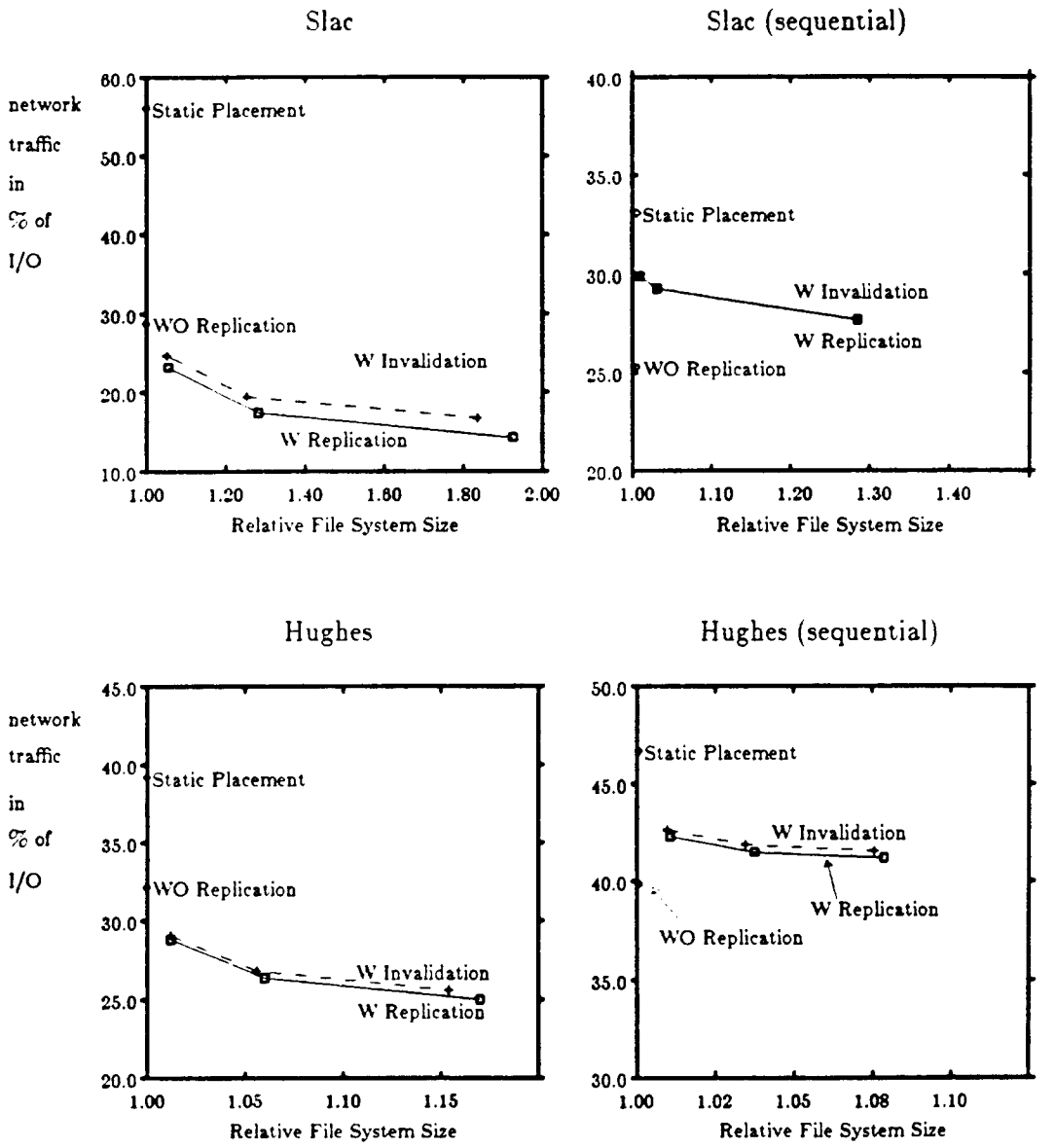


Figure 5.1: displays the performance of the different look-ahead policies (WFMC). The policies that are displayed are 1) static placement, 2) dynamic placement without replication, 3) placement with replication, and 4) placement with replication and invalidation. The left figure is for all files, while the right is for sequential files only. Note that the figures use different scales.

erasing the old copy each time a new copy is created. The behavior for the last trace is due to an anomaly in the traced data. For sequential and partitioned files, the reduction in network traffic is respectively 7% and 12% with replication. However, for that particular trace, 137 direct and index sequential files represents a majority of the I/O traffic, and

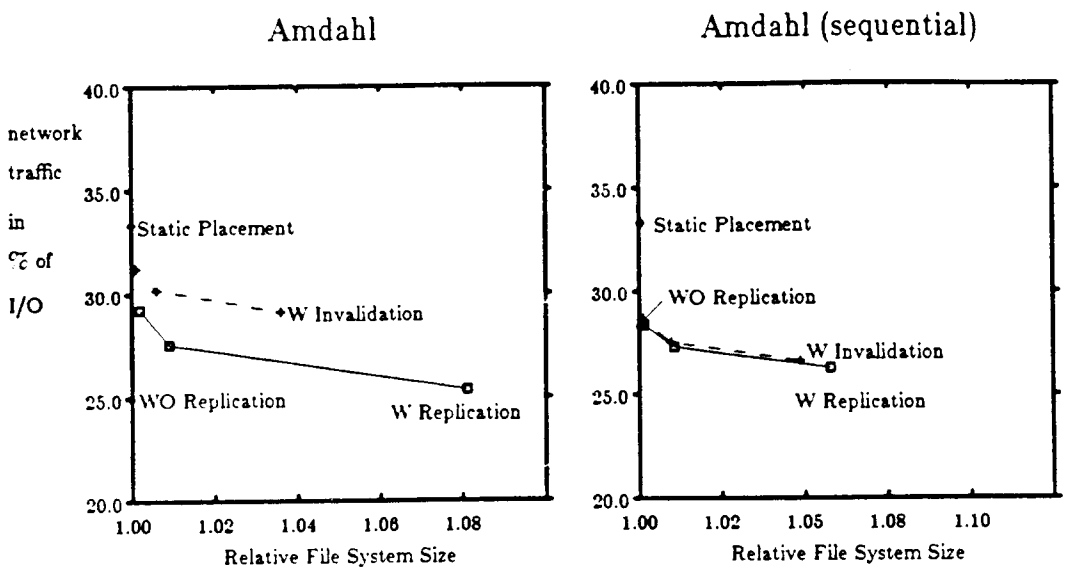


Figure 5.1 cont: displays the performance of the different look-ahead policies (WFMC). The policies that are displayed are 1) static placement, 2) dynamic placement without replication, 3) placement with replication, and 4) placement with replication and invalidation. The left figure is for all files, while the right is for sequential files only. Note that the figures use different scales.

for these files replication with a fixed master copy does not offer an advantage.

For two of the traces (SLAC and Hughes), replication reduces the network traffic for all three values of storage cost, so even a moderate increase in file system size can result in a reduction of network traffic. At the lowest storage cost, the increase in file system size varies between 92%, for the SLAC trace, and 17%, for the Hughes trace. With the highest storage cost, the increase in file system size is 5.5% for the SLAC trace and 1.7% for the Hughes trace, while the optimal look-ahead (WFMC) network traffic is reduced by 17% and 10% respectively. Replication therefore offers advantages even in systems that put a high premium on storage space.

Replication does not offer any advantage for sequential files. For two of the traces the optimal look-ahead policy (WFMC) with replication has higher network traffic than the optimal look-ahead policy without replication. This is, as previously explained, due to the added constraint of a fixed master copy. The amount of remote network traffic is high for the optimal look-ahead policy (WFMC) for sequential files. There are two reasons for this high fraction of remote I/O traffic (row labeled "r"). For sequential files many users are accessing the file only once to make their own copy of the file, and there is no difference between making a copy and referencing the file remotely. In this situation, the algorithms are set up to choose remote access as the preferred action. In addition, when sequential files are updated, they are mostly overwritten. With a single overwrite from a node other than the master copy, the preferred action is to do the update as a remote reference instead of a local reference combined with updating the master copy.

Table 5.4a: The optimal look-ahead policy (WFMC) for the trace from SLAC. Both the results with and without replication are presented.

Slac												
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Optimal Lookahead	nt	2.7	0.3	2.4	0.5	2.0	0.3	1.7	0.4	1.6	0.2	1.4	0.3
	n	23.2	29.9	22.8	24.5	17.4	29.3	16.5	18.5	14.3	27.7	13.2	15.2
	r	15.1	28.9	14.0	15.7	10.2	28.4	8.6	11.4	7.4	26.4	5.8	8.2
	u	0.1	0.0	0.1	0.0	0.3	0.0	0.3	0.1	0.6	0.0	0.7	0.3
	c	8.0	1.0	8.7	8.8	6.9	0.9	7.5	7.0	6.2	1.3	6.7	6.7
	sp	1.1	1.0	1.1	1.0	1.3	1.0	1.3	1.3	1.9	1.3	2.0	2.0
	nc	1481	238	1235	874	1111	162	940	631	924	148	767	577
	nfc	174	31	142	126	191	38	151	138	197	40	155	141
Optimal Lookahead w Invalidation	nt	2.8	0.3	2.6	0.5	2.2	0.3	2.0	0.4	1.9	0.2	1.7	0.3
	n	24.7	29.9	24.4	27.0	19.5	29.3	18.8	21.0	16.7	27.7	15.8	17.7
	r	17.2	28.9	16.2	19.0	12.8	28.4	11.5	14.6	10.4	26.4	9.1	11.4
	u	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	7.6	1.0	8.2	8.0	6.7	0.9	7.3	6.4	6.3	1.3	6.7	6.3
	sp	1.1	1.0	1.1	1.0	1.3	1.0	1.3	1.3	1.8	1.3	1.9	2.0
	nc	1428	235	1185	842	1089	158	922	614	931	144	778	569
	nfc	152	30	121	109	167	38	129	120	174	38	134	123

However, if the update is followed by read accesses, creating a local copy is the best decision.

The updating pattern to sequential files may explain why migration is as good as replication. For files that are updated frequently, little is gained by letting a copy remain at a node after a new node starts accessing the file; the current user may overwrite the file, and all other copies will have to be recreated. Replication will then offer no advantage compared with migration of a single copy.

We conclude that replication can potentially reduce the network traffic, but the reduction depends on the willingness to trade disk storage for network traffic. For sequential files, replication offers no advantages over migration of single copy. These results are the same whether the policy of invalidating or updating copies is used.

Table 5.4b: The optimal look-ahead policy (WFMC) for the trace from Hughes. Both the results with and without replication are presented.

		Hughes											
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
	nt	3.9	1.3	2.3	0.5	3.6	1.3	2.0	0.5	3.4	1.3	1.8	0.5
	n	28.8	42.3	27.1	28.5	26.4	41.5	23.5	26.1	25.0	41.2	21.4	25.5
Op-	r	23.2	38.4	19.9	23.5	21.1	37.2	17.0	20.9	19.2	36.7	14.3	20.4
timal	u	0.4	0.4	0.3	1.2	0.6	0.4	0.5	1.4	0.8	0.4	0.8	1.4
Looka-	c	5.2	3.5	6.9	3.8	4.7	3.9	6.0	3.9	5.0	4.1	6.4	3.7
head	sp	1.0	1.0	1.0	1.0	1.1	1.0	1.1	1.1	1.2	1.1	1.2	1.1
	nc	668	383	258	488	665	413	227	514	671	426	219	520
	nfc	311	218	79	264	368	261	89	318	390	275	95	333
	nt	3.9	1.3	2.3	0.5	3.6	1.3	2.1	0.5	3.5	1.3	1.9	0.5
Op-	n	29.1	42.6	27.3	29.2	26.8	41.9	24.0	27.0	25.6	41.6	22.1	26.4
timal	r	23.9	39.4	20.2	26.0	21.9	38.3	17.4	23.8	20.4	37.6	15.4	23.3
Looka-	u	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
head w	c	5.2	3.3	7.0	3.2	4.9	3.6	6.5	3.3	5.2	4.0	6.8	3.1
Invali-	sp	1.0	1.0	1.0	1.0	1.1	1.0	1.1	1.1	1.2	1.1	1.2	1.1
dation	nc	582	318	255	401	589	356	228	432	599	376	218	444
	nfc	251	178	72	210	301	217	81	256	321	230	87	270

5.7.1.4. Features of the Optimal Look-ahead

Under the optimal look-ahead (WFMC) policy, only few files are replicated at all. Table 5.5 summarizes the number of files replicated and the number of copies made.

Table 5.4c: The optimal look-ahead policy (WFMC) for the trace from Amdahl. Both the results with and without replication are presented.

Amdahl													
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Optimal Look-ahead	nt	10.0	0.6	4.8	0.8	9.4	0.6	4.5	0.7	8.6	0.6	3.9	0.6
	n	29.3	28.4	27.7	21.4	27.5	27.3	25.9	19.8	25.4	26.3	22.5	17.3
	r	22.6	24.4	23.9	18.0	19.5	22.7	21.9	16.0	15.9	20.9	16.6	12.9
	u	2.5	0.4	0.6	0.8	3.1	0.2	0.8	1.0	4.2	0.4	1.8	1.4
	c	4.1	3.5	3.3	2.6	4.9	4.4	3.2	2.8	5.2	4.9	4.1	3.0
	sp	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.1	1.1	1.1	1.1
	nc	745	162	502	509	790	190	505	531	828	202	517	539
	nfc	301	83	189	211	342	95	214	239	402	114	245	282
Optimal Look-ahead w Invalidation	nt	10.6	0.6	4.9	0.8	10.3	0.6	4.7	0.8	9.9	0.6	4.4	0.7
	n	31.2	28.5	28.4	21.9	30.2	27.5	27.2	20.6	29.1	26.6	25.5	18.7
	r	29.1	24.9	25.7	19.3	26.8	23.2	24.1	17.8	25.3	21.6	21.9	15.7
	u	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	2.1	3.7	2.7	2.5	3.4	4.4	3.1	2.8	3.8	5.1	3.7	3.0
	sp	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.1
	nc	631	152	437	470	739	177	504	525	829	198	559	553
	nfc	249	73	159	182	289	85	181	206	341	104	204	246

Table 5.5: The table summarizes the number of copies created by the optimal look-ahead policy (WFMC) for the three traces. The third row tabulates the number of copies created in addition to the master copy. The second row gives the total number of instances of a node using a file (node/file pairs) found in the traces, while the fifth row tabulates the number of these node/file pairs that will have a local copy. The fourth row tabulates the number of files that are replicated.

	SLAC	Hughes	Amdahl
No files	427	5666	2485
No node/file pairs	2043	12259	8179
No copies made	924	671	1081
No files replicated	197	390	402
No node/file pairs with copy in addition to master copy	768	619	677

The optimal look-ahead policy (WFMC) replicates only a few files, but the average number of additional copies per replicated file varies from 4.7 for the trace from SLAC to 1.7 for the trace from Hughes. As already discussed, the amount of update traffic to the

copies is small and only a few copies are updated.

5.7.2. Results with One Department per Node

To judge the effect on the results from the schemes used to place users at nodes, the optimal look-ahead policy was also calculated when users from the same department were allocated to the same node. A user's department could only be identified for the traces from SLAC and Amdahl. Briefly summarized, the optimal look-ahead policies retain the same features. However, the network traffic is drastically reduced. This confirms our conjecture that most sharing would be within departments.

Allocating users from the same department to the same node does not affect the difference between the optimal look-ahead policy and the optimal look-ahead policy with a fixed master copy. The difference is still around 10 to 20%, and the optimal policy creates a larger number of copies (see Table 5.6) than the policy with a fixed master copy. The two optimal policies have nearly the same time average file system size, and the difference in the number of copies is most likely due to the migration of the master copy.

For brevity, only the difference between the two policies for the lowest storage cost is displayed in Table 5.6. The optimal policy is calculated for all files shared by less than 4 departments. These files represent 62% of all I/O traffic to shared files, 95% of all update traffic, and 66% of all department/file pairs for the trace from SLAC. The same numbers from the trace from Amdahl is 73% of the I/O traffic, 71% of the update traffic, and 82% of all department/file pairs. Because the deviations between the true and the optimal look-ahead policy (WFMC) are small, the policy with a fixed master copy can be used as an approximation of the true optimal look-ahead policy.

Table 5.6a: Performance of the optimal look-ahead algorithm for the trace from SLAC with and without the added constraint of a fixed master copy. The policies are calculated with users from the same department placed at the same node. Due to the computational complexity, only the results for files accessed by less than 4 users are included.

	Optimal lookahead with Fixed Master Copy for files accessed by less than 4 nodes		Optimal lookahead for files accessed by less than 4 nodes	
		With Invalidation		With Invalidation
$c_s = \frac{1}{length\ of\ trace}$				
nt	0.17	0.22	0.16	0.19
n	2.4	3.1	2.2	2.7
r	0.8	1.7	0.7	1.0
u	0.5	0.0	0.3	0.0
c	1.1	1.4	1.3	1.7
sp	1.1	1.1	1.1	1.1
nc	89	92	133	146
nfc	61	54	83	83

Invalidating versus Updating. Mapping users from the same department has some effect on the difference between invalidating and updating (see Figure 5.2). The difference measured in percent remains nearly the same as with one user per node ($\approx$ 20%), and the difference is small (at most equivalent to 2.2 % of the I/O traffic). For sequential files, allowing updating instead of invalidating has no effect on network traffic.

Table 5.6b: Performance of the optimal look-ahead algorithm for the trace from Amdahl with and without the added constraint of a fixed master copy. The policies are calculated with users from the same department placed at the same node. Due to the computational complexity, only the results for files accessed by less than 4 users are included.

	Optimal lookahead with Fixed Master Copy for files accessed by less than 4 nodes		Optimal lookahead for files accessed by less than 4 nodes	
		With Invalidation		With Invalidation
$c_s = \frac{1}{length\ of\ trace}$				
nt	2.50	3.00	2.02	2.06
n	10.0	12.0	8.1	8.2
r	4.1	10.6	3.5	3.7
u	3.2	0.0	0.1	0.0
c	2.7	1.4	4.5	4.5
sp	1.0	1.0	1.0	1.0
nc	152	133	233	233
nfc	123	107	179	175

For the true optimal policy, the difference between updating and invalidating is 2% and 18% (the latter number is for the trace from SLAC). With a department mapping, the difference between updating and invalidating is larger than the difference observed with one user per node. The potential usefulness of updating is therefore larger under this mapping rule.

Potential Advantage of Replication. Replication is even more advantageous with one department per node. For the trace from Amdahl, we previously found that the restriction of a fixed master copy made replication less advantageous than migration. With one department per node, the network traffic is comparable. For sequential files, replication still does not offer any advantages compared with migration of a single copy. An extended locality does not seem to exist for these files. To summarize, replication is still advantageous, and it will not result in a large increase in file system size. The largest increase we observed was 40% of the time average file system size.

Features of the Optimal Look-ahead Policy. Only a few of the files are replicated. At the lowest storage cost, only 95 out of 427 files are replicated for the trace from SLAC, while 384 files out of 2485 files are replicated for the trace from Amdahl. Under this policy, the update traffic is a larger fraction of the network traffic, 11% for the trace from SLAC and 23% for the trace from Amdahl. With the department mapping, copies are referenced frequently enough to justify updating them.

5.8. Simulation Results

This section represents the simulation results for the different estimation methods. We find that the performance is not sensitive to the estimation method used, but an estimation method is needed, since demand copying degrades performance. In an environment where disk storage is at a premium, the normalized rate has the best performance. However, the performance is most sensitive to the estimate of the expected lifetime of a

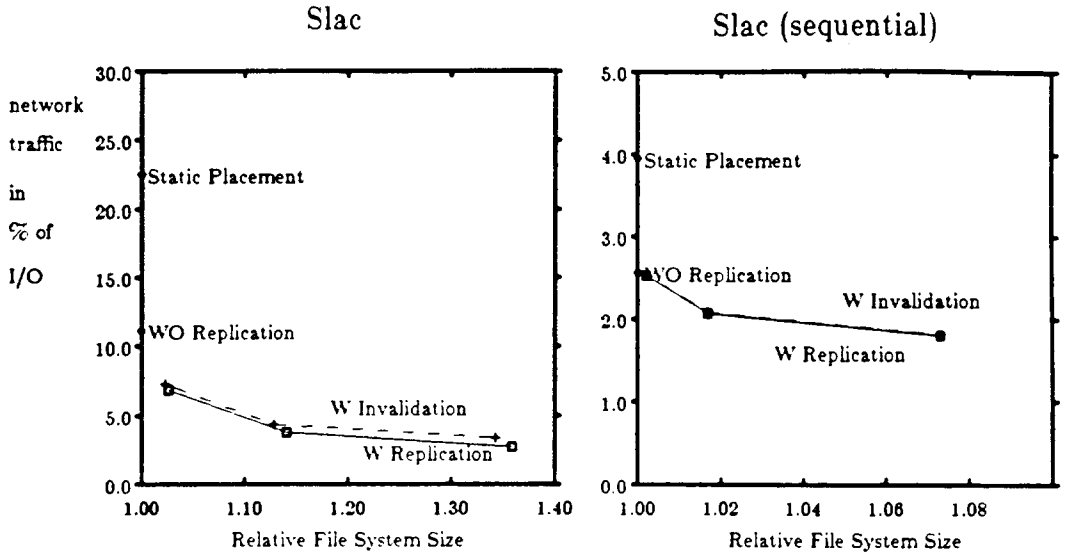


Figure 5.2a: displays the performance of the different look-ahead policies (WFMC) when users from the same department are placed at the same node. The policies that are displayed are 1) static placement, 2) dynamic placement without replication, 3) placement with replication, and 4) placement with replication and invalidation. The left figure is for all files, while the right is for sequential files only. Note that the figures use different scales.

copy. The expected lifetime is used to amortize the cost of creating a copy over the expected time the copy will exist, and the algorithms that include this cost element have a better performance than those that assume that the amortized cost can be neglected (i.e., those that assume the expected lifetime is ∞). The scheme used to account for the uncertainty of the estimates worked, and reduced the network traffic. With one user per node, invalidation of all additional copies each time a file is updated results only in a small increase in the network traffic, and invalidation is recommended as the best scheme. However, with users from the same department mapped to the same node, the difference in network traffic between updating and invalidating is $\approx$ 30%. A similar behavior was seen in the analysis of the look-ahead policies.

The final part of this section analyzes the difference between the proposed algorithms and the look-ahead algorithm. For two of the traces, the major part of the difference is due to remote references while the algorithms are estimating the various parameters. However, demand copying is not a suitable solution. The difference between the decisions made by the proposed algorithms and the decisions made by the look-ahead algorithms are from 0 to 50%. There will always be a difference in performance between placement decisions of the best stochastic algorithm and those of the optimal look-ahead algorithm. To judge the magnitude, we used a simulation model where the best stochastic decisions could be calculated. All the observed differences are within the range that should be expected between a realizable algorithm and a look-ahead algorithm, and we believe the quality of the proposed algorithms is reasonable.

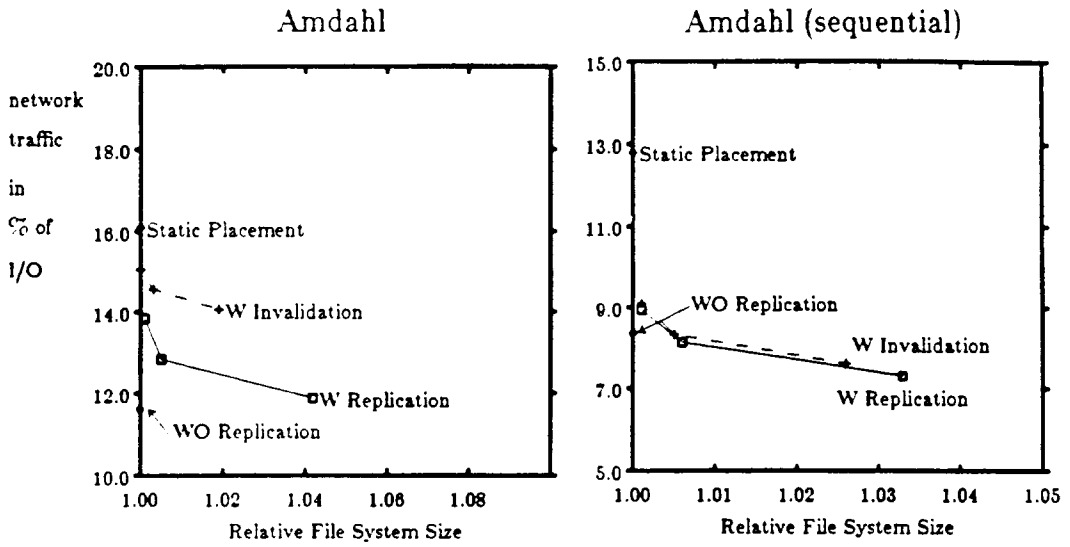


Figure 5.2b: displays the performance of the different look-ahead policies (WFMC) when users from the same department are placed at the same node. The policies that are displayed are 1) static placement, 2) dynamic placement without replication, 3) placement with replication, and 4) placement with replication and invalidation. The left figure is for all files, while the right is for sequential files only. Note that the figures use different scales. For sequential files, replication has a worse performance than with migration alone. The optimal look-ahead placement with replication is calculated with a fixed master copy. Since so few sequential files are replicated, the disadvantage of a fixed master copy will be larger than the advantage of replication, and migration alone will have a better performance.

5.8.1. Results with One User per Node

The performance of some of the algorithms is summarized in Figures 5.3, 5.4, and 5.5. A more detailed view of the performance for the different algorithms is found in Tables 5.7, 5.8, and 5.9.

5.8.1.1. Results for Demand Copy Algorithms

The demand copy algorithms create a copy at all nodes that reference a file. This is only beneficial if most of the nodes access either the whole file at least once or parts of the file repeatedly. The results show that demand copying is not advisable in the environments represented in the traces; users typically access only a fraction of a file a few times.

Three different versions are compared, and they differ in the rules for erasing copies. We compare the two extreme algorithms of always updating and always invalidating the additional copies when the file is updated. The third algorithm uses a selective updating scheme that will invalidate some copies, while the remaining ones are updated. A more detailed description of the algorithms is found in section 5.5.

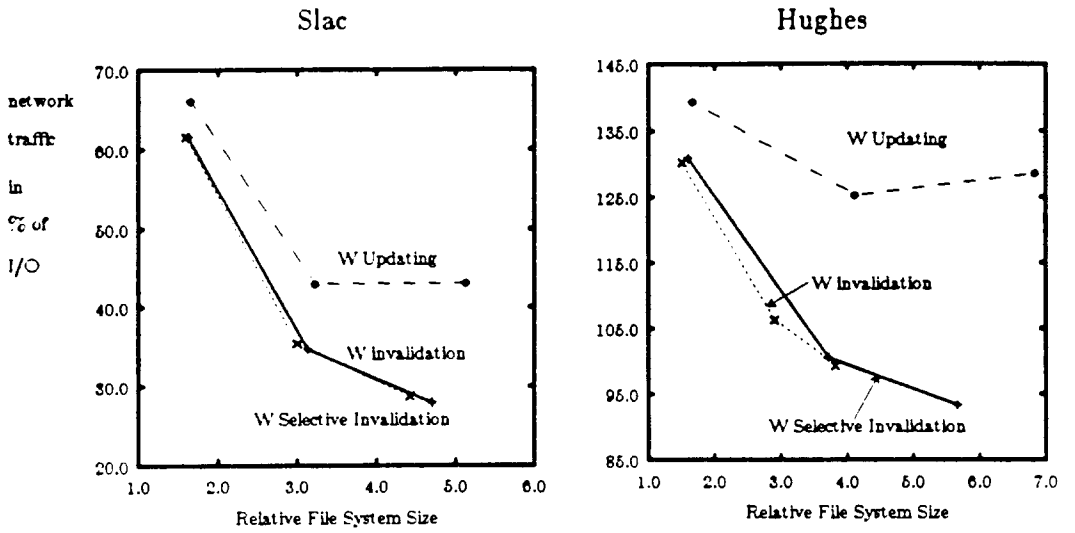


Figure 5.3: The performance for the three version of demand copy algorithms for the traces from SLAC and Hughes. The results are with one user per node. The figures have different scale to improve the graphic presentation.

The performance of the demand copy algorithms is summarized in Table 5.7.a, b, and c and in Figure 5.3, where we see that performance varies between the traces. For two of the traces, the network traffic exceeds the total I/O traffic, while for the third, SLAC, the performance is similar to the best single copy algorithm.

The high network traffic is mostly due to creating copies at nodes that will access only a part of a file at most a few times. The version with the selected updating scheme has the best overall performance, and the selection criterion seems to be reasonable.

The extended locality is not large, i.e., there are few nodes that will randomly reference a file during an interval. One node will reference the file, followed by another node, and so on with few repetitions of the cycle. This is reflected in the update traffic. Most of the update traffic is to copies that will be erased before they are referenced again (row s in Table 5.7). This means that once a different node starts updating the file, few of the users that have referenced the file so far will do so again.

Potentially, algorithms with invalidation could have a better performance, since most of the update traffic is wasted. This is true for the traces from SLAC and Hughes, while for Amdahl, invalidation has a worse performance. At Amdahl, most updates are small compared to the size of the file. The mean size of the replicated files is 9.8M bytes, while the average update size is 338K bytes (with the highest storage cost). Under these conditions invalidation of the copies in the extended locality is costly, and this is reflected in the poor performance of the algorithm with invalidation.

The version with selective update scheme has the best performance at all three installations. The scheme will update copies only when the expected update cost is smaller than the cost of recreating the copy. Otherwise, the copy is invalidated. With

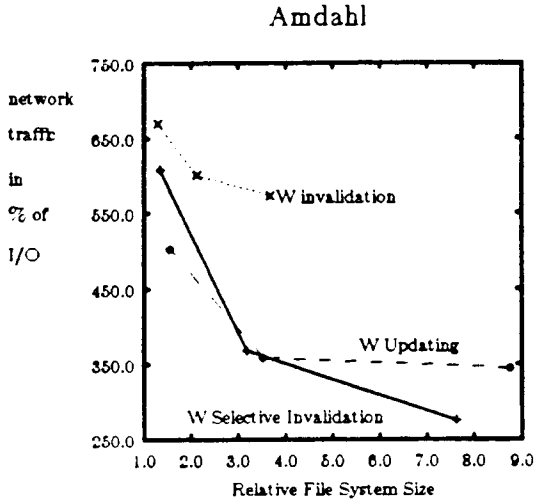


Figure 5.3 cont.: The performance for the three version of demand copy algorithms for the trace from Amdahl. The results are with one user per node.

this scheme the update traffic is low for the trace from SLAC, which implies most of the copies are invalidated, while the update traffic is high for Amdahl, which is the correct scheme for that trace. The selection criterion therefore seems to work as intended.

5.8.1.2. Performance of the Various Methods for Estimating the Reference Rate

Figure 5.4 displays the performance of the three estimation methods, the average rate, the normalized rate, and the discounted rate. Tables 5.8 and 5.9 contain a more detailed description of the performance of the different methods.

We found only small differences between the three estimation methods. Only for the highest storage cost is there a difference; the performance with the normalized rate is then better than with either the average or the discounted rate. With the normalized rate, more copies are created, which results in a net reduction in the network traffic. The normalized rate is therefore recommended for installations where storage space is a limited resource.

The procedure for accounting for the uncertainty of the estimate resulted in an improvement in the performance for all three traces; both the network traffic and the file system size were reduced. It appears that the benefit of creating a copy is overestimated during the first few opens to the file, since references to files tend to be clustered. The average reference rate calculated over the first few opens will be higher than the long term average, and the benefit of creating a copy will be overestimated. Thus, the procedure that accounts for the uncertainty is worth implementing.

However, the factor with the largest impact on the performance is the estimate of a copy's lifetime. Originally we thought the lifetime would be so large that the cost of creating a copy could be neglected. Such an assumption resulted in a high network traffic

Table 5.7a: The performance of the three different demand copy algorithms for the trace from SLAC. The results are with one user per node. The explanation of the row labels are found in Table 5.2.

Slac													
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Demand Copy	n	66.0	97.9	63.1	58.4	43.1	101.0	38.1	42.8	43.2	108.2	37.2	49.3
	r	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	u	9.8	58.4	5.8	6.3	18.1	75.7	13.3	18.0	26.5	87.4	21.0	33.5
	s	6.6	40.3	3.9	4.7	8.4	46.4	5.3	11.0	10.7	48.4	7.1	14.1
	c	56.2	39.6	57.3	52.0	25.0	25.3	24.8	24.7	16.7	20.8	16.3	15.8
	e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	sp	1.66	1.17	1.74	1.69	3.33	1.85	3.57	3.54	5.13	2.54	5.55	5.68
Demand Copy Selective Updating	n	61.6	63.4	61.1	55.1	34.7	56.0	32.7	30.7	27.9	53.4	25.6	24.5
	r	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	u	2.1	15.6	0.9	1.3	4.0	19.1	2.6	3.3	5.2	20.2	3.9	4.4
	s	1.5	10.6	0.7	1.1	2.8	13.6	1.9	2.3	3.3	14.1	2.4	2.7
	c	59.6	47.8	60.2	53.8	30.7	36.9	30.1	27.4	22.7	33.3	21.7	20.1
	e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	sp	1.62	1.10	1.70	1.67	3.13	1.58	3.37	3.35	4.71	1.98	5.12	5.05
Demand Copy Invalidation	n	61.6	55.7	61.6	54.7	35.4	45.8	34.1	30.7	28.7	43.0	27.1	24.3
	r	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	61.6	55.7	61.6	54.7	35.4	45.8	34.1	30.7	28.7	43.0	27.1	24.3
	e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	sp	1.60	1.05	1.68	1.65	3.00	1.46	3.24	3.22	4.42	1.77	4.82	4.77

caused by creating copies at most nodes referencing the file more than once.

5.8.2. Effect of Estimating the Lifetime of a Copy

In the algorithms with the label "W Cost", an estimate of a copy's lifetime is maintained, and the cost of creating a copy is amortized over this expected lifetime. In Section 5.5 we argued why it is reasonable to use the time over which a node has referenced the file as an estimate of the remaining time the node will continue to use the file. For algorithms that invalidate, the expected remaining lifetime is modified to account for the expected time until the next update of the file.

Including the cost for creating a copy ($T \neq \infty$) results in a reduction of the number of copies created and an increase in the remote I/O traffic. Compared to the other algorithms, the net result is a substantial decrease in the file system size and a decrease in the total network traffic (about 30%). With this scheme, a copy is only created at nodes where either the expected gain is large, or at nodes that have referenced the file for a longer period. The latter is only an advantage if the age of a node's reference process is a good indicator that the node will continue to reference the file. Note that the age of the reference process is only used for nodes that continue to open the file; the condition for creating a copy is evaluated only when the node opens the file. The combination of an "old" reference process that is still referencing the file is a good indicator that the node will reference the file again.

Table 5.7b: The performance of the three different demand copy algorithms for the trace from Hughes. The results are with one user per node. The explanation of the row labels are found in Table 5.2.

		Hughes											
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Demand Copy	n	139.4	69.5	181.2	69.8	125.3	106.8	144.5	60.0	128.7	122.3	140.8	59.4
	r	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	u	20.7	16.8	20.0	7.3	47.9	59.5	42.5	14.3	67.7	77.2	63.2	20.1
	s	17.2	15.7	16.1	5.3	33.5	57.5	29.5	8.6	44.3	71.5	38.5	12.7
	c	118.7	52.6	161.2	62.5	77.3	47.3	102.0	45.7	61.0	45.1	77.6	39.3
	e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	sp	1.67	1.21	1.88	1.34	4.12	2.34	5.02	2.33	6.84	3.37	8.64	3.38
Demand Copy Selective Updating	n	130.9	55.5	175.7	66.8	100.6	51.6	131.8	52.7	93.3	50.0	120.9	47.6
	r	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	u	6.8	2.6	8.8	2.4	16.2	3.6	21.9	4.5	21.5	4.1	29.7	5.7
	s	5.6	2.2	7.1	1.9	11.2	3.0	15.6	3.3	15.3	3.2	21.7	4.1
	c	124.1	53.0	166.9	64.3	84.4	47.9	110.0	48.2	71.8	45.9	91.3	41.9
	e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	sp	1.60	1.20	1.81	1.31	3.72	2.22	4.59	2.23	5.66	3.14	7.15	3.18
Demand Copy Invalidation	n	130.3	53.5	176.1	66.0	106.3	48.7	139.8	52.6	99.3	47.1	129.4	47.5
	r	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	130.3	53.5	176.1	66.0	106.3	48.7	139.8	52.6	99.3	47.1	129.4	47.5
	e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	sp	1.51	1.19	1.68	1.28	2.91	2.18	3.49	2.09	3.82	3.07	4.62	2.93

Whether copies are updated or invalidated affects the performance only slightly for two of the traces, with difference from 3 to 5%. For the last trace (Amdahl), the difference is 22%. This large increase is due to larger amount of remote I/O traffic for the algorithm that invalidates.

For the optimal algorithm, the network traffic would decrease with decreasing storage cost, since the algorithms should be more willing to create copies to avoid network traffic. For both the traces from Hughes and Amdahl, the opposite is observed; the network traffic is increasing with decreasing storage cost. The remote I/O traffic is reduced, while the update traffic and the file transfer traffic are increased. The net result is higher network traffic. This is probably caused by a highly clustered reference pattern with few repeated clusters from a node. As the storage cost decreases, the reference rate will be high enough to make creating a copy beneficial. However, many of these nodes will not continue to reference the file (at least not before it is updated), and transferring of a copy to the node results in a performance loss. Amortizing the transfer cost over the time a node has referenced a file in equation 5.4c counteracts this tendency. A copy will only be created at nodes that have referenced the file for a while, and the algorithm avoids creating copies at nodes that will reference the file a few times within a short period.

To conclude, the performance of the algorithms was improved when the cost of creating a copy was included in the decision criterion. We used the age of a node's reference process as an estimate of the remaining time the node would continue to reference the file. This is a filtering process which creates a copy only at nodes where either the potential gain is large or at nodes that have referenced the file for a long time. The latter

Table 5.7c: The performance of the three different demand copy algorithms for the trace from Amdahl. The results are with one user per node. The explanation of the row labels are found in Table 5.2.

Amdahl													
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Demand Copy	n	502.7	69.2	626.9	96.8	358.0	56.8	385.6	78.5	344.8	55.0	280.8	89.8
	r	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	u	51.3	2.4	22.9	9.1	112.9	5.9	57.1	22.6	222.3	13.9	118.2	59.6
	s	43.6	1.7	18.2	6.8	61.8	3.5	32.0	14.1	105.5	7.3	64.1	37.0
	c	451.4	66.8	604.1	87.7	245.1	50.9	328.5	55.9	122.5	41.1	162.6	30.2
	e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	sp	1.55	1.04	1.46	1.07	3.53	1.28	3.16	1.35	8.76	1.91	8.15	2.56
Demand Copy Selective Updating	n	607.9	69.6	664.5	97.1	387.5	54.2	404.0	71.4	275.1	47.8	253.9	54.4
	r	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	u	14.0	0.8	8.3	2.8	69.2	1.9	38.5	8.2	124.9	4.3	81.0	15.8
	s	11.8	0.4	6.5	1.9	36.3	0.9	22.0	5.1	57.4	2.5	38.7	9.0
	c	594.0	68.9	656.2	94.3	298.3	52.3	365.5	63.1	150.2	43.5	172.9	38.6
	e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	sp	1.39	1.04	1.40	1.05	3.17	1.26	3.00	1.29	7.62	1.79	7.68	2.15
Demand Copy Invalidation	n	668.8	69.8	676.3	99.8	600.4	54.3	548.1	80.1	573.3	47.7	500.3	66.5
	r	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	668.8	69.8	676.3	99.8	600.4	54.3	548.1	80.1	573.3	47.7	500.3	66.5
	e	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	sp	1.30	1.04	1.36	1.05	2.12	1.24	2.34	1.21	3.68	1.66	4.24	1.75

is a good indication the node will continue to reference the file. To obtain satisfactory performance, such a criterion should be implemented, since it substantially reduces both the average file system size and the network traffic.

5.8.2.1. Invalidating versus Updating

For the optimal look-ahead policy, the performance with invalidation is worse than with updating. The same is not true for the algorithms in the simulation. For two of the traces, some the algorithms that invalidate actually have a better performance than the ones that update. There are two reasons for this: 1) Most of the update traffic is to copies that are never accessed again (see row "s" in Tables 5.8 and 9), and the traffic is therefore wasted. A copy is only updated if the expected cost is less than the cost to recreate the copy. We are, however, not able to identify in advance the nodes that will not reference the file again, and updates will therefore be directed to nodes that may not use the file again. 2) With invalidation, an estimate of the expected time until a copy will be invalidated is maintained. The cost of creating a copy is amortized over this expected lifetime. In the previous section, we saw that including this amortized cost improved the performance. The effect of maintaining an estimate for the expected time until invalidation will have a similar effect.

If we compare the two algorithms that account for the cost of creating a copy (with label "W Cost") in Figure 5.4, the one that invalidates has a higher network traffic than the one that updates. This is in spite of the large fraction of updates to copies that will never be accessed before they are erased. The mechanism for updating files therefore

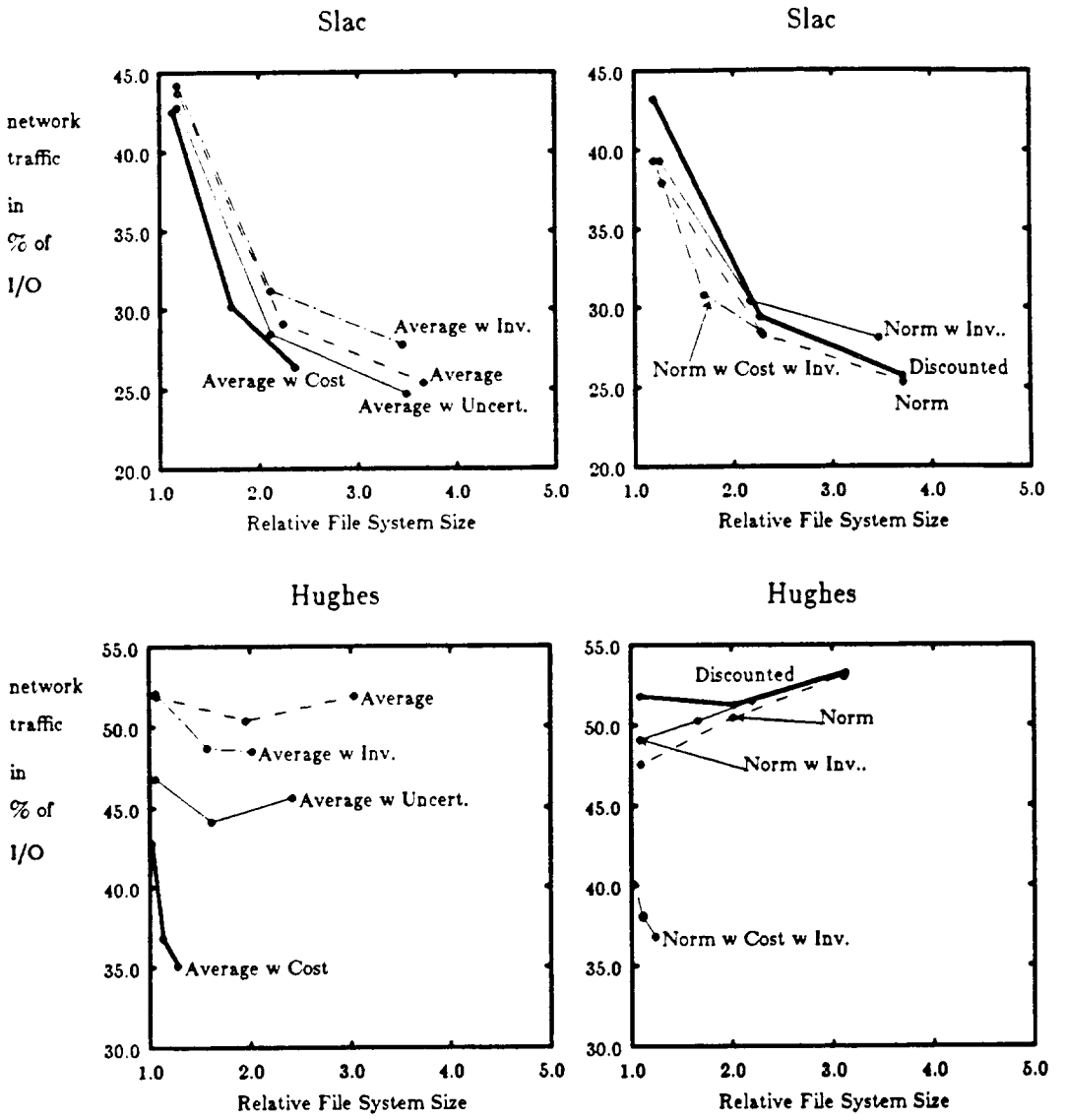


Figure 5.4: displays the performance of the various estimation methods for the traces from SLAC and Hughes. For two of the methods the results with and without invalidation are included. The results are with one user per node.

seems to target the right copies for updates. As already mentioned, for two of the traces the difference between updating and invalidating is only from 3% to 5%. For the last trace, Amdahl, invalidating increases the network traffic by 22%. A similar increase also applies to partitioned files.

Most of the increase is due to an increase in the remote I/O traffic. In the analysis of the look-ahead algorithms, we found that for this trace the typical update is small compared with a typical copy size, roughly 3%. However, for the invalidation algorithms, the

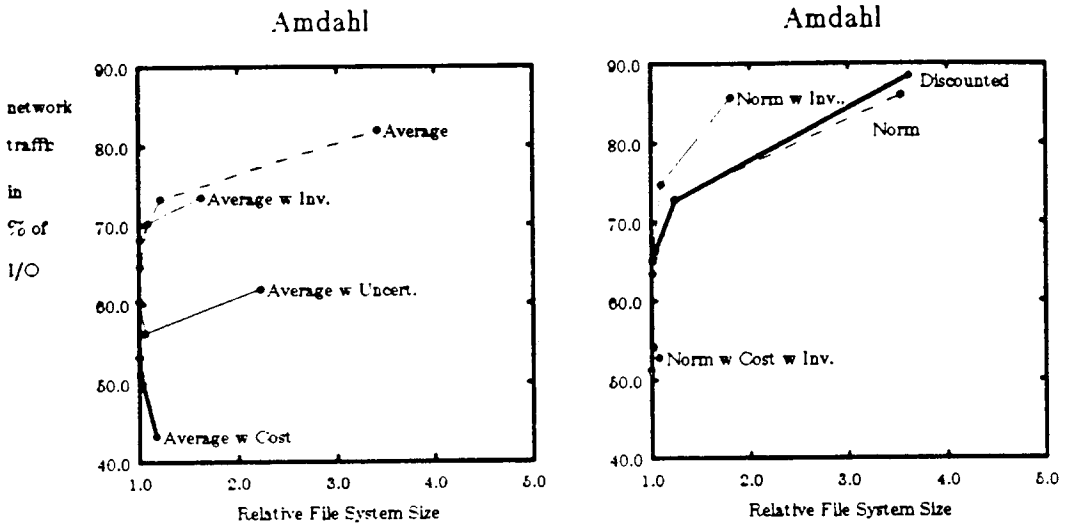


Figure 5.4 cont: displays the performance of the various estimation methods for the trace from Amdahl. For two of the methods the results with and without invalidation are included. The results are with one user per node.

size of the update is not important. The important factor is time between updates, since this affects the time horizon, T used in equation 5.4c. With small but frequent updates, the expected lifetime will be too short to justify creating a copy, and the amount of remote traffic will increase.

To summarize, for two of the traces invalidating results in a performance comparable to updating, and invalidation is the preferred scheme. For the last trace, Amdahl, the difference is about 22 %, and invalidation is no longer the obvious choice.

5.8.2.2. Results for Different File Organizations

For sequential files there is only a small difference in performance for all the various algorithms we have analyzed (see Tables 5.8 and 5.9 and Figure 5.4). Both invalidation and demand copying are appropriate schemes. However, the actions of the various algorithms are quite different. The ones that account for the cost of creating a copy have nearly 50% lower traffic due to transfers, but the total traffic is nearly constant. These algorithms are still preferred, since they result in the smallest average file system size.

The optimal look-ahead policy indicated that sequential files should be invalidated instead of being updated. This is confirmed in our simulation. The difference between updating and invalidating is less than 1% of the I/O traffic.

The demand copy algorithms have a reasonable performance for sequential files compared to the best algorithms. For the lowest storage cost, the network traffic is lower for one trace (47% versus 49% for the trace from Hughes), comparable for another (43% versus 39% for the trace from SLAC), while substantially worse for the last trace (47% versus 35%). A reasonable performance was expected because a larger fraction of opens to sequential files referenced the whole file. Always creating a copy will then be

advantageous, at least for the lowest storage cost. As the storage cost increases, a copy must be repeatedly referenced to justify its existence, and demand copying is then less desirable.

To summarize, we found that for sequential files invalidating instead of updating the copies is the preferred scheme for the algorithms that replicate files. However, as we will discuss in the next section, for sequential files replication is not advantageous compared to migration; the network traffic with migration is lower or equal to the one obtained with replication. Migration is therefore the preferred scheme for sequential files.

Table 5.8a: The table displays the results for the average estimation methods for the trace from SLAC with one user per node. The results are analyzed for all shared files, sequential, partitioned, and small (< 1M bytes) files. The results for large files are similar to the results for all files, since the behavior of these files dominates the performance of the small files. The symbols used are explained in table 5.2 The row labeled "n" tabulates the total network traffic, while the rows labeled "r, u, and c" tabulate the traffic owing to remote opens, updates, and transfers of files. All network traffic is measured in percent of total I/O traffic. The row labeled "sp" tabulates the time average file system size.

Slac													
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Average Rate	n	43.7	48.7	43.4	42.5	29.1	42.1	28.1	27.0	25.4	39.6	24.3	23.4
	r	33.4	47.2	32.5	33.2	15.6	39.7	13.6	15.1	11.2	37.3	9.0	10.7
	u	0.2	0.0	0.3	0.1	1.2	0.0	1.3	0.7	2.1	0.1	2.3	1.2
	s	0.2	0.0	0.2	0.1	0.8	0.0	0.9	0.5	1.1	0.0	1.2	0.7
	c	10.0	1.5	10.7	9.2	12.3	2.4	13.2	11.2	12.1	2.2	13.0	11.5
	e	9.8	1.3	10.5	7.4	6.6	1.8	7.1	5.7	5.0	1.6	5.3	5.0
	sp	1.19	1.01	1.22	1.11	2.24	1.13	2.41	2.21	3.66	1.66	3.97	3.89
Average W Uncertainty	n	42.8	48.5	42.5	41.8	28.5	42.1	27.4	26.4	24.7	39.6	23.6	22.8
	r	33.7	47.0	32.8	33.4	16.4	39.8	14.5	15.6	11.6	37.3	9.5	11.1
	u	0.2	0.0	0.3	0.1	1.2	0.0	1.3	0.7	2.1	0.1	2.2	1.2
	s	0.1	0.0	0.2	0.1	0.8	0.0	0.9	0.5	1.1	0.0	1.2	0.7
	c	8.8	1.5	9.4	8.4	10.9	2.3	11.6	10.1	11.0	2.2	11.8	10.5
	e	10.1	1.3	10.9	7.5	7.0	1.8	7.5	5.9	5.3	1.6	5.6	5.2
	sp	1.18	1.01	1.21	1.10	2.12	1.13	2.28	2.03	3.49	1.66	3.77	3.68
Average W Invalidation	n	44.2	48.7	44.0	43.3	31.2	42.3	30.4	29.2	27.8	40.0	26.9	25.4
	r	34.4	47.2	33.5	34.1	19.4	39.9	17.7	17.3	15.9	37.5	14.2	12.9
	u	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	9.8	1.5	10.5	9.2	11.8	2.4	12.6	11.9	11.9	2.5	12.7	12.4
	e	10.0	1.3	10.8	6.9	6.7	1.8	7.2	6.9	5.3	1.6	5.6	5.4
	sp	1.18	1.01	1.21	1.10	2.12	1.12	2.28	2.14	3.45	1.62	3.73	3.75
Average W Transfer Cost	n	42.5	48.6	41.6	42.9	30.2	43.4	29.0	29.2	26.4	40.6	25.2	25.6
	r	38.1	48.1	36.9	38.7	23.8	41.8	22.3	23.6	19.6	39.0	18.0	19.7
	u	0.1	0.0	0.1	0.0	0.6	0.0	0.7	0.4	1.4	0.0	1.5	0.5
	s	0.1	0.0	0.1	0.0	0.4	0.0	0.5	0.3	0.7	0.0	0.7	0.3
	c	4.3	0.6	4.6	4.2	5.7	1.6	6.0	5.3	5.4	1.5	5.7	5.4
	e	10.1	1.1	10.9	7.9	9.9	3.2	10.4	8.4	9.0	2.5	9.4	8.8
	sp	1.13	1.00	1.15	1.05	1.72	1.08	1.82	1.53	2.36	1.47	2.49	2.26

Table 5.9a: The table displays the results for the normalized and discounted estimation methods for the trace from SLAC with one user per node. The results are analyzed for all shared files, sequential, partitioned, and small (< 1M bytes) files. The results for large files are similar to the results for all files. The symbols are explained in table 5.2 The row labeled "n" tabulates the total network traffic, while the rows labeled "r, u, and c" tabulate the traffic owing to remote opens, updates, and transfers of files. All network traffic is measured in percent of total I/O traffic. The row labeled "sp" tabulates the time average file system size.

Slac												
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Normalized Rate	n	37.9	48.2	37.2	33.3	28.3	42.0	27.2	25.6	25.3	39.2	24.2	23.1
	r	21.0	44.7	19.1	18.0	14.4	39.4	12.3	13.0	11.1	35.5	9.1	10.2
	u	0.3	0.0	0.3	0.1	1.2	0.0	1.4	0.8	2.1	0.1	2.3	1.2
	s	0.2	0.0	0.2	0.1	0.8	0.0	0.9	0.5	1.2	0.0	1.3	0.8
	c	16.7	3.5	17.8	15.2	12.7	2.5	13.5	11.9	12.1	3.6	12.8	11.6
	e	8.8	1.3	9.4	6.7	6.5	1.7	6.9	5.5	5.1	3.1	5.3	4.6
	sp	1.29	1.04	1.33	1.21	2.30	1.14	2.49	2.25	3.71	1.67	4.02	3.91
Normalized W Invalidation	n	39.7	48.2	39.2	35.5	30.4	42.0	29.5	27.9	28.1	39.3	27.2	25.0
	r	24.0	44.7	22.4	19.8	17.9	39.6	16.1	15.1	15.6	35.7	13.9	12.4
	u	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	15.7	3.5	16.8	15.7	12.5	2.5	13.4	12.8	12.5	3.6	13.3	12.6
	e	8.8	1.3	9.5	7.7	6.3	1.7	6.7	6.7	5.5	3.1	5.7	5.7
	sp	1.26	1.04	1.30	1.20	2.18	1.13	2.34	2.20	3.46	1.62	3.75	3.77
Normalized W Invalidation W Transfer cost	n	39.3	47.9	38.1	35.6	30.8	43.1	29.7	29.5	28.5	40.6	27.5	26.7
	r	29.5	46.0	27.7	25.2	24.7	41.4	23.3	22.7	22.7	39.0	21.3	20.2
	u	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	9.7	1.9	10.4	10.4	6.1	1.7	6.4	6.8	5.9	1.5	6.2	6.4
	e	11.1	1.8	11.7	10.1	9.2	3.1	9.6	9.3	8.6	2.5	9.1	9.3
	sp	1.20	1.02	1.23	1.14	1.71	1.09	1.81	1.57	2.28	1.47	2.40	2.23
Discounted Rate	n	43.2	48.8	43.0	42.8	29.4	42.2	28.4	27.4	25.7	39.2	24.6	23.6
	r	33.0	47.1	32.0	33.1	15.8	39.6	13.8	15.2	11.6	35.5	9.6	10.8
	u	0.2	0.0	0.3	0.1	1.2	0.0	1.4	0.7	2.1	0.1	2.3	1.2
	s	0.1	0.0	0.2	0.1	0.9	0.0	0.9	0.5	1.2	0.0	1.3	0.8
	c	10.0	1.6	10.7	9.6	12.4	2.6	13.2	11.4	12.0	3.6	12.7	11.6
	e	11.5	1.8	12.4	9.1	7.7	2.3	8.2	6.9	5.8	3.1	6.0	5.2
	sp	1.20	1.01	1.22	1.12	2.27	1.13	2.45	2.21	3.70	1.67	4.02	3.90

Table 5.8b: The table displays the results for the average estimation methods for the trace from Hughes with one user per node. The results are analyzed for all shared files, sequential, partitioned, and small (< 1M bytes) files. The results for large files are similar to the results for all files, since the behavior of these files dominates the performance of the small files. The symbols are explained in table 5.2 The row labeled "n" tabulates the total network traffic, while the rows labeled "r, u, and c" tabulate the traffic owing to remote opens, updates, and transfers of files. All network traffic is measured in percent of total I/O traffic. The row labeled "sp" tabulates the time average file system size.

Hughes												
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Average Rate	n	51.9	51.7	59.6	40.5	50.4	50.1	57.5	36.6	51.9	50.1	60.2	35.9
	r	34.3	44.2	34.7	33.7	24.1	40.7	19.6	27.5	21.6	40.0	16.1	25.5
	u	0.5	0.1	0.7	0.1	3.6	0.3	5.6	0.3	5.6	0.4	8.7	0.6
	s	0.4	0.1	0.6	0.1	2.8	0.2	4.3	0.2	4.2	0.2	6.6	0.4
	c	17.1	7.4	24.2	6.7	22.6	9.1	32.3	8.8	24.7	9.8	35.3	9.8
	e	5.6	6.5	6.1	5.2	5.5	7.2	5.6	5.1	5.9	8.2	5.8	6.1
	sp	1.08	1.04	1.11	1.03	1.96	1.37	2.29	1.30	3.04	1.73	3.75	1.91
Average W Uncertainty	n	46.8	52.0	51.3	39.5	44.1	50.0	47.7	35.2	45.6	50.3	50.2	34.1
	r	34.7	45.3	34.8	33.8	25.1	42.3	20.6	28.0	22.5	41.7	16.9	26.1
	u	0.3	0.1	0.4	0.1	3.1	0.3	4.9	0.3	4.8	0.4	7.5	0.5
	s	0.2	0.1	0.4	0.1	2.4	0.2	3.7	0.2	3.6	0.2	5.6	0.4
	c	11.8	6.6	16.1	5.6	15.9	7.4	22.2	6.9	18.3	8.2	25.7	7.6
	e	5.7	6.4	6.3	5.2	5.5	6.8	5.9	5.1	5.8	7.4	5.9	6.1
	sp	1.06	1.04	1.08	1.03	1.62	1.24	1.83	1.27	2.42	1.56	2.90	1.63
Average W Invalidation	n	52.1	51.8	59.7	40.9	48.7	50.2	54.8	37.3	48.5	50.3	54.5	37.1
	r	35.3	44.3	36.1	34.1	27.6	41.0	24.8	28.6	25.8	40.4	22.3	27.4
	u	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	16.8	7.5	23.6	6.8	21.2	9.2	29.9	8.8	22.8	9.9	32.2	9.7
	e	6.5	6.6	7.5	5.3	6.5	7.3	7.3	5.1	6.6	7.8	7.2	5.5
	sp	1.07	1.04	1.10	1.03	1.58	1.35	1.75	1.32	2.02	1.70	2.32	1.77
Average W Transfer Cost	n	42.8	53.3	44.4	41.1	36.8	51.9	35.5	36.8	35.1	51.7	32.9	36.0
	r	39.4	50.8	40.0	38.9	32.6	49.0	29.8	34.0	30.7	48.5	27.1	32.9
	u	0.1	0.0	0.1	0.0	0.2	0.0	0.3	0.1	0.4	0.1	0.7	0.2
	s	0.1	0.0	0.1	0.0	0.2	0.0	0.2	0.1	0.3	0.1	0.5	0.1
	c	3.3	2.4	4.3	2.2	4.0	2.9	5.4	2.7	4.0	3.1	5.1	2.9
	e	6.1	5.5	7.7	5.6	6.3	6.0	7.9	5.6	6.4	6.6	7.7	6.4
	sp	1.02	1.01	1.03	1.01	1.13	1.10	1.16	1.11	1.28	1.20	1.36	1.22

Table 5.9b: The table displays the results for the normalized and discounted estimation methods for the trace from Hughes with one user per node. The results are analyzed for all shared files, sequential, partitioned, and small (< 1M bytes) files. The results for large files are similar to the results for all files. The symbols are explained in table 5.2 The row labeled "n" tabulates the total network traffic, while the rows labeled "r, u, and c" tabulate the traffic owing to remote opens, updates, and transfers of files. All network traffic is measured in percent of total I/O traffic. The row labeled "sp" tabulates the time average file system size.

Hughes													
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Normalized Rate	n	47.6	49.9	53.5	37.3	50.5	49.3	58.2	35.3	53.1	48.8	62.7	34.7
	r	27.3	40.5	24.9	28.2	22.7	38.0	18.5	24.9	20.7	37.6	15.7	23.7
	u	0.5	0.1	0.7	0.1	3.8	0.3	6.0	0.4	6.3	0.4	9.9	0.7
	s	0.4	0.1	0.6	0.1	2.9	0.2	4.5	0.3	4.6	0.2	7.1	0.5
	c	19.9	9.3	27.9	9.0	24.0	10.9	33.7	10.0	26.1	10.8	37.1	10.3
	e	5.7	7.1	5.9	5.6	6.0	7.9	6.1	5.4	5.6	7.8	5.4	5.2
	sp	1.10	1.05	1.13	1.05	2.02	1.42	2.37	1.40	3.12	1.74	3.87	1.93
Normalized W Invalidation	n	49.1	50.2	55.5	38.3	50.3	49.6	57.7	36.4	51.5	49.2	59.8	36.3
	r	28.7	40.8	27.0	29.1	26.0	38.7	23.5	26.3	24.6	38.0	21.5	25.4
	u	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	20.4	9.4	28.6	9.2	24.3	11.0	34.2	10.1	27.0	11.2	38.3	10.8
	e	6.3	7.3	6.8	5.9	7.3	8.0	8.1	5.5	6.6	8.0	7.0	5.8
	sp	1.09	1.05	1.12	1.05	1.67	1.40	1.88	1.33	2.21	1.71	2.57	1.79
Normalized W Invalidation W Transfer cost	n	40.1	52.3	40.5	38.0	38.1	51.3	37.8	36.4	36.8	51.0	35.7	35.7
	r	34.1	48.6	32.4	33.7	33.1	47.8	31.2	32.8	31.8	47.5	29.2	32.1
	u	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	6.0	3.6	8.1	4.3	5.0	3.5	6.6	3.6	5.0	3.5	6.6	3.6
	e	6.8	6.1	8.3	6.4	7.2	6.1	9.0	6.9	7.2	6.4	8.9	7.1
	sp	1.03	1.02	1.04	1.03	1.11	1.11	1.14	1.11	1.24	1.20	1.31	1.21
Discounted Rate	n	51.8	50.6	59.8	40.3	51.3	49.5	59.4	35.6	53.3	48.8	63.0	34.6
	r	33.7	42.4	34.4	33.2	23.4	38.5	19.4	25.6	20.5	37.6	15.5	23.7
	u	0.5	0.1	0.7	0.1	4.1	0.3	6.4	0.4	6.4	0.4	10.0	0.7
	s	0.4	0.1	0.6	0.1	2.9	0.2	4.5	0.3	4.6	0.2	7.2	0.5
	c	17.6	8.1	24.7	7.0	23.9	10.6	33.7	9.6	26.4	10.8	37.5	10.3
	e	7.0	6.6	8.2	5.1	6.8	7.9	7.3	6.0	6.1	7.8	6.1	5.7
	sp	1.09	1.04	1.12	1.03	2.03	1.42	2.39	1.40	3.14	1.74	3.90	1.93

Table 5.8c: The table displays the results for the average estimation methods for the trace from Amdahl with one user per node. The results are analyzed for all shared files, sequential, partitioned, and small (< 1M bytes) files. The results for large files are similar to the results for all files, since the behavior of these files dominates the performance of the small files. The symbols are explained in table 5.2 The row labeled "n" tabulates the total network traffic, while the rows labeled "r, u, and c" tabulate the traffic owing to remote opens, updates, and transfers of files. All network traffic is measured in percent of total I/O traffic. The row labeled "sp" tabulates the time average file system size.

Amdahl													
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Average Rate	n	68.2	43.5	76.5	50.6	73.3	36.7	89.3	39.3	82.0	34.5	107.6	32.2
	r	54.0	38.4	54.8	42.6	42.8	30.5	39.7	26.0	26.6	27.2	13.0	13.5
	u	0.3	0.0	0.5	0.1	4.4	0.1	7.4	1.4	15.1	0.4	27.1	5.3
	s	0.2	0.0	0.4	0.1	2.4	0.1	4.3	1.0	7.2	0.2	12.5	3.3
	c	13.9	5.1	21.2	7.8	26.1	6.1	42.2	11.8	40.3	7.0	67.5	13.4
	e	5.1	5.2	1.0	2.5	9.1	5.5	7.4	5.7	9.2	5.5	4.5	3.5
	sp	1.01	1.00	1.01	1.00	1.22	1.04	1.25	1.07	3.42	1.22	3.98	1.64
Average W Uncertainty	n	60.4	43.5	63.5	46.5	56.4	36.8	58.7	35.7	61.9	34.3	70.9	30.1
	r	53.9	38.5	55.5	41.4	44.8	30.8	43.2	26.3	28.7	27.3	17.0	14.4
	u	0.1	0.0	0.2	0.1	0.7	0.1	1.0	0.9	9.7	0.4	17.3	4.5
	s	0.1	0.0	0.1	0.1	0.5	0.1	0.6	0.6	4.9	0.2	8.1	2.6
	c	6.4	5.0	7.8	5.0	11.0	5.9	14.5	8.5	23.5	6.6	36.7	11.2
	e	5.0	5.2	1.1	2.6	9.1	5.7	7.5	5.7	9.4	5.6	5.0	3.9
	sp	1.00	1.00	1.00	1.00	1.06	1.03	1.06	1.05	2.23	1.15	2.50	1.55
Average W Invalidation	n	64.7	43.7	74.6	49.7	70.3	37.3	86.2	43.5	73.5	35.2	91.7	38.2
	r	52.5	38.6	55.2	42.5	48.4	31.1	49.8	33.2	43.1	28.3	41.9	23.9
	u	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	12.2	5.1	19.4	7.1	21.9	6.2	36.3	10.3	30.4	6.9	49.9	14.3
	e	4.1	5.2	2.2	2.5	7.1	6.0	4.9	4.5	9.3	5.7	4.4	5.6
	sp	1.01	1.00	1.01	1.00	1.09	1.03	1.10	1.04	1.63	1.16	1.77	1.35
Average W Transfer Cost	n	53.3	41.4	54.3	39.6	51.1	36.8	48.6	32.5	43.3	34.8	36.5	26.0
	r	52.0	41.0	53.5	38.8	48.4	35.3	46.8	30.4	37.9	33.1	30.4	22.3
	u	0.1	0.0	0.1	0.0	0.1	0.0	0.2	0.2	1.5	0.2	2.3	1.1
	s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.6	0.1	0.6	0.4
	c	1.3	0.3	0.7	0.7	2.6	1.5	1.6	1.9	3.8	1.6	3.9	2.7
	e	3.2	0.3	1.2	1.3	9.2	3.5	7.2	5.2	9.9	3.1	8.2	5.5
	sp	1.00	1.00	1.00	1.00	1.01	1.01	1.01	1.01	1.17	1.07	1.19	1.14

Table 5.9c: The table displays the results for the normalized and discounted estimation methods for the trace from Amdahl with one user per node. The results are analyzed for all shared files, sequential, partitioned, and small (< 1M bytes) files. The results for large files are similar to the results for all files. The symbols used are explained in table 5.2 The row labeled "n" tabulates the total network traffic, while the rows labeled "r, u, and c" tabulate the traffic owing to remote opens, updates, and transfers of files. All network traffic is measured in percent of total I/O traffic. The row labeled "sp" tabulates the time average file system size.

Amdahl												
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Normalized Rate	n	63.5	38.4	72.8	42.4	72.9	36.3	87.2	36.1	86.1	35.4	110.9	32.3
	r	46.2	31.9	49.3	32.2	37.4	28.2	34.1	21.3	24.2	25.9	12.8	12.7
	u	0.3	0.0	0.6	0.2	5.0	0.1	7.5	1.5	17.8	0.5	27.7	5.5
	s	0.3	0.0	0.4	0.1	3.0	0.1	4.4	1.1	9.7	0.3	13.2	3.4
	c	16.9	6.5	22.9	10.0	30.4	7.9	45.5	13.2	44.1	8.9	70.4	14.0
	e	7.7	5.3	3.6	5.5	10.1	5.4	8.1	5.1	8.5	5.8	4.2	3.3
	sp	1.01	1.01	1.01	1.01	1.24	1.05	1.26	1.08	3.54	1.24	4.06	1.65
Normalized W Invalidation	n	65.0	38.9	75.7	45.0	74.8	36.8	94.1	42.3	85.8	36.1	115.2	40.4
	r	47.9	32.3	52.6	33.9	45.8	28.9	48.2	28.2	41.3	27.0	41.2	22.1
	u	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	17.1	6.6	23.1	11.1	29.0	7.9	45.9	14.1	44.5	9.1	74.0	18.3
	e	8.8	5.6	5.3	4.5	9.0	5.6	7.0	5.3	9.1	6.3	4.8	5.2
	sp	1.01	1.01	1.01	1.01	1.10	1.05	1.11	1.05	1.81	1.16	1.98	1.37
Normalized W Invalidation W Transfer cost	n	51.3	36.5	53.1	34.0	54.2	35.5	51.0	31.0	52.8	35.3	49.7	28.8
	r	48.1	34.6	51.6	30.9	49.9	33.5	48.3	28.1	46.9	33.3	44.9	25.3
	u	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	c	3.2	1.9	1.5	3.1	4.3	2.0	2.7	3.0	5.9	2.0	4.8	3.5
	e	6.6	2.4	4.5	3.3	9.9	2.9	6.0	4.9	10.3	4.0	5.0	5.8
	sp	1.00	1.00	1.00	1.00	1.02	1.01	1.01	1.01	1.08	1.06	1.08	1.09
Discounted Rate	n	65.3	41.2	71.7	45.2	72.7	37.1	87.3	36.7	88.5	35.0	113.8	32.3
	r	49.1	35.6	46.0	36.0	38.0	29.7	31.7	22.4	23.7	25.7	11.1	12.1
	u	0.3	0.0	0.6	0.2	4.6	0.1	8.0	1.5	19.2	0.5	30.0	5.9
	s	0.3	0.0	0.4	0.1	2.8	0.1	4.6	1.2	10.2	0.3	13.8	3.5
	c	15.9	5.5	25.1	9.0	30.1	7.3	47.6	12.8	45.5	8.8	72.7	14.3
	e	8.3	7.0	7.8	7.6	10.3	6.0	7.3	6.5	9.1	5.5	4.4	3.5
	sp	1.01	1.00	1.01	1.01	1.24	1.04	1.28	1.08	3.62	1.22	4.16	1.66

5.8.2.3. The Advantage of Replication Compared to Migration

A previous section concluded that replication potentially has a substantial advantage over migration, since the performance of the look-ahead algorithm with replication is better than the one for migration. This was true only for two traces. For the last trace, Amdahl, and for sequential files, replication does not offer any additional benefits compared to migration.

The same conclusions are true for the proposed realizable algorithms. For SLAC, Hughes and Amdahl. the differences between the best algorithm with replication and the best algorithm with migration (BAT0) are 41%, 20% and 9%. For sequential files, the best algorithm with replication has the same or higher network traffic than the best migration algorithm (BAT0). This implies that the proposed algorithms take advantage of the potential benefit of replication when it is beneficial to replicate. However, for sequential files they offer no advantage over migration.

5.8.3. Results with One Department per Node

The results are similar when users from the same department are placed at the same node. The major difference is the increased disadvantage of invalidating versus updating. In addition, the algorithms with elaborate estimation schemes only offer a slight advantage over demand copy algorithms for the trace from SLAC, since under department mapping most nodes can justify creating their own local copies. For the other trace, demand copying has a much worse performance, since the typical file should be replicated at only a few of the departments using it. The algorithms with the more elaborate estimation scheme then have a superior performance.

Demand Copying. Only for the trace from SLAC is demand copying suitable (see Figure 5.5), and it is suitable for all file types. Even at the highest storage cost, demand copying is viable. However for both traces, a substantial part of the update traffic is to copies that will be erased before they are accessed. Only a few of the nodes that have referenced a file before it is updated will do so after the update. Thus the extended locality is fairly limited.

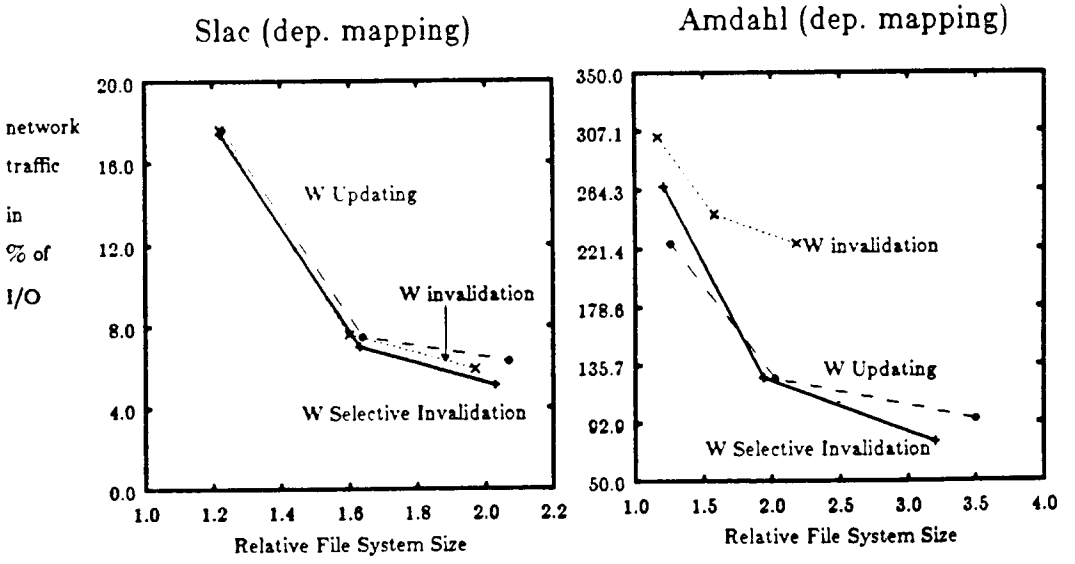


Figure 5.5: The performance for the three version of demand copy algorithms for the traces from SLAC and Amdahl. The results are with one department per node. Note that the left and the right figure use different scales.

Estimation Methods. As we already mentioned, demand copying is suitable for one of the traces (SLAC), and the difference between the estimation methods will be small. Only for the highest storage cost is the normalized reference rate preferable. On the other hand, the results confirm that the estimation methods will not result in a large performance degradation even when most nodes have reference rates that are high enough to justify demand copying.

For the other trace, the simulation displays the same results as with one user per node; the scheme accounting for the uncertainty of the estimates reduces the network traffic, and accounting for the estimated lifetime of a copy reduces the network traffic further.

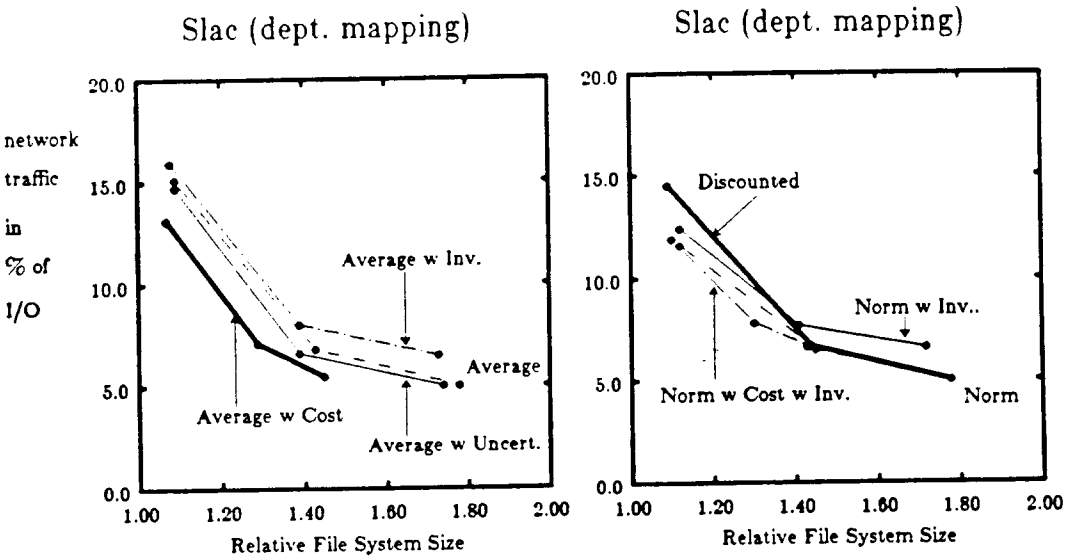


Figure 5.6a: The performance for the various estimation methods for the traces from SLAC. The results are with one department per node.

Invalidating versus Updating. This is the only aspect where the results are changed. For the trace from Amdahl, invalidation increases the network traffic $\approx 30\%$, compared with updating. Even though most of the update traffic is to copies that will never be accessed before they are erased, updating is still advantageous. With invalidation, both the remote traffic and the traffic owing to creating copies increase.

To conclude, with the mapping of departments to nodes, there is a difference in the performance between updating versus invalidating ($\approx$30%). The added complexity of updating may therefore be justified.

5.8.4. Evaluation of the Proposed Algorithms

We only analyzed a small fraction of all possible algorithms, and may have overlooked algorithms that could have better performances. It is therefore necessary to investigate how "good" our algorithms are; i.e., how close are they to the best realizable algorithms. Unfortunately, the best realizable algorithms are unknown, and the only measures

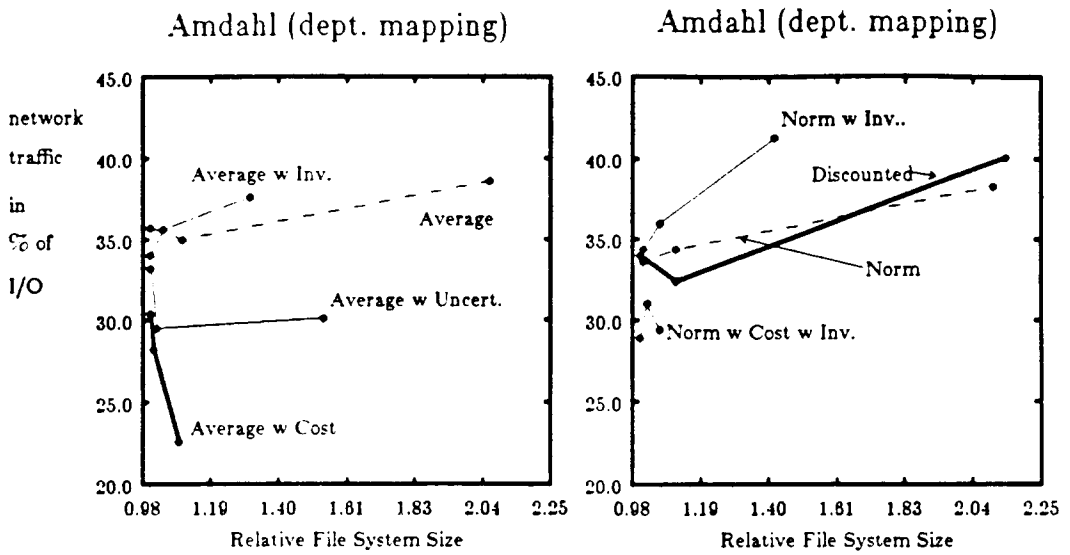


Figure 5.6b: The performance for the various estimation methods for the traces from Amdahl. The results are with one department per node.

we can use are the look-ahead algorithms. However, since these algorithms make decisions based on full knowledge of future references, there will always be a difference in their performance and that of any realizable algorithms. Whether an algorithm has a "good" performance or not is a judgement based on the magnitude of the difference in performance.

To judge whether the difference is reasonable or not, we built a simulation model with a reference pattern similar to the ones we observed in the traces. The model is described in the next section. The goal was to develop a model that had a simple enough reference process so the best stochastic policy could be calculated. On the other hand, the model should also reproduce some of the features observed in the traces. The difference between the best realizable policy and the look-ahead policy for this model is then a measure of the performance penalty owing to decisions made under uncertainty. Such an approach is similar to the one used by Smith in an analysis of the look-ahead algorithms for paging [Smith76]. For completeness the proposed algorithms were also run for the simulation model.

In this section, we analyze the difference between the results for the proposed algorithms and results for the look-ahead policies. We find that the difference is similar to what must be expected between the look-ahead and the best possible stochastic algorithm, and the proposed policies seem to have a reasonable performance.

For SLAC and Hughes, most of the difference between the proposed algorithms and the look-ahead algorithms is due to remote I/O traffic from nodes that eventually will have their own copy of a file. The proposed algorithms do not make any placement decisions until at least the second reference from the node, and the largest potential for improving the performance is to reduce this remote traffic by making the placement decision immediately when a node starts referencing a file. As we have seen, demand copying

is not a suitable scheme, since most files should only be replicated to a few of the users referencing them. We conjecture that instead user identity and file content can be used to determine when and where files should be replicated.

5.8.4.1. Sources of Difference between a Realizable Algorithm and a Look-ahead Algorithm

Three different factors contribute to the difference in performance between a look-ahead algorithm and a realizable algorithm. A realizable algorithm is based on a characterization of the future references that may be wrong or incomplete, and it must also estimate the different parameter values. In addition, there is always a difference between decisions based on the correct stochastic model and the ones made with full knowledge of the future.

The algorithms we investigate are based on a simple characterization of the reference process. They use the average reference read and update rate, the number of consecutive update references, and the time during which the node has referenced the file to determine when and where files should be replicated. The characterization is incomplete, and it is likely that a better performance could be obtained with a more complete characterization. The purpose of this section is to show that the error due to this simple characterization is small. Since the correct characterization is not known, our claim can only be supported by analyzing the contribution the two other sources make to the difference in performance. As previously mentioned, these sources are the parameter estimation and the uncertainty of the future references.

All the realizable algorithms, except the demand copy algorithms, must estimate the values of the various parameters, and at least one open will have to be done remotely before a copy can be created. A look-ahead algorithm has no need to estimate any parameters and will create a copy immediately when it is beneficial. The rows with label "e" (wasted network traffic due to estimation) in Tables 5.8, 9, and 10 are upper bounds on the network traffic wasted as remote I/O while the algorithms estimate the various parameters.

Even if the algorithms are based on the correct characterization, and all parameters are known, there will be a difference in performance between a realizable algorithm and a look-ahead algorithm. As an example, let us assume that for each node the number of opens before a copy is erased has a geometric distribution with mean 10. Further, assume that each open references half of the file, all opens are in read mode, and the cost of making a copy is equal to the file size. The best stochastic strategy is to always make a copy when a node starts referencing a file. However, in 10% of the cases a node will open the file only once before the copy is erased, and a look-ahead algorithm will instead do the open remotely. A file transfer is avoided and the difference in performance will then be 5% (10% times 0.5).

5.8.4.2. Measured Difference between the Proposed Algorithms and the Look-ahead Policies

Table 5.10 summarizes the difference in performance between the proposed algorithms and the look-ahead algorithms. The row labeled "e" tabulates the remote access traffic from nodes that eventually will have their own copy, and we believe it is a tight bound on the remote traffic avoided by knowing the parameter estimates. Most of the remote traffic from nodes that will have their own local copy is done while the algorithms estimate the various parameters. All opens from new nodes are handled remotely until the nodes that should have a copy can be identified, since most files should be replicated to only a few nodes. A look-ahead algorithm need not estimate any parameters at all, and it will avoid most of this remote traffic. The row is only an upper bound on the traffic that can be avoided by a look-ahead algorithm, because it is feasible that some opens should be handled remotely even from nodes that will have a local copy; by delaying creating the local copy, some or all of the update traffic to the copy can be avoided. The row labeled "s" (network traffic wasted in updating stale copies) contains all update traffic to copies that will be erased before they are accessed again. All look-ahead algorithms avoid this traffic.

Table 5.10a: The table displays the sources of difference in performance between the proposed algorithms and the look-ahead policies (WFMC) for the trace from SLAC. The results are obtained with one user per node. The description of the content of the various rows is found in Table 5.2 and in Section 5.8. The rows labeled "e" and "s" tabulate network traffic due to estimating parameters and updating stale copies. The total network traffic minus these two sources of wasted traffic is tabulated in the row labeled "p". The difference between this traffic and the optimal look-ahead traffic (WFMC) is then a measure of the performance penalty due to wrong decisions. The row labeled "%" tabulates this difference measured in percent of the performance for the optimal look-ahead algorithm (WFMC).

Slac													
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Average W Transfer Cost	n	42.5	48.6	41.6	42.9	30.2	43.4	29.0	29.2	26.4	40.6	25.2	25.6
	s	0.1	0.0	0.1	0.0	0.4	0.0	0.5	0.3	0.7	0.0	0.7	0.3
	e	10.1	1.1	10.9	7.9	9.9	3.2	10.4	8.4	9.0	2.5	9.4	8.8
	p	32.3	47.5	30.6	35.0	19.9	40.2	18.1	20.5	16.7	38.1	15.1	16.5
	%	39.2	58.9	34.2	42.9	14.4	37.2	9.7	10.8	16.8	37.5	14.4	8.6
Normalized W Invalidation W Transfer cost	n	39.3	47.9	38.1	35.6	30.8	43.1	29.7	29.5	28.5	40.6	27.5	26.7
	s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	e	11.1	1.8	11.7	10.1	9.2	3.1	9.6	9.3	8.6	2.5	9.1	9.3
	p	28.2	46.1	26.4	25.5	21.6	40.0	20.1	20.2	19.9	38.1	18.4	17.4
	%	14.2	54.2	8.2	-5.6	10.8	36.5	6.9	-3.8	19.2	37.5	16.5	-1.7

The total network traffic minus the rows labeled "s" and "e", which tabulate wasted network traffic that any look-ahead algorithm would avoid, is tabulated in the row labeled "p". This row (labeled "p") is a bound on the network traffic of a look-ahead algorithm

Table 5.10b: The table displays the sources of difference in performance between the proposed algorithms and the look-ahead policies (WFMC) for the trace from Hughes. The results are obtained with one user per node.

Hughes													
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Average W Transfer Cost	n	42.8	53.3	44.4	41.1	36.8	51.9	35.5	36.8	35.1	51.7	32.9	36.0
	s	0.1	0.0	0.1	0.0	0.2	0.0	0.2	0.1	0.3	0.1	0.5	0.1
	e	6.1	5.5	7.7	5.6	6.3	6.0	7.9	5.6	6.4	6.6	7.7	6.4
	p	36.6	47.8	36.6	35.5	30.3	45.9	27.4	31.1	28.4	45.0	24.7	29.5
	%	27.1	13.0	35.1	24.6	14.8	10.6	16.6	19.2	13.6	9.2	15.4	15.7
Normalized W Invalidation W Transfer cost	n	40.1	52.3	40.5	38.0	38.1	51.3	37.8	36.4	36.8	51.0	35.7	35.7
	s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	e	6.8	6.1	8.3	6.4	7.2	6.1	9.0	6.9	7.2	6.4	8.9	7.1
	p	33.3	46.2	32.2	31.6	30.9	45.2	28.8	29.5	29.6	44.6	26.8	28.6
	%	14.4	8.5	17.9	8.2	15.3	7.9	20.0	9.3	15.6	7.2	21.3	8.3

Table 5.10c: The table displays the sources of difference in performance between the proposed algorithms and the look-ahead policies (WFMC) for the trace from Amdahl. The results are obtained with one user per node.

Amdahl													
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Average W Transfer Cost	n	53.3	41.4	54.3	39.6	51.1	36.8	48.6	32.5	43.3	34.8	36.5	26.0
	s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.6	0.1	0.6	0.4
	e	3.2	0.3	1.2	1.3	9.2	3.5	7.2	5.2	9.9	3.1	8.2	5.5
	p	50.1	41.1	53.1	38.3	41.9	33.3	41.4	27.2	32.8	31.6	27.7	20.1
	%	71.0	44.7	91.7	79.0	52.4	22.0	59.8	37.4	29.1	20.2	23.1	16.2
Normalized W Invalidation W Transfer cost	n	51.3	36.5	53.1	34.0	54.2	35.5	51.0	31.0	52.8	35.3	49.7	28.8
	s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	e	6.6	2.4	4.5	3.3	9.9	2.9	6.0	4.9	10.3	4.0	5.0	5.8
	p	44.7	34.1	48.6	30.7	44.3	32.6	45.0	26.1	42.5	31.3	44.7	23.0
	%	43.3	19.6	71.1	40.2	46.7	18.5	65.4	26.7	46.0	17.7	75.3	23.0

Table 5.10d: The table displays the sources of difference in performance between the proposed algorithms and the look-ahead policies (WFMC) for the trace from SLAC. The results are obtained with one department per node.

Slac (department mapping)													
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Average W Transfer Cost	n	13.1	4.2	13.9	16.1	7.1	3.4	7.5	7.8	5.5	3.0	5.7	6.2
	s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.0	0.1	0.1
	e	4.9	0.3	5.4	4.4	2.7	0.6	3.0	2.7	2.2	0.5	2.4	2.5
	p	8.2	3.9	8.5	11.7	4.4	2.8	4.5	5.0	3.2	2.5	3.2	3.6
	%	18.8	56.0	16.4	33.0	15.8	33.3	15.4	4.2	14.3	38.9	10.3	2.9
Normalized W Invalidation W Transfer cost	n	11.9	3.7	12.7	12.4	7.8	3.2	8.2	7.8	6.7	2.9	7.1	6.6
	s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	e	4.7	0.5	5.1	4.0	2.6	0.6	2.8	2.6	2.3	0.5	2.5	2.6
	p	7.2	3.2	7.6	8.4	5.2	2.6	5.4	5.2	4.4	2.4	4.6	4.0
	%	-1.4	28.0	-1.3	-6.7	18.2	23.8	17.4	0.0	29.4	33.3	27.8	5.3

Table 5.10e: The table displays the sources of difference in performance between the proposed algorithms and the look-ahead policies (WFMC) for the trace from Amdahl. The results are obtained with one department per node.

Amdahl (department mapping)													
		d= 8h				d = 48h				d = length of trace			
		all files	seq. files	part. files	small files	all files	seq. files	part. files	small files	all files	seq. files	part. files	small files
Average W Transfer Cost	n	30.4	15.5	27.0	16.8	28.2	12.6	22.9	13.9	22.6	11.3	16.9	11.1
	s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.4	0.1
	e	2.6	0.7	1.9	0.7	8.4	2.4	7.5	1.9	6.7	2.6	4.7	2.4
	p	27.8	14.8	25.1	16.1	19.8	10.2	15.4	12.0	15.4	8.7	11.8	8.6
	%	100.0	66.3	156.1	76.9	54.7	25.9	71.1	39.5	29.4	19.2	53.2	16.2
Normalized W Invalidation W Transfer cost	n	28.9	12.7	27.8	14.4	31.0	11.8	25.6	13.1	29.4	11.7	25.0	12.0
	s	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	e	5.3	2.2	3.8	1.1	8.4	2.2	4.6	1.7	9.1	3.2	4.5	2.7
	p	23.6	10.5	24.0	13.3	22.6	9.6	21.0	11.4	20.3	8.5	20.5	9.3
	%	56.3	15.4	137.6	43.0	54.8	15.7	114.3	29.5	44.0	11.8	130.3	19.2

that creates copies at the same nodes as the proposed algorithms. The difference between this look-ahead algorithm and the optimal look-ahead algorithms is then a measure of the difference in performance owing to different decisions. For convenience, this difference is calculated in percentage of the optimal look-ahead policy in the row labeled "%". The network traffic in the row labeled "p" can be lower than the optimal look-ahead traffic (WFMC) for two reasons: 1) The optimal look-ahead policy (WFMC) assumes a fixed master copy, and policies without this restriction may have a better performance. 2) The row labeled "e" is only an upper bound on the remote traffic avoided by not having to estimate the various parameters. Subtracting the row from the total network traffic may therefore overestimate the advantage of not having to do the various parameter estimates.

The difference is in the 0 to 15% range for two of the traces, while it is in the 30 to 50% range for the last trace (Amdahl). The difference is higher for sequential files and when users from the same department are placed at the same node. However, in the latter case, the algorithms we compare are not the ones that consistently have the best performance. As explained later, we believe these differences are reasonable. For the traces we consider, such a difference is to be expected between decisions made under uncertainty and those made with full knowledge of the future.

From Table 5.10 it is evident that for two of the traces, remote network traffic from nodes that eventually will have their own copy dominates any increase in network traffic due to wrong decisions. Most of this network traffic can be avoided if the nodes that should have a copy can be identified immediately when they start using a file. As we have already seen, demand copying is not the answer; most nodes should not have a copy. We have not been able to come up with the appropriate algorithm, but we conjecture it must be based on the file content and the identity of the user accessing the file. However, such a study is outside both our scope and the traces used. For the last trace (Amdahl), the wasted remote I/O is small compared to the error due to wrong decisions.

In the next section, we justify why the observed differences, in particular for SLAC and Hughes, are reasonable. To judge whether the observed differences are reasonable or not, we built a simulation model with a reference pattern similar to the ones we observed. In the model, users are interested in referencing a file only for a limited time, and the mean number of concurrent users (the extended locality) is small (1.51). Likewise, the number of users sharing the file is low (4.0), and the average number of opens per user is also small (4.0). All of these features were found in the three traces. Unfortunately, the model can only have a simplistic representation of the actual reference process, since it must be feasible to find the optimal realizable algorithm. The difference between this policy and the look-ahead policy is then a measure of the performance penalty due to decisions made under uncertainty. For completeness, the proposed algorithms were also run for the simulation model.

5.8.4.3. Description of Simulation Model

The model must incorporate the time between opens, the probability an open will update the file, and the limited time a node is interested in using a file. Further, most files should have a limited locality, and the number of users sharing the file should be small. The mean number of opens per user per file should also be small. All of these features were found in the traces.

Unfortunately, the model can only have a simplistic representation of the actual reference process, since it must be feasible to find the optimal realizable algorithm. We believe our example with the geometric distribution is typical for the differences in the performance between the algorithms we propose and the look-ahead algorithms. The

model was therefore set up so the number of opens per user had a geometric distribution with mean $\frac{1}{1-p}$, and the time between opens is modeled by an exponential distribution. The model can be described in terms of a M/G/∞ queuing model [Wolff88]. The server represents the file, while the customers represent the users. An arrival to the system signals that a user has started using the file, while a departure signals that a user has stopped using the file (has lost interest). When the last customer leaves the system, the file is erased. The extended locality of the file is then the number of customers in the system.

Users arrive to the system, i.e., start accessing the file at a rate $\lambda = 0.125$. This is the rate at which new users become interested in using a file. The time between opens from the same user has an exponential distribution with mean $\mu = 0.5$. The values for λ and μ were chosen so the extended locality would be limited, and their actual values are less important. Since the number of opens per user has a geometric distribution with mean $\frac{1}{1-p}$, the probability that the user will continue to reference the file, p, is constant and equal to 0.75, i.e., a mean of 4 opens per user. The time a user will be interested in referencing a file then has a distribution which is a mixture (with probability p) of an exponential distribution with mean $(\frac{1}{\mu(1-p)})$ and (with probability $(1-p)$) a constant equal to 0.0. The probability an open is in update mode, u, is constant and set to 0.1. All opens transfer a constant fraction of the file equal to 0.5.

This model behaves as a M/G/∞ queue with arrival rate λ and mean service time equal to $\frac{p}{\mu(1-p)}$. With the given parameter values 37.5% of the files are shared, with a mean number of 4.0 users per shared file. The mean number of users in the extended locality for shared files (the number of users currently interested in using the file) is 1.51, and the average number of opens per user is small (4.0). This simulation model displays many of the characteristics we observed in the traces.

The model is simple enough so the best realizable algorithm can be found. Equation 5.3a and 5.3b is used to calculate when copies should be created. The time horizon, T, should be the expected remaining time a node will access a file ($T = \frac{1}{(1-p)\mu}$). The expected number of updates to a copy from one of the nodes currently referencing a file is $\frac{u}{2(1-p)}$. In addition, $\frac{\lambda}{\mu(1-p)}$ new nodes will start using the file during the remaining lifetime of the copy. A copy should only be created when the expected advantage of a local copy, $\frac{1}{1-p}$, is larger than the expected update cost, transfer cost and storage cost.

$$\frac{1}{1-p} > u\left[(k-1)\frac{1}{2(1-p)} + \frac{\lambda}{\mu(1-p)}\left[1 + \frac{p}{2(1-p)}\right]\right] + \left[c_t + c_s\frac{1}{\mu(1-p)}\right]F$$

F is the relative file size compared to the fraction accessed per open, and k is the number of nodes using the file (the extended locality). Similarly, if there is an update to the file, all copies except one should be erased when k is such that

$$\frac{1}{1-p} < u\left[(k-2)\frac{1}{2(1-p)} + \frac{\lambda}{\mu(1-p)}\left[1 + \frac{p}{2(1-p)}\right]\right] + \left[1 + \frac{u \cdot p}{2(1-p)}\right] + c_s\frac{1}{\mu(1-p)}F$$

The conditions for creating / erasing copies can be summarized as : create copies as long as only $\leq k'$ nodes are using the file. If there are more than k' nodes using the file, no

new copies should be created. The additional copies should, however, not be erased until the number of nodes using the file is larger than k'' ($k' \leq k''$). When the number of users is larger than k'', the expected update traffic will be larger than the expected advantage of a local copy, and all additional copies should be erased. Obviously, the additional copies should not be erased until the file is updated. For the parameter values in our model $k' = 2$ and $k'' = 8$ with $c_s = \frac{1}{11}$ and $c_t = 1$. All nodes have the same properties, and the master copy can be placed at any of the nodes with a copy.

The purpose of the simulation is to illustrate the difference in performance between the realizable algorithm and the look-ahead algorithm. Table 5.11 summarizes the results. For completeness, the results for all files accessed by less than 4 nodes are also included. The performance metric is the network traffic in percent of I/O traffic. For convenience, the numbers in the parenthesis are the differences to the optimal look-ahead policy (WFMC) given in percent of the optimal solution. (For row i, the number in parenthesis is [row i - row 1] / row 1).

Table 5.11: The table displays the results from the simulation. The network traffic is measured in percent of I/O traffic. The numbers in parenthesis measure the difference in performance between the various algorithms and the look-ahead algorithm (WFMC) in percent of the latters performance.

	all files		Files accessed by < 4 nodes	
Optimal Look-ahead Policy	28.2%		23.8%	
Best Stochastic Algorithm	41.7%	(48%)	28.8%	(21%)
Proposed Algorithms				
Average Rate	48.8%	(73%)	33.1%	(39%)
Normalized Rate	47.7%	(69%)	32.3%	(36%)

The differences between the optimal look-ahead (WFMC) and the best realizable algorithm are from 21% to 48%. A large part of this difference is caused by creating copies at nodes which will only access a file once. For all files, this error constitutes about 21% (out of a 48% difference). The simulation model at least exemplifies that the differences between a realizable algorithm and a look-ahead algorithm can be of the same order as those observed in the trace driven simulation. Since the simulation model had similar reference pattern to the ones observed in the traces, the quality of the proposed algorithms seems reasonable, at least for the traces from SLAC and Hughes.

In the simulation model, the proposed algorithms perform substantially worse than the best realizable algorithm. This is due to the properties of the environment we simulated. In the simulation, most nodes will have their own copy, and a copy is created for the first open. However, the algorithms we propose cannot create any copy before at least the second reference from a node; the first open must be handled remotely. If we assume that our proposed algorithms will create a copy just before the second reference from all nodes that should have a copy, this forced remote access traffic corresponds to 9.8% of the I/O traffic (or 34% of the difference to the optimal look-ahead (WFMC)). It appears that the remote traffic while estimating parameters causes most of the observed difference between our proposed algorithms and the best stochastic algorithm in the simulation

model. This is similar to our observations for the trace driven simulation.

5.8.4.4. The Quality of the Proposed Algorithms

This section summarizes the results from the three previous sections. For two of the traces, the differences in performance between the decisions made by our proposed algorithms and the look-ahead algorithm are in the 0 to 15% range (for SLAC and Hughes), while they are between 30% and 50% for Amdahl. Through a simulation model, we showed that such differences are to be expected between decisions made under uncertainty and those made with full knowledge about the future.

For the trace from SLAC and Hughes, the major source for the difference in performance was the remote I/O traffic from nodes that eventually will have their own copy. To make the placement decision, our proposed algorithms need to observe the reference pattern before they replicate the file. This remote traffic constitutes most of the difference in performance between the algorithms and the look-ahead algorithms. Demand copy algorithms are not a solution, since a typical file should only be replicated at a few of the sites using it. The most likely way to improve the performance is to immediately identify the nodes that should have a copy of a file when they start referencing it. Such an identification must be based on file content and identity of the user accessing the file. Hopefully these two parameters will be sufficient to describe the future access pattern to the file.

5.9. Conclusion

In this chapter we have quantified the advantage replication offers compared to migration alone, and the corresponding cost in terms of increased file system size. Sequential files are best manged by migration alone, since their extended locality is limited. Little is gained by replicating them.

Whether copies should be invalidated or updated when a file is updated depends on the installation and the rule for placing users to nodes. The largest difference between invalidation and updating was a 22% difference in network traffic for the look-ahead policy and $\approx$ 30% difference for the algorithms we propose. The smallest difference was in the 3% range. Sequential files are often overwritten when they are updated, and invalidation is the preferred scheme for these files. However, these files are best managed by migration alone.

The best placement decisions are based on the average reference rate, the expected number of consecutive update references, and the time a node has referenced a file. It is also worth accounting for the uncertainty of the estimates in the placement decisions. Several different estimation methods were evaluated, but they all had similar performances. However, demand copying is not a suitable solution.

In this chapter we justified why there is little to be gained from improving the decisions made by the proposed algorithms; they are reasonably close to the optimal decisions, at least for two of the traces (SLAC and Hughes). Most likely, any further improvement will come from reducing the number of references it takes the algorithms to make a placement decision. Currently, estimating the various parameters requires observing several opens. While the estimates are made, the opens are done remotely, which results in unnecessary network traffic from the nodes that eventually will have their own copy. This unnecessary network traffic constitutes most of the difference in performance between the proposed algorithms and the look-ahead algorithms. However, we are not able to avoid this traffic. As we already mentioned, demand copying is not a suitable scheme; only a few nodes should have their own copy of a file.

Table 5.12a: The table displays the definitions used to explain the look-ahead algorithms.

b_n^u	Amount of update I/O traffic by the n'th access to the file
b_n^r	Amount of read I/O traffic by the n'th access to the file
b_n	Amount of I/O traffic by the n'th access to the file $b_n = b_n^r + b_n^r$
T_n	time between the n-1'st and n'th access
F_n	File size after the n-1'st access
i_n	Identity of the node issuing the n'th access
a_n	mode of the n'th access. If the access is an update, $a_n = 1$, otherwise $a_n = 0$
N	Total number of accesses to the file
m	Total number of nodes accessing the file
I	Subset of the m nodes that have a copy of the file
$\|I\|$	Number of elements in set I. $\|I\| > 1$, since I is never an empty set.
$V_n(I)$	The minimal cost up to the n'th access when I is the state at the end of the n'th access
Cost	
c_r	Cost of a remote access measured per byte transferred
c_u	Cost of updating a copy measured per byte transferred
c_t	Cost of creating a copy measured per byte transferred
c_s	Cost of storing a file measured per byte stored per time unit

Table 5.12b: The table explains the definitions used to describe the various file replication algorithms.

$E(F)$	Expected file size
$E(B_i^r)$	Expected number of bytes transferred in read mode by node i per open
$E(B_i^u)$	Expected number of bytes transferred in update mode by node i per open in update mode
$E[R_i(T)]$	Expected number of read references during an interval T from node i
$E[U_i(T)]$	Expected number of update references during an interval T from node i
$E(T_i)$	Expected time between read references from node i
$E(T_i^u)$	Expected time between update references from node i
$\lambda_i^r = \frac{1}{E(T_i)}$	reference rate measured in accesses per time unit
$\lambda_i^u = \frac{1}{E(T_i^u)}$	update reference rate measured in accesses per time unit
$r_i = \frac{E(B_i)}{E(T_i)}$	reference rate measured in bytes per time unit
$r_i^u = \frac{E(B_i^u)}{E(T_i^u)}$	update reference rate measured in bytes per time unit
$!=i$	short hand notation for all other nodes except node i
$t_i^{r,j}$	time of the j'th read reference from node i
$t_i^{u,j}$	time of the j'th update reference from node i
$b_i^{r,j}$	number of bytes transferred by the j'th read reference from node i.
$b_i^{u,j}$	number of bytes transferred by the j'th reference in update mode from node i.
$n_i(S)$	number of references from node i to the file up to epoch S
α	discount rate
d	maximum time a copy can remain unreferenced before it is erased

Chapter 6
Research Contributions and Future Work

6.1. Research Contributions

This thesis makes two contributions to the research on file placement; it analyzes several models of how files are referenced, and it proposes new placement algorithms that improve the system performance.

Chapter 3 discusses different models of how files are referenced. Several models are compared, and we conclude that a batch Poisson model is the most appropriate one. It is simple, and it has most of the properties observed in the traces. In addition, it is also a plausible model of how files are referenced.

The batch Poisson model is used to formulate several placement algorithms for distributed file systems with single or multiple file copies. These algorithms adapt themselves to the reference pattern for each user/file pair so the algorithms are robust. More important, they have a better performance than any of the algorithms based on other models.

The performance of the the various algorithms is measured with trace driven simulation. To estimate the quality of the algorithms, we calculate the optimal look-ahead placement, which is the optimal placement given that the future reference pattern to the file is known. The look-ahead placement provides a bound on the performance of any placement algorithm, and it can be used to evaluate the quality of the proposed algorithms. The optimal look-ahead placement is also used to measure the advantage of replication and migration over static placement.

The look-ahead placement showed that only a few files should be moved or copied. The algorithms that we propose are able to identify these files and locations, but to do so the algorithms must estimate several parameters for each user/file pair.

The important estimate for the migration algorithms is the number of consecutive opens from a location to the same file. This estimate must be based on the pattern of previous references from the location to the node, but the estimation method is not important.

For the algorithms that can replicate files, the important estimates are the average reference rate and the time the node has referenced the file. Whether copies should be updated or invalidated, when the file is updated, depends on several factors. For one of the traces it is beneficial to update, while for the two others the difference between updating and invalidating is small.

To summarize, our research has proposed and analyzed a new model of how files are referenced. This model was used to formulate placement algorithms that had better performance than algorithms based on other models. We were also able to document the advantages replication and migration have over static placement.

6.2. Future Work

We have claimed throughout this thesis (Chapter 4 and 5) that there is little room for improvement of our algorithms. This is only valid for the migration or replication decisions made by the algorithms. Since only a few files should be migrated or replicated to some of the nodes, the reference pattern for each node/file pair must be estimated.

This estimation results in a performance degradation that could be avoided if it was possible to recognize the nodes a file should be replicated or migrated to before they started referencing the file.

We are not able to this, since we lack knowledge about the users, the projects they are working on, and who they are cooperating with. A future area of research is how more information about the users and their work can substitute for or supplement the estimation schemes we use. If this is possible, the performance can be substantially improved (see Chapters 4 and 5).

An underlying assumption for this thesis is that parts of the file cannot be migrated or copied. Instead, the only choice is to migrate or replicate the whole file. However, in the traces we analyzed many users accessed only a part of the file at each open. It is feasible that these users are only interested in a specific part of the file, and the performance can then be improved by migrating or replicating locally only those parts the user is interested in. An interesting area of future research is therefore migration or replication algorithms that operate on only parts of a file. With these algorithms the individual disk blocks can be replicated or migrated, and the cost for obtaining local I/O may be smaller than when the whole file must be migrated or replicated. Owing to the lack of granularity in the traces, we were not able to investigate these partial migration and replication algorithms.

The file system, however, becomes more complex, since the granularity of the file administration is no longer at the file level but at the disk block level. For example, it is conceivable that none of the nodes will have a complete copy of a file. This makes file retrieval more complex.

Partial file migration and replication will also closely interact with the buffer management. Under partial replication a disk block can be placed in any combination of the following alternatives: 1) in a remote buffer, 2) at a remote disk, 3) in a local buffer, and 4) at a local disk. Partial replication will therefore interact with the buffer management. How to achieve the best interaction between these two schemes should be an interesting area of future research.

References

Almes85.
G.T. Almes, A.P. Black, E.D. Lazowska, and J.D. Noe, "The EDEN System: A Technical review," *IEEE Transaction on Software Engineering*, vol. SE-11, pp. 43-59, January 1985.

Bassio84.
M. A. Bassiouni and J. R. Spirn, "Workload Characterization at the Program Level - a Model of I/O," *Computer Performance*, vol. 5, no. 1, pp. 23-30, March 1984.

Bernst81.
P.A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," *ACM Computing Surveys*, vol. 13, no. 2, pp. 185-221, June 1981.

Birrel82.
A. Birrell, R. Levin, R.M. Needham, and M.D. Schroeder, "Grapevine: An Exercise in Distributed Computing," *Communication of the ACM*, vol. 25, no. 4, pp. 260-274, April 1982.

Birrel84.
A. Birrell, R. Levin, R.M. Needham, and M.D. Schroeder, "Experience with Grapevine: The Growth of a Distributed System," *ACM Transaction on Computer Systems*, vol. 2, no. 1, pp. 3-23, February 1984.

Blevin76.
P. R. Blevins, and C.V Ramamoorthy, "Aspects of a Dynamically Adaptive Operating System," *IEEE Transactions on Computers*, vol. C-25, no. 7, pp. 713-725, July 1976.

Cabrer87.
L.F. Cabrera and J. Wyllie, "QuickSilver Distributed File Services: An Architecture for Horizontal Growth," IBM Almaden RJ5578, 4/1/1987.

Casey72.
R.G. Casey, "Allocation of Copies in an Information Network," *Proceedings of AFIPS 1972 Spring Joint Conference*, vol. 40, pp. 617-625, 1972.

Ceri82.
S. Ceri and G. Pelegatti, "A Solution Method for the Non-Additive Resource Allocation Problem in a Distributed System Design," *Informatiion Processing Letters*, vol. 15, no. 4, 31 October 1982.

Chandy83.
K.M Chandy, J. Misra, and L.M Haas, "Distributed Deadlock Detection," *ACM Transaction on Computer Systems*, vol. 1, no. 2, May 1983.

Chen73.
P.P-S. Chen, "Optimal File Allocation in Multilevel Storage Systems," *Proceedings of AFIPS 1973 National Computer Conference*, vol. 42, pp. 277-282, AFIPS Press, Arlington,VA, 1973.

Chen80.
P. Pin-Shan. Chen and J. Akoka, "Optimal Design of Distributed Information Systems," *IEEE Transaction on Computers*, vol. C-29, no. 12, pp. 1068-1080, December 1980.

Cherit83.
D.R. Cheriton and W. Zwaenepol, *The Distributed V-kernel and its Performance*, 9th, SOSP, Operating Systems Reviewing, 17, pp. 129-140, November 1983.

Chou80.
T.C.K Chou and J.A. Abraham, "Load Balancing in Distributed Systems," *IEEE Transaction on Software Engineering*, vol. SE-8, no. 4, pp. 401-412, July 1980.

Chu73.
W.W. Chu, "Optimal File Allocation in a Computer Network," in *Computer-Communication Networks*, ed. N. Abramson and F.F. Kua, Prentice-Hall, 1973.

Clark78.
D.D. Clark, K.T. Pogran, and D.P. Reed, "An Introduction to Local Area

Networks." *Proceedings of the IEEE*, vol. 66, no. 11, pp. 1497-1517, November 1978.

Cox66.
D.R. Cox and P.A.W. Lewis, *The Statistical Analysis of Series of Events*, Methun & co Ltd, London, 1966.

Dandam85.
S. P. Dandamundi and R. B. Bunt, "Refined models of Program Behavior for Improved Memory Management," *Proceedings of the Eighteenth Annual Hawaii International Conference on System Science*, 1985.

Dennin72.
P.J. Denning and S.C. Schwartz, "Properties of the Working Set Model," *Communication of ACM*, vol. 15, no. 3, pp. 191-198, March 1972.

Dennin80.
P.J. Denning, "Working Sets Past and Present," *IEEE Transaction on Software Engineering*, vol. SE-6, no. 1, pp. 64-84, January, 1980.

Dowdy82.
L.W. Dowdy and D.V. Foster, "Comparative Models of the File Assignment Problem," *ACM Computing Surveys*, vol. 14, no. 2, pp. 286 -313, 1982.

Dowdy84.
L.W. Dowdy and et al. , "Dynamic File Assignment for Local Area Networks: Models and Analysis," Vanderbilt Univeristy CS-84-06, 1984.

Dutta82.
A. Dutta, "On the Modelling Of Multiple Copy Update Costs for File Allocation in a distributed Database," University of Rochester Technical Report , November 1982.

Easton75.
M.C. Easton, "Model for Interactive Data Base Reference String," *IBM Journal of Research and Development*, vol. 19, no. 6, pp. 550-556, November 1975.

Fajman73.
R. Fajman and J. Borgelt, "Wylbur: An Interactive Text Editing and Remote Data Entry System," *Communication of ACM*, vol. 16, no. 5, pp. 314-327, May 1973.

Fisher80.
M.L. Fisher and D.S. Hochbaum, "Database Location in Computer Networks," *Journal of the ACM*, vol. 27, no. 4, pp. 718-735, October 1980.

Floyd86.
R. Floyd, "Short-Term File Reference Patterns in a UNIX Environment," TR177, Computer Scinece Department, Univerity of Rochester, March 1986.

Foster81.
D.V. Foster, L.W. Dowdy, and J.E Ames, "File Assignment in Memory Hierarchies," *Computer Networks*, no. 5, pp. 341-349, September 1981.

Gavish86.
B. Gavish and H. Pirkul, "Computer and Database Location in a Distributed Computer System," *IEEE Transaction on Computers*, vol. C-35, no. 7, July 1986.

Hartig75.
J.A. Hartigan, *Clustering Algorithms*, John Wiley & Sons, 1975.

Hughes73.
P.H Hughes and G. Moe, "A Structural Approach to Computer Performance Analysis," *Proceedings AFIPS 1973 Spring Joint Computer Conference*, pp. 109-120, AFIPS Press, Arlington,VA, 1973.

IBM73.
IBM, in *OS SMF GC28-6712-7 (eight edition)*, April 1973.

IBM78.
IBM, in *OS/VS2 MVS System Programming Library System Management Facility (SMF) (Gn28-2903)*, May 1978.

Irani82.
K.B. Irani and N.G. Khabbaz, "A Methodology for the Design of a Communication Networks and the Distribution of DAta in Distributed Supercomputer Systems," *IEEE Transactions on Computers*, vol. C-31, no. 5, May 1982.

Jenny82.
C.J. Jenny, "Methodologies for Placing Files and Processes in Systems with Decentralized Intelligence," IBM Computer Science Research Report RZ1139, IBM Zurich Research Labratory, April 1982.

Kendal75.
M.G. Kendall and A. Stuart, *The Advanced Theory of Statistics*, 2 and 3, Hafner Publishing Company, 1975.

Kleinr76.
L. Kleinrock, in *Queing Systems - Volume 2: Computer Applications*, Wiley, New York, 1976.

Kronen86.
N.P Kronenberg, H. Levy, and W.D. Strecker, "VAX Clusters: A Closely-Coupled Distributed System," *ACM Transactions on Computer Systems*, vol. 4, no. 2, pp. 130-146, May 1986.

Laning83.
L. J. Laning and M. S. Leonard, "File Allocation in a Distributed Computer Communication Network," *IEEE Transaction on Computers*, vol. C-32, no. 3, pp. 232-244, March 1983.

Lawrie82.
D. H. Lawrie, J. M. Randal, and R. R. Barton, "Experiments with Automatic File Migration," *IEEE Computer Society, Computer*, vol. 15, no. 7, pp. 45-55, July 1982.

Lazows84.
E. D. Lazowska, J. Zahorjan, D. R. Cheriton, and W. Zwaenepoel, "File Access Performance of Diskless Workstations," Technical report 84-06-01, Department of Computer Science, Universtity of Washington, Seattle, June 1984.

Levin76.
K.D. Levin and H.L. Morgan, "A Dynamic Optimization Model for Distributed Databases," University of Pennsylvania, Dept. of Decision Science 76-09-01, 1976.

Lewis73.
P.A.W. Lewis and G.S. Shedler, "Micromodels for Sequences of Page Exceptions," *IBM Journal of Research and Development*, 1973.

Lo84.V.M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," *Proceedings of the Fourth International Conference on Distributed Computing Systems, IEEE*, pp. 422-431, 1984.

Mahmou76.
S. Mahmoud and J.S. Riordon, "Optimal Allocation of Resources in a Distributed Information Network," *ACM Transaction on Database Systems*, vol. 1, no. 1, pp. 66-78, March 1976.

Majuma86.
S. Majumar and R. B. Bunt, "Measurement and Analysis of Locality Phases in File Referencing Behavior," *Proceedings of Performance '86, Joint Conference on Computer Performance Modelling, Measurement and Evaluation*, vol. 14, no. 1, pp. 180-192, May 1986.

McKusi83.
M.K. McKusick, W.N. Joy, S.J. Leffler, and R.S. Fabry, "A Fast File System for UNIX," University of California, Berkeley, report CSRG TR/7, July 27, 1983.

McQuil77.
J.M. McQuillan and D.C. Walden, "The ARPA Network Design Decisions," *Computer Networks*, vol. 1, no. 5, pp. 243-289, 1977.

Mincer84.
J. Mincer-Daszkiewiez and Z. Weiss, "An approach to Program I/O Reference

Behavior Modeling," *Performance Evaluation*, vol. 4, no. 4, Novemeber 1984.

Mitche82.
J.G. Mitchell and J. Dion, "A Comparison of Two Network-Based File Servers," *Communication of ACM*, vol. 25, no. 4, pp. 233-245, April 1982.

Morgan77.
H.J Morgan and K.D Levin, "Optimal Program and Data Location in Computer Networks," *Communication of ACM*, vol. 20, no. 5, pp. 315-322, May 1977.

Morris86.
James H. Morris and et al, "Andrew: A Distributed Personal Computing environment," *Communication of the ACM*, vol. 29, no. 3, pp. 184-201, March 1986.

Mullen85.
S. Mullender and A. Tanenbaum, "A Distributed File Service Based on Optimistic Concurrency control," *Tenth ACM Symposium on Operating System Principles*, pp. 51-62, Orcas Island, Washington, U.S.A, 1-4 December 1985.

Needha82.
R.M Needham and A.J Herbert, in *The Cambridge Distributed Computing System*, Addison-Weasley, Reading Mass., 1982.

Nikola86.
C. Nikolaou, G. Kar, D. F. Ferguson, and G. Leitner, "Allocation and Relocation of Processes in a Distributed Computer System," in *Current Advances in Distributed Computing and Communication*, ed. Y. Yemini, Computer Science Press, 1986.

Ouster85.
J. Ousterhout and et al., "A Trace Driven Simulation of the Unix 4.2 BSD File System," *Tenth ACM Symposium on Operating System Principles*, pp. 15-24, Orcas Island, Washington, U.S.A, 1-4 December 1985.

Pope84.
B. Pope, "Dynamic Access of CMS files," IMB Yorktowm RC10483, 16 April 1984.

Porcar82.
J. Porcar, "File Migration in Distributed Systems," Ph.D Thesis, U.C. Berkeley, 1982.

Powell83.
M.L. Powell and B.P. Miller, "Process Migration in DEMOS/MP," *ACM Operating Systems Review*, vol. 117, no. 5, October 1983.

Price81.
C. C. Price, "The Assignment of Computational Tasks among Processors in a Distributed System," *National Computer Conference, 1981*, 1981.

Ramamo83.
C. V. Ramamoorthy and B. W. Wah, "The Isomorphism of Simple File Allocation," *IEEE transaction of Computers*, vol. c-32, no. 3, March 1983.

Rao79.
G. S. Rao and H. S. Stone, "Assignment of Tasks in a Distributed Processor System with Limited Memory," *IEEE Transaction on Computers*, vol. c-28, no. 4, April 1979.

Revell75.
Ron Revelle, "An Empirical Study of File Reference Patterns," IBM RJ 1557, April 21, 1975.

Ritchi74.
D. M. Ritchie and K. Thompson, "The UNIX Timesharing System," *Communication of ACM*, vol. 17, no. 7, pp. 365-375, 1974.

Ross83.
S.M. Ross, *Introduction to Stochastic Dynamic Programming*, Academic Press, 1983.

Ruan87.
Z. Ruan and W.F. Tichy, "Performance Analysis of File Replication Schemes in Distributed Systems," *Eleventh ACM Symposium on Operating System Principles*,

pp. 205-215, Banff, Alberta, Canada., 11-17 May 1987.

Satyan81.
M. Satyanarayanan, "A Study of File Sizes and Functional Lifetimes," *Proceedings of Eighth Symposium on Operating System Principles, ACM SIGOPS*, vol. 15, no. 5, December 1981.

Satyan85.
M. Satyanarayanan, "The ITC Distributed File System: Principles and Design," *Tenth ACM Symposium on Operating System Principles*, pp. 35-50, Orcas Island, Washington, U.S.A, 1-4 December 1985.

Schroe85.
M. Schroeder, D. Gifford, and R. Needham, "A Caching File System for a Programmers Workstation," *Tenth ACM Symposium on Operating System Principles*, pp. 25-34, Orcas Island, Washington, U.S.A, 1-4 December 1985.

Segal76.
A. Segal, "Dynamic File Assignment in a Computer Network," *IEEE transactions on Automatic Control*, vol. ac-21, no. 2, April 1976.

Segal79.
A. Segal, "Dynamic File Assignmnet in a Computer Network- Part II: Decentralized Control," *IEEE Transaction on Automatic Control*, vol. AC-24, no. 5, October 1979.

Sheltz86.
A.B. Sheltzer and G.J. Popek, "Internet Locus: Extending Transparency to an Internet Environment," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 11, pp. 1067-1075, November 1986.

Smith75.
A.J. Smith, "A Locality model for Disk Reference Patterns," *Proceedings of IEEE Computer Society Conference*, pp. 109-112, February 1975.

Smith76.
A.J. Smith, "Analysis of the Optimal Look-ahead Demand Paging Algorithms," *SIAM Journal of Computing*, vol. 5, no. 4, December 1976.

Smith78.
A.J. Smith, "Sequentiality and Prefetching in Database Systems," *ACM Transaction on Database Systems*, vol. 3, no. 3, pp. 223-247, September 1978.

Smith81.
A.J. Smith, "Optimization of I/O Systems by Cache Disks and File Migration: A Summary," *Performance Evaluation*, vol. 1, pp. 249-262, 1981.

Smith81b.
A.J. Smith, "Analysis of long term File Reference Patterns for Application to File Migration Algorithms," *IEEE Transaction on Software Engineering*, vol. se-7, no. 4, July 1981.

Smith81c.
A.J. Smith, "Input/Output Optimization and Disk Architectures: A Survey," *Performance and Evaluation*, vol. 1, pp. 104-117, 1981.

Smith82.
A.J. Smith, "Cache Memories," *ACM Computing Surveys*, vol. 14, no. 3, pp. 473-530, September 1982.

Smith82b.
A.J. Smith, "Long-term File Migration Development and Evaluation of Algorithms," *Communication of ACM*, vol. 24, no. 8, pp. 521-532, August 1982.

Smith85.
A.J. Smith, "Disk Cache-Miss Ratio Analysis and Design Considerations," *ACM Transactions on Computer Systems*, vol. 3, no. 3, pp. 162-202, August 1985.

Smith85b.
A.J. Smith, "Problems, Directions and Issues in Memory Hierarchies," *Proceedings of the Eighteenth Annual Hawaii International Conference on System Science*,

pp. 468-476, 1985.

Spirn77.
J. R. Spirn, *Program Behavior: Models and Measurement*, Elsevier, 1977.

Stritt77.
E.P. Stritter. "File Migration." Stanford University Technical Report STAN-CS-77-594, January 1977.

Sturgi80.
H. Sturgis, J. Mitchell, and J. Israel, "Issues in the Design and Use of a Distributed File System," *Operating Systems Review*, vol. 14, pp. 55-69, 1980.

Svobod81.
L. Svoboda, "A Reliable Oject Oriented Data Repository for a Distributed System," *Communication of the ACM*, 1981.

Svobod82.
L. Svobodova, "File Servers for Network Based Distributed Systems," IBM Computer Science Research Report RZ1258, IBM Zurich Research Labratory, 17 October 1982.

Swineh79.
D. Swinehart, G. Mcdaniels, and D. Boggs, "WFS: A Simple Shared File System for a Distributed Environment." *Proceedings of the 7th SOSP, Operating Systems Review*, vol. 13, no. 5, pp. 9-17, 1979.

Tanenb85.
A.S. Tanenbaum and R. van Renesse, "Distributed Operating Systems," *ACM Computing Surveys*, vol. 17, no. 4, December 1985.

Thomps87.
J.G Thompson, "Efficient Analysis of Caching Systems," Report No. UCB/CSD 87/374, Computer Science Division (EECS), University of California, Berkeley, October 1987.

Visin80.
I. Visin, C. Ross, and E. Russell, *ASP User Guide*, SLAC Computing Services SCS, Menlo Park, CA, March 1980.

Wah84.
B.W. Wah, "File Placement in Distributed Computer Systems," *IIEEE Computer*, pp. 23-32, January 1984.

Walker83.
B Walker and et al, *The Locus Distributed Operating System*, 9th SOSP Operating Systems Reviewing, 17, pp. 49-70, November 1983.

Wang85.
Y. Wang and R.J.T. Morris, "Load Sharing in Distributed Systems," *IEEE Transactions on Computers*, vol. c-34, no. 3, pp. 204-217, March 1985.

Wolff88.
R.W. Wolff, in *Stochastic Modeling and the Theory of Queues*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

Woodsi86.
C. M. Woodside and S. K. Tripathi, "Optimal Allocation of File Servers in a Local Network Environment," *IEEE Transaction on Software Engineering*, vol. SE-12, no. 8, August 1986.

Wupit83.
A. Wupit, "Comparison of UNIX Network Systems," *Proc. of ACM Conference on Personal and Small Computers*, pp. 99-108, San Diego, Ca, December 7-9 1983.

Zadeh71.
L. A. Zadeh, "Similarity Relations and Fuzzy Orderings," *Information Sciences*, vol. 3, no. 2, pp. 177-200, April 1971.

Zhou87.
S. Zhou, "Performance Studies for Dynamic Load Balancing in Distributed Systems," Ph.D Thesis, U.C. Berkeley, December 1987.