THE UNIVERSITY OF CALGARY

File Allocation in a Local Area Network

by

Frank Deur

A Thesis

Submitted To The Faculty Of Graduate Studies

In Partial Fulfillment Of The Requirements For The

Degree of Master of Science

Department of Computer Science

Calgary, Alberta

April, 1989

ISBN 0-315-50408-0

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled, "File Allocation in a Local Area Network" submitted by Frank Deur in partial fulfillment of the requirements for the degree of Master of Science.

Supervisor, Dr. Anton W. Colijn
Department of Computer Science

Dr. Michael R. Williams
Department of Computer Science

Dr. Kenneth D. Loose
Department of Computer Science

Dr. Peter R. Newsted
Faculty of Management

Date April 12, 1989

Abstract

The advent of a number of computers connected in a network has brought to the foreground some old problems. Among these is the problem of file allocation, or file distribution, among the nodes of the network. Two possibilities exist when deciding when and how to allocate files.

The first is that of fixed allocation. In this situation files are allocated once to a particular node and remain there from then on. The second possibility is dynamic allocation. Here files are initially assign to nodes. From this point on however files may transfer to other nodes depending upon a number of factors.

The research is designed to explore the possibilities of dynamic file allocation through the use of a heuristic algorithm. In particular the areas of frequency of access, workload, and user response time will be explored.

A prototype is developed in order to test the validity of the results when compared to theoretical results.

Acknowledgements

I would like to thank my supervisor, Dr. Anton Colijn for his support and direction in this research project. His insight, knowledge and patience were fundamental to the success of the thesis. I am also very grateful to Dr. Michael Williams for his willingness to endure my initial drafts, for his organizational skills were instrumental in producing a much better product. I would also like to thank my friend David Cleland for his help in the mathematical areas of this thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Centralized System Fundamentals

Databases and DBMS (Database Management Systems) were born out of necessity, for file environments suffered from a number of problems. Among these were the problems of data redundancy and consistency of data. An application program that created and even maintained its own files of data and its own particular data structures is quite usable as long as it is the only one accessing this data. The major problem with this approach is the fact that any other application program using this data must take into account this particular file and data structure. If another application program is created that uses files of essentially equivalent data, and possibly structures it differently from the first example, a problem has now been created in the area of data redundancy and possibly consistency.

Data redundancy may be described as the same data being stored in a number of different locations. At the very least this is an inefficient use of memory, but more often other considerations such as data updates are also involved. If a data update is required, it is now quite possible for one occurrence of the data to have a new value and yet another to not have been updated as yet. This is usually referred to as the inconsistency problem. A further problem that often arises is that of an application program needing data from files which have a different structure as well as containing data of different types. This tends to make access to data

much more difficult because applications programs must now take these differences into account. DBMS were designed to eliminate some of these problems as well as provide the Database Administrator with centralized control of its operations [14]. In order for a DBMS to solve some of these problems it is required that an applications program's call for data be intercepted by the DBMS and it, in turn, do the actual data accessing.

This abstraction brings forward the concept of different views of the data. In a centralized database, data independence is achieved through the use of the notions of a conceptual schema, an external schema and a logical schema. The external view may be defined as the view of the data from that of an individual program or user. The internal view may be said to be a low level representation (although still one level removed from the physical level) consisting of actual records complete with how values are represented, indexes and pointers [14]. The conceptual view is an intermediate view between these two. It consists of the entire data base viewed as it really is but without its pointers and indexes and values required in the internal view.

One of the real aims of a DBMS is to achieve data independence. This will then allow application programs to be unaffected by changes to either the data structures or the database. This data independence can then be used to allow a single occurrence of a data item to meet the needs of multiple different applications programs and so avoid duplication of data and related update and inconsistency problems.

Closely tied to the idea of data independence through the use of a database is the idea of what structure the data should actually take. Initially there were generally

three types: Hierarchial such as IMS by IBM; Network such as IDMS by Cullinet; and Inverted like ADABAS by Software AG. In more recent years almost all new database systems are of the Relational type such as INGRES and DB2. Although there is little doubt that the relational approach is the present trend [14] it still has difficulty competing in terms of speed with older database packages such as IMS. IMS has a transaction speed of approximately 100 - 1000 transactions per second due to relatively simple updates and retrievals. In comparison IBM's DB2 is much slower due to software overhead; even so DB2 is easier to program due to the high level DBML (Database Management Language) and transaction speed should soon improve [7].

Of prime importance when designing any database is to take into account the use to which it will be put. Here file size, file activity ratios and volatility must be considered. A file activity ratio (FAR) may be defined as:

- FAR = number of records accessed/number of records in file

A FAR value approaching 1 for any job indicates that almost all records were accessed, so READS are very important. A FAR number approaching 0 means that few records were accessed and so random access becomes most important. File volatility ratio (FVR) may be measured as follows:

- FVR = (number of additions + number of deletions) / number of records in original file.

If this FVR value approaches 1 then WRITE and DELETE should be made as fast as possible, if close to 0 then READS are again most important [25]. An example

of a file with low volatility and activity would be a dictionary or thesaurus, here few changes or additions are made so random access reads are of prime importance. This is in contrast to a bank chequeing account where there may be numerous updates and so WRITES should be given a higher priority than in the previous example.

Most users care little about underlying concepts or mechanisms and are primarily concerned with product or program usability. In this light, response and performance become the prime criteria. Performance depends directly on the number of I/O operations that must be done to do a particular job [25].

Traditional databases tend to be I/O bound in that they spend a relatively large amount of time waiting for I/O operations to be performed while spending relatively little time on actual computations. There has been general agreement that I/O costs are to a large extent a limiting factor on reducing user response time. The advent of faster disk drives has reduced the access time somewhat but it still remains, by far, the single most important factor in any sort of database retrieval. Due to the laws of elementary physics, the speed at which mechanical parts can be moved or caused to change direction is limited and reductions in access times, such as those that have been seen in primary memory, are unrealistic.

In an effort to reduce I/O times for file access, a number of different techniques have been developed in the way files are actually stored. These range from sequential files, random access files and multi-keyed files. Each of these has its own advantages and tradeoffs but have been developed and modified to such a point that major reductions in I/O times are unlikely to be achieved by farther manipulation or modification in this area.

The use of multiple secondary storage devices with large centralized databases

signaled a further complication. Now not only must the way files are stored be looked at, but also which drive unit it will be stored on must be considered. This was essentially the start of what is now referred to as the file allocation problem (FAP).

Further complicating an already increasingly difficult problem is the fact that files are not always used in the same way by either the same or different users. An example of this is in the use of different data files with the same applications program.

1.2 Distributed System Objectives

The time when a large centralized database was the only possible solution to the storage and processing problems of a large organization has passed. The advent of the mini and micro computer, as well as related network hardware and software has been shown by both the banking system and the airline industry to be a viable alternative to the centralized concept, while at the same time still allowing the user to view the system as if it were a single centralized entity. Keeping this unified view is one of the major objectives of a distributed system.

This is not to say however that all the problems of either centralized or distributed systems have been overcome or should be minimized. On the contrary, what appears to have happened is that many of the problems related to centralized systems have been carried over to distributed systems. Along with this is the fact that distributed systems have created some of their own unique problems. One of the major differences is that while in centralized systems I/O was to be minimized, in a distributed

system one important aim is to minimize the total amount of data transmitted and messages sent within the system. Yet even with the rise of these new problems, as well as some remaining from the older centralized systems, there are few who will disagree with the fact that distributed systems have much to offer.

Some of the global objectives of distributed systems could be summarized as follows:

1. connecting existing databases
2. reducing communications costs
3. local autonomy
4. availability
5. increased efficiency
6. maintain a centralized view

Some of the more important aspects of distributed system will be discussed in the following sections.

1.3 Distributed Systems Fundamentals

A number of distributed systems were introduced in the late 70's and early 80's. These ranged from the first system called SDD-1 from Computer Corporation of America, to System R/Star from IBM and then Distributed Ingres from Relational Technology. As with almost any new system or product there were the inevitable

areas in each of these that have been found lacking, due to either poor or incorrect decisions on the part of the designers, or due to the fact that there are some very difficult problems involved that have as yet not been solved satisfactorily. Distributed Systems are naturally more complex than centralized ones. An example of the added complexity is the fact that while in a Centralized system only an External, Conceptual and Internal view are required to achieve data independence, in a distributed database system the concept of data transparency must be added. Data transparency means that the applications program should not need to know at which site the required data is maintained. If data transparency is not achieved in a distributed database then applications programs must take into account possibilities which soon become almost unmanageable. For example if two files are requested, are they both on this site, are they on different sites and so on.

In centralized databases data redundancy or replication is minimized in an effort to ensure data consistency. In a distributed database system it can be shown that a certain degree of data replication can be a positive factor. Although it is true that the same negatives, such as the update problem and data consistency, still exist, as they did in the centralized system, there are now certainly also some positive aspects. For example, by having data at the needed site it can quickly and easily be accessed. With data replication the failure of one site and its database still allows other sites to use this replicated data in processing.

In tandem with the availability question is the concept of increased site autonomy. With both data and processing power available locally, sites can more easily answer questions of concern to them, as well as reduce demand on centralized facilities and so increase overall efficiency.

1.3.1 Topologies

The most common type of computer networks are, Star, Ring, Completely Connected and Bus as shown in figure 1.1.

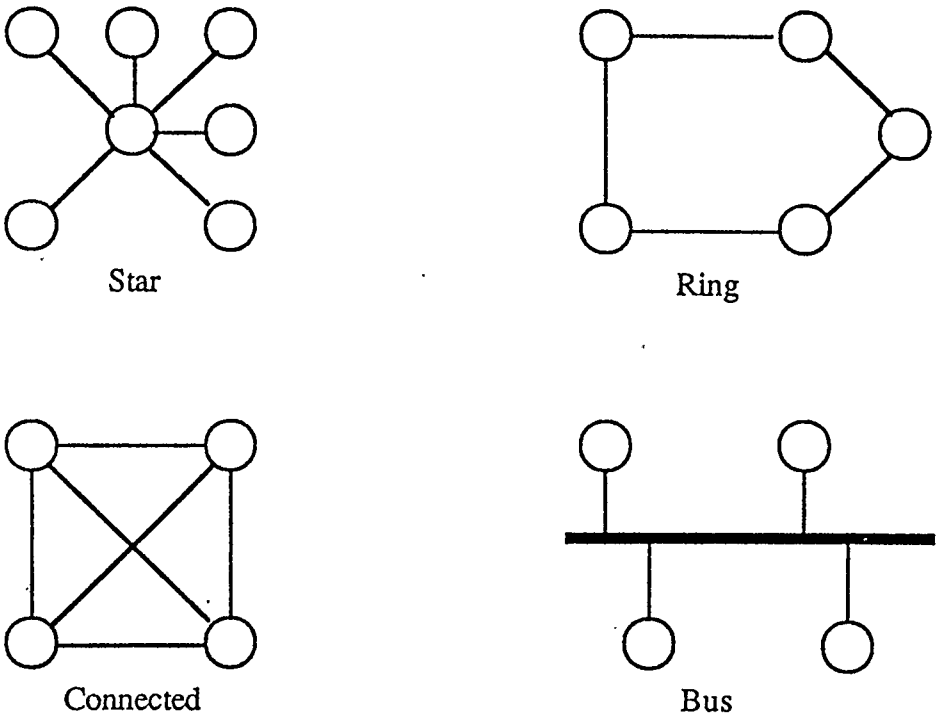

Figure 1.1: System Topologies

Each of these has its advantages and drawbacks, often inherent in the design, but also due to the choice of application. One of the more obvious drawbacks of the Star design is that if the central site fails, no communication can take place between any other sites. This design however does allow a high degree of control over both data and communication flows. Some Ring networks have a facility to bypass a failed site

and in this way allow the network to continue to function without that site. The failure of subsequent sites may be allowed in some Ring networks. It can be seen from this that the Ring design has an advantage over the Star design in that if any site fails the other sites may still communicate with each other. The Completely Connected design has the advantage of allowing any site directly to communicate with any other site. This tends to be feasible for only a small number of sites for as the number of sites grows, the number of connections grows even quicker. The Bus design is certainly popular especially for a LAN (Local Area Network) but is unfeasible for Long Haul Networks.

1.3.2 Distributed Database Manager

In a centralized system the DBMS is the software package that controls and regulates all access to the database. In a distributed database system this software is also required, except now each site has its own DBMS which could now be more appropriately called a site or local DBMS. Due to the fact that each site is but one node of the network and must be able to communicate with other sites, some extra components are required to complement and supplement each DBMS. A DDB (Distributed Database) component must be added, at each site, so as to enable intersite communication and so allow each site to function as part of the network.

A more complete set of functions provided by a DDB manager could be described in the following ways:

1. To provide location transparency for the data used by the applications program. A local request for data may or may not require a remote data request, either way it should be immaterial to the applications program.

2. Remote database access to another site.

3. Code to complement the local DBMS in the area's of concurrency, monitoring, statistics and recovery.

1.4 Distributed System Advantages

In most large organizations there is the head office concept, but this is complemented with the concept of districts of control and branch plants that are to a large extent autonomous. With a large centralized database an attempt was made to have this often physically distributed organization of central office and branch plants fit into a centralized mold. It would seem that the idea of a network and a distributed database system would more closely fit and meet the needs of a distributed company organization.

1.4.1 Availability and Reliability

The reliability of a distributed system can be dramatically improved, as compared to that of a centralized system, for in a centralized system if the main site fails all processing stops. In the distributed system, the effects of the failure of one site are more likely to range from no effect at all, to a partial degrading of the system capacity, but even this degradation must be preferred to a total failure. Availability of data may be greatly enhanced in a distributed system in that if data is replicated on another site, the failure of one site will still allow other sites to have access to this replicated data.

1.4.2 Site Autonomy

A branch plant or site can be given a degree of responsibility for its own database administration ranging from very little to complete. Site autonomy allows the local organization a much greater degree of control over data, especially over data that only they need, as well as making it much less dependent on the database located at some other, possibly far away, location.

1.4.3 Increased Efficiency

This autonomy can also help achieve much higher efficiency. Imagine for instance the example of a main office in Calgary and a branch plant 2000 kilometers away with no local autonomy. All processing would have to be sent to Calgary, processed and the results sent back. It is obvious that this incurs extra costs in the area of communication and time delays. All other factors remaining equal, local processing would at least eliminate the communication and time costs.

Change is a part of any organization, as few remain static for long. Distributed databases can more easily allow for change than can centralized databases. As the network grows new sites may be added with a minimum of disruption. The added benefits of allowing data replication and increased processing power will likely result in a network that is speedier and more resistant to failures.

1.5 Distributed System Difficulties

There is little doubt that distributed systems do solve some of the problems found in centralized systems, but many of the same problems still exist together with some

new ones, as well as some new versions of old ones. In the area of query processing it is now possible that part of the data required, data that resides on another site, will be inaccessible due to the failure of that particular site. This problem may be partially solved by replication of data but this then introduces the problem of update control and data consistency.

The problem of ensuring consistency of the data after an update spans a very wide spectrum. At the one end, the system that does no updates at all. In a situation where only READS occur there is no update or consistency problem. This situation is very rare however and a more realistic situation would be one of at least a number of updates, even for a READ intensive situation such as an name and address database. In a centralized database with no replication a lock may be put on a file or record that needs to be changed, changes made, and the lock removed so that subsequent reads are again allowed and no inconsistency occurs. In a distributed database with file replication at more than one site, any change in data at one site should also occur at the other site or else the inconsistency problem comes into play. This simultaneous updating at more than one site may be quite difficult in real time, due to factors such as, another and different, change occurring to the data at the other site simultaneous with the first, or possibly the other site has even failed at this time. It can be seen that this is not a trivial problem, it remains one in which there are still many unanswered questions.

1.5.1 Communications

One of the most apparent and obvious problems is that of communications. This added factor of communications adds complexity to the overall system. Compound-

ing this is the problem of failure. How does one site determine whether or not another site has failed, or is it a communication line failure? This point may have major implications since it is now possible for independent sites to continue updating data of which another site is unaware.

Long Haul and Local Area Networks

Communication systems can really be subdivided into two areas, that of Long Haul Networks, such as Arpanet and Datapac, and Local Area Networks such as Ethernet. It can be argued that failures may occur equally in both Long Haul and Local Area Networks (LAN), but there is little doubt that Long Haul Networks data transfer, with data rates in the area of 25,000 bytes per second, is much slower than the average LAN. If a lot of data must be transmitted between sites the time taken for communications becomes an important factor for not only does the file transfer take place but it may now become unavailable to any other site.

Message Passing and RPC's

Not only does the actual data that is sent to a site need to be taken into account but also the type of message passing and RPC (Remote Procedure Call) used. A number of systems exist which require messages between machines in the form of acknowledgement for send or receive. It has been found that to send a message and receive an acknowledgement may cost in the neighborhood of 5000 to 10000 machine instructions of operating system and communication system control [15].

Versions and Temporaries

A scheme implemented by some systems to reduce data transfer between sites is that of Versions and Temporaries. Applications exist which do not need the most up to date information. Quite possibly the information required, although out of date is still sufficient for this particular need. This is done by applying a version number to the data and the process and then comparing this to the application's version number.

Temporaries are often created at a site that does not have a copy of the data but requires it more then once. If this site often needs the same data for a particular application a temporary copy of the data maybe created and stored at this site for a period of time.

1.5.2 File Allocation Problem

Placement of Files

In centralized systems the need to minimize I/O operations was often given a high priority when making decisions on which device a file would be stored. This file allocation was static in nature with the DBA making decisions based on device speeds, channel capacities and access frequencies. In a centralized system changes in file usage can affect system performance dramatically but as the following example will show, in a distributed system there are even more ways in which system performance may be affected. Consider the case where file usage remains the same and that the site that normally generates a certain query also contains the required data file to answer this query but that the query now originates from a different site. If this new querying site has none of the data required then there is little doubt that there are

extra costs involved in generating results to this query. In a distributed database system the site of a query can have a major impact on system performance quite unlike a centralized system where the choice of which terminal is used has little effect on performance. It is for this and other reasons that file allocation and specifically dynamic file allocation (in which files can be automatically moved to the sites where they are most in demand) can be of such importance in a distributed database system. When attempting to allocate files what criteria should be followed? Should the most current usage be used as an indicator of future usage or should prior usage be used in determining file allocation or re-allocation? This area will be looked into extensively in Chapter 4.

Replication of Files

Notwithstanding the additional costs of time, complexity and processing power, the potential gains to be made in distributed systems far outweigh these costs.

While in a centralized system the replication of files was to be avoided if at all possible, this is not the case in distributed systems. As was shown previously there are certain advantages to replicated files, such as increased availability, efficiency and autonomy. Some very difficult problems arise because of this. If some replication is good, is total replication best, or partial replication? The following example will show the point. If a network has 10 sites and only 10 files exist and each file may only exist on one site, the number of possibilities of where to put a particular file are 10^{10}. If full replication is allowed at each site the possibilities increase to 2^{100}, and this is only a very small example, quite atypical of a large commercial database.

Primary and Secondary Copies

When file redundancy is introduced it is self evident that introducing a second copy has more benefit than the introduction of a third, fourth or subsequent copies. As with many applications, there is decreasing benefit or return as more is introduced. The difficulty with this problem is to determine at what level of replication the benefits still outweigh the costs incurred. One answer to this has been to allow only 2 instances of a file to exist. One will be designated a Primary copy and the other the Secondary copy. With this scheme, reads are allowed from both copies, but when an update is to occur the write lock is not released on the primary copy until the secondary copy has also been updated, thus giving consistency. Of real benefit is the possibility of continued processing, due to the availability of a second copy, if a failure occurs. If a site failure occurs at the primary copy site the secondary site can now become the primary copy and take over. When the failed site again becomes active it may be updated from the new primary copy and resume being the primary or it may continue and be designated the secondary copy. This is not without its pitfalls however, for how does a site determine that another site has failed? As was pointed out earlier, is it really a site failure or is it a failure in the communications system.

1.5.3 System Catalog

Unlike the centralized systems catalog, the distributed systems catalog must be extended to include information about the system itself eg. at which site a particular file is stored. With the system's catalog existing as really just another file, the question arises, should it be replicated as well? Not only does the replication question

arise but also the question of partitioning. Should one site have a complete systems catalog or should each site have that part of the catalog that applies only for data stored at that site? With query optimization being an integral part of most databases a systems catalog now often contains values relating to how many rows and columns are in each table. A number of possibilities exist here and as will be shown latter, different systems have used different approaches. The real options are these:

1. Completely Centralized: the catalog is stored at only one site.
2. Fully Replicated: the catalog is stored at every site.
3. Completely Partitioned: each site has a catalog only for its own data.
4. A combination of the three above.

In analyzing these four approaches it would seem that a Completely Centralized approach goes counter to the underlying theory of a distributed system in that the whole system depends on one site. If that one site fails the whole system fails, not much different than the original centralized DBMS approach.

The second approach of Fully Replicated has much merit in that each site can easily determine where data is as well as allowing it a high degree of autonomy. It does suffer from at least one serious drawback however, as with any replicated approach the higher the degree of replication the more problematic becomes the area of updates. If the system's catalog contains values stating how many rows and columns are in each table an update must be distributed to all sites each time a change occurs. This can be very resource intensive as well as problematic if a site

fails either before or during an update. If the system is highly read intensive then Fully Replicated may be a proper alternative if there also exists a need for high availability.

The Completely Partitioned approach appears reasonable from the standpoint that the complete system catalog can be derived from the individual catalogs due to the fact that each site has a catalog for the data at that site. Any updates need be done only to one particular site so the update problem is reduced. A problem arises however in the area of remote data requests. In the event of a remote data call, the originating site must do a broadcast to all other sites to determine where the required data is situated. In a LAN this may be feasible although even here time delays again become a factor. In a Long Haul Network it would appear to be too expensive to really be practical, due to the communications costs for the broadcast, as well as acknowledgements and possible data transfers.

There has as yet not been one generally accepted answer, as for example System R/Star and Distributed Ingres both use different methods which will be described in a later section.

1.6 Overview

The increased use and growing importance of distributed systems and distributed data base systems has led to a new awareness of the importance of file allocation within distributed systems. In this thesis the area of dynamic file allocation is explored and data presented in support of dynamic file allocation as both possible and advantageous to system performance. Chapter 2 presents earlier work on the file

allocation problem. This work is divided into two distinct groups, proposed solutions that attempt to minimize systems costs such as storage and communications and solutions which try to maximize system performance as measured by user response time and system throughput. Chapter 3 describes R/Star and Ingres and their implementation of data base systems. These two were chosen because their methods for file allocation are quite distinct with R/Star using its own data base system and Ingres developers deciding to use what was already available in UNIX. In Chapter 4 a heuristic algorithm for dynamic file allocation is developed and tested. Evidence is presented which supports the premise that significant benefit may be obtained by implementing dynamic re-allocation.

Chapter 2

File Allocation Solutions

With the advent of computer networks which do not just share a common database but actually distribute the database among the nodes of the network, a number of benefits have been obtained, at the cost of creating some very difficult problems. The increase in node autonomy, as well as file availability and parallel processing among nodes, has certainly increased overall productivity. Yet only part of the potential has been realized. With better allocation of files, dramatic reductions in I/O times are possible.

There appears little doubt that optimal file assignment can enhance system performance tremendously. The FAP (File Allocation Problem) has been studied extensively [41] and improvements have been made, but as yet no one has put forth an optimal solution.

Many proposed solutions or improvements were designed for centralized systems and may not be completely appropriate for distributed systems, for what may work well in a centralized system may not work as well in a distributed system or may even be counter productive.

It is important to decide what sort of system is being discussed, that of a single dedicated machine, a file server, or that of a distributed database among a number of sites. In a centralized system it has long been realized that I/O was an overriding factor and to minimize disk accesses was often the ultimate goal. This is not necessarily the case in DDB systems for here communications costs are often more

important and methods to reduce remote data calls or minimize acknowledge messages must be looked at first. Compounding this is the fact, that unlike a centralized system were data redundancy is to be minimized, in a DDB system, replication of files among sites is often desirable and thus the situation is quite different.

The proper allocation of files within a computer system is of prime importance if the nodes of the system are to be equally used or at least not have one underused while another suffers delays caused by overuse. If the solution were easy it would already have been solved, but it has not. It has been shown that the file allocation problem in a DDB is NP-Complete [19] [23]. So for all except a small number of files, in which case an exhaustive search may be done, algorithms for solving the problem have an exponential growth function and it soon becomes so time consuming and complex as to become unsolvable in a reasonable time.

The term "proper allocation" is also too general in that it could mean to minimize cost in the area of file reads and writes, remote procedure call (RPC) costs or it could be viewed in light of maximum performance where the performance goal is to minimize response time. One of the major problems lies in the fact that in an attempt to optimize one function, for instance cost, another function such as response time performance is likely to degrade and vice versa.

It would appear that one principle of prime importance in attempting to establish a file allocation for a distributed system is putting data that is usually referenced together at a common site.

Notwithstanding these facts, some basis must be established as a starting point for file allocation. Some file assignment strategies will be presented ranging from considering each file assignment separately and attempting to minimize costs, to

assigning multiple files and attempting to maximize performance through the use of heuristics.

The areas of system reliability and queuing delays will not be looked at in the first 2 solutions presented but will be included in the last 2 considered.

The placement of a single file within a system will be looked at first. Two simplifying assumptions will be made as a starting point. The first is that update communications and query communications have equal cost. The second assumption is that once a file has been assigned a location it is never moved to another site. Both of these assumptions will be relaxed in later models.

2.1 Single File Assignment

In a multiple user system with only one file on the complete system it can be seen that, for the situation of non replicated files, the cost to each user for a simple query is the same: a remote access, which breaks into time delay costs and communications costs. Improvement can take two roads, one for a read only situation, the other where updates are involved. In the read only situation the optimal situation would be complete replication of the file at each site, in this way eliminating time and communications costs. The more realistic situation, one which incorporates updates, must take into account the cost of updates in the areas of communications and consistency. A solution has been developed by Wah [41] which takes into account the costs of updates as well as factors such as frequency of use. File allocation is started by taking a single file and determining from queries and updates to this file which sites should be assigned this file. The overall objective is to minimize systems

costs while allowing all accesses. Once the initial file is assigned to the appropriate site or sites, other files are now allowed on the system and each allocated to the site or sites where the benefits outweigh costs. In this way a complete distributed file system may be built as shown by Ramamoorthy [35].

Shortcomings in this solution are not only in the areas of the original assumptions, in particular that of disregarding queuing delays, but also in the area of static file assignment, once assigned a file stays at that site. A further limitation is that no consideration is given to the interaction between files.

2.2 Program/Data Assignment

A solution developed by Morgan [33] specifically takes into account the interdependence of a program file to a data file. In this solution it is assumed that program files may be restricted to execute only on particular sites and that assigning a data file, say Y1, is independent from the assignment of data files Y2 ... Yn.

It can be seen in the following example that different allocations of a program file and a data file can lead to different cost results. A query from site A may need to be evaluated by a program file residing at site B. This program in turn may need to access a data file residing at site C or possibly even site A. It can easily be seen that when this is compared to the situation where a query, program file and data file all occur on a single site there is a difference in costs involved in, at the very least, communications. A very extensive cost function, taking numerous pages of text to explain and incorporating program query costs, program updating costs, data file update costs as well as program and data file storage costs has been

developed by Morgan [33]. With the possible exception of the inclusion of file storage costs, the price of file storage has dropped dramatically since this work was done [41] the cost function presented still appears valid. This cost function does allow optimal file allocation and the inclusion of program/data file dependence in the allocation algorithm must be considered an advance over the first solution presented. However, the lack of attention to queuing delays in this solution may be considered a shortcoming due to the fact that data files are allocated independently of each other regardless of which program uses them.

2.3 Heuristics to Minimize Communications

While the previous solutions to the FAP have attempted to minimize costs in the areas of communications, both query and update, as well as storage costs, the following solutions attempt to maximize performance by minimizing one or more of the following: access time, execution time, data transfer, queuing delays, and user response time.

A solution developed by Mahmoud [30] which concentrates on replicating files is described below. The solution objective is to minimize communications costs while taking into account network system delays. Two of the more major ideas in Mahmoud's approach are, (i) only feasible possibilities will be explored and (ii) the heuristic solution found to have the lowest cost is assumed to be close to the global optimal solution.

The proposed method has two components, a start routine and an optimising routine. The start routine finds possible initial file allocations, subject to the system

delay and availability constraints. With a number of possible starting solutions now created, the optimising routine is then used. Each initial solution is presented to the optimising routine which in turn tries to improve the cost by adding or removing one copy of the file. If a solution is found that has a lower cost than the starting solution, this new solution now becomes the new starting point. This process repeats itself for all feasible solutions until no further cost reductions are possible by movement of a file. By comparing the final cost relating to each of the starting solutions the smallest cost is assumed to be one closest to the optimal solution. Examples have been tested that support the conclusion but differences of at least 10 percent between the best heuristic solution and an optimal solution have also been found.

This heuristic method does appear to allow a reasonable solution to large allocation problems, but does have some shortcomings. No consideration has been given to program file and data file relationships and it also ignores system storage constraints. If no storage constraints exist and few updates occur, all sites could be assigned all files and in this way minimize costs.

2.4 Heuristics to Minimize Access Time

A solution that attempts to minimize average access time has been developed by Ramamoorthy [34]. An optimal file assignment is achieved but a number of assumptions are used that are quite possibly too restrictive. Among these are: file activity is known beforehand, only one site makes file requests, query and update costs to a particular file by one user are always identical. Even though this solution may have overly restrictive assumptions, it does bring into play the fact that a memory

hierarchy exists and is very important in the FAP.

This initial restrictive solution has been expanded upon by Chen [12] to a more complete solution. A more complete cost function is presented in an effort to minimize the total cost of the system for any sort of network topology. The costs considered are actual computer equipment cost, software costs, storage costs of both database and programs, communication links and transaction processing costs. The idea of incorporating a comprehensive cost function to include both initial design costs as well as actual production or operating costs is certainly what is required. The solution does however, overlook an important aspect of most databases. It is stated that the solution "avoids some of the disadvantages of heuristic and decomposition algorithms". This is true but only with restrictive assumptions and a minimal set of sites and databases. The problem has already been shown to be intractable and so the use of heuristics can not be avoided if a solution is to be found in a reasonable time for a large number of sites and a large number of files.

With the problem of file allocation being NP-Complete and an exhaustive search mechanism being unfeasible it would appear that some sort of heuristic solution must be employed to attempt to optimize or at least tend toward an optimal solution.

Chapter 3

Implemented Solutions

An outline will be presented for both the logical and physical architectural requirements of a DDB system. Two systems will be looked at to see how specific problems were solved as well as areas where improvements might be made.

3.1 System R/Star

As with other distributed systems, R/Star is really centralized database technology that has been improved upon in order to make a distributed version. The foundations for R/Star were laid in 1976 with a system called "System R". System R was one of the first relational database management systems and was the first to implement the relational query language SQL, called SEQUEL at the time. It is this centralized system that is used in today's DB2 and SQL/DS.

The R/Star system was designed to allow sharing of data among separate data bases on separate machines. Three design principles which guided the project were [4] [13] [11]:

1. Site Autonomy.

2. A single site view. (location transparency)

3. Support of Replicated and Partitioned tables.

Site autonomy requires that a site must be able to fully control its own data as well as being able to control access of other sites to its data. This should occur even in the event of communications failure. A single site view is required so that from the users point of view the system appears to be the same as a centralized version eg. System R. The final major objective was that of replicated and partitioned tables. This objective is closely linked to good system performance. Fragmentation and Replication are not implemented on current versions but views can simulate fragmentation.

3.1.1 System R/Star Architecture

The R/Star system consists of three major components [13]:

1. A Transaction Manager. (TM)
2. A Data Communication unit.
3. A Database Management system.

The Database Management system in turn may be looked as as consisting of two major components:

1. A storage system which is concerned with the actual storage and retrieval of data.
2. A database language processor which translates programs written in SQL into operations provided by the storage system [11].

These interaction of these components may be seen in figure 3.1.

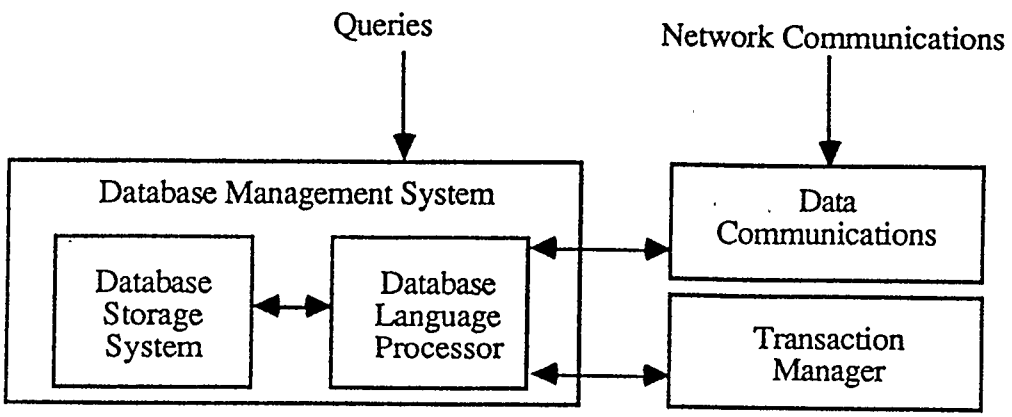

Figure 3.1: R/Star component interaction

Transaction Manager

R/Star may be accessed through the Relational Data System (RDS) in two ways. SQL may be used interactively or SQL may be embedded within other programming languages. A transaction may be defined as one or more SQL statements enveloped by a start-trans and end-trans statement. The Transaction Manager (TM) at the invoking site assigns a unique identifier to the transaction consisting of a unique local identifier combined with a site identifier. A database application program can only get access to the database through the R/Star system and in particular, through the TM. The site at which the transaction is invoked is defined as the Master site. The master site has the responsibility of determining the global optimization of the transaction. In the R/Star system it was decided early on that compilation of transactions or queries was desirable from a performance standpoint, since the advantages of compiled code outweighed the cost of doing the compilation. It is during this compilation process that the master site determines the actual execution

plan. The database objects referenced in the query are resolved by systems catalog references, user authorization is verified and an access plan derived. It is up to the master site to decide which parts of the global access plan will be distributed to other sites called apprentice sites. The master site now proceeds to compile that part of the access plan that pertains to its site, while the apprentice sites repeat the planning process at their own sites until complete compilation is achieved. In the event of a similar transaction in the future it can quickly be resolved, since all objects and access paths have already been determined and compiled code already exists, there is no need to repeat these operations again. The compiled code residing at a site is called an access module. If upon a subsequent query it is discovered that some object or dependency has been changed the entire module is destroyed and a new one created incorporating the new values. This occurs for this one site only, and not for all sites since their access modules are still up to date, thus in essence, only a partial recompilation need take place.

Systems catalogs are stored in the system as ordinary tables. There are two types of names allowed with R/Star, a print-name and a system-wide-name. A print-name is a name normally used by a user to refer to an object. The system-wide-name is a name used to give every object a unique identifier. It consists of a concatenation of four values in the following design:

- $< CREATOR - NAME > @ < CREATOR - SITED > . < OBJECT - NAME > @ < OBJECT - BIRTH - SITE >$

$< CREATOR - NAME >$ is a user name and defaults to the current user but is unique for that site. $< CREATOR - SITE >$ and $< OBJECT - BIRTH -$

$SITE >$ are unique system wide names whose uniqueness is guaranteed at system installation time, these default to the current site. An $< OBJECT - NAME >$ or simple print-name is resolved to a full system wide name using synonym tables and name completion defaults [13].

Each site maintains catalog entries for each object that was created there. As well, a table is maintained for all objects for which this is the current site; this is required since objects may move between sites. Regardless of where an object resides, the object's birth-site table contains enough information to allow full data retrieval from another storage site. If a storage site of an object is different than the birth site, entries in the birth site table show where the storage site is. In this way at most two accesses are ever required to retrieve any object.

Data Communication Unit

Communications between R/Star sites is provided by IBM's Customer Information Control System (CICS) designed to handle I/O and communications. It is the Inter System Communications (ISC) component of the CICS that does the actual communications. Both the applications program and R/Star run within the CICS, but it is the R/Star system at different sites that communicate, with all queries being processed by an R/Star system.

When an initiating site makes a request at another site for data, this second site starts up a process to handle this request. This child process remains linked with the parent process until the request is completed. Identification and authorization is only done once, at process creation time, once completed, a session of communication lasts until one of a number of terminating conditions. Upon successful completion of

the request the process sends the data to the requester and the process terminates. In the event of communication failure the use of primitives such as acknowledge or timeouts will notify the child process that a problem has occurred and to terminate.

Database Management System

The actual storage and retrieval of data is done using a storage system known as RSS*, which, in turn, is based on the Database Management subsystem known as RSS, an acronym for Relational Storage System developed for and used with System R. In System R data is stored in a logical address space called a segment. All the tuples of a relation must be assigned within one segment as set out in the Relational Database System (RDS) and as set out by the user during initialization of the system. Different types of segments exist however, each with its own set of functions. It is the function of the RSS to map these logical segments onto the physical disk space available. A physical page size is determined at system design time and in this way, each segment depending on its size, consists of a number of pages.

Bit maps exist that are directly associated with physical page areas on disk. By setting the bit map, areas can be allocated to segments of any size and created or destroyed dynamically. It is the allocation of disk space through these bit maps that allows clustering of related data physically on disk.

When a relation is created, each tuple within that relation is stored with a prefix. This prefix consists of a tuple identifier (TID), the number of fields required for the data, and the number of fields required for pointers. It is at creation time that each tuple within the relation is assigned a TID by the RSS. It is this TID that is used by the RSS when referencing a tuple. These TID's are also used in linked lists during

requests such as "next" and "previous". The TID itself is actually a combination of a page number within the segment as well as the byte offset from the bottom of the page. The byte offset refers to a fixed location reserved at the bottom of each page which contains the actual byte location of the tuple on the page. By allowing more than one relation to occupy a single data segment page, tuples from another relation can be stored on the same page if required. This clustering of tuples which have some matching data fields can lead to dramatic improvements in access times due to reduced page accesses. It is the Relational Data Interface that supplies the RSS with clustering hints. These hints are in the form of possible TID's which are to be used by the RSS if possible. If the tuple in question cannot be put on the page requested, it is then assigned to a nearby page.

In order to facilitate efficient access and logical ordering of tuples of one or more relations, a system of links is used. These links or pointers may be used to access either previous or next tuples dependent on some field, or to designate either a parent or a child relationship. It is these pointer values that are kept in the prefixes attached to every tuple and are essentially in the form of linked list.

In order to access tuples the RSS maintains a hierarchial structure of indexes in the form of B trees. The B trees take a balanced structure with the leaf nodes consisting of a concatenation of the key value as well as the TID. The leaf nodes are linked in the form of a linked list so as to facilitate sequential access between leaves and pages if need be. The actual data value is accessed through the leaf node value and not directly.

As was stated earlier in this section R/Star is in many ways System R with communications facilities. Due to this foundation it may be that certain decisions

that were made in System R are not as correct for R/Star as they might have been. One of the choices that may have been correct for System R but a poor choice for R/Star is that of file allocation. In System R the choice of file allocation may have been obvious as there was but one site and the need to minimize I/O operations was given a high priority [4]. In R/Star the need to minimize I/O operations is still important but of more concern is the need to minimize communications. More importantly from a file allocation aspect is the fact that in R/Star tables must be manually allocated by the Database Administrator (DBA) to specific sites [15] based on statistics gathered during usage and experience. There appears to be no automated or dynamic file re-allocation available at this time.

3.2 Distributed Ingres

Distributed Ingres (Interactive Graphics and Retrieval System) has its roots in the single version of Ingres. In the same way that R/Star is in many ways multiple versions of System R, with communications between them, Distributed Ingres is really multiple copies of Ingres with communications between them. When an application program invokes Ingres at a site, it becomes the "master" site and any other site from which data may be required will become "slave" sites. It is the master site that makes all decisions in resolving queries. Designed to run within the Unix operating system, it is Unix that provides the actual physical file handling for Distributed Ingres. All communications aspects are also done through the Unix system.

One of the main criteria that the designers of Distributed Ingres wanted to follow was that of a large degree of parallelism, so as to improve overall system performance. This parallelism was to be achieved through the use of multiple machines and a fast communications network.

3.2.1 Ingres System Architecture

The Distributed Ingres system consists of really two major components:

1. Ingres
2. The Unix Operating System

The Ingres component may be broken farther into two subcomponents:

1. Access procedures
2. Storage structures

The Unix Operating System component will be looked at in light of:

1. File System

2. Processes and Communications

Ingres

There are two ways in which a user can communicate with Ingres, either through use of the interactive query language known to the system, called QUEL (QUEry Language) or through use of EQUEL (Embedded QUEL). QUEL statements may be looked at as requests to Ingres to perform certain functions while EQUEL is really more complex due to the use of a precompiler designed to convert the EQUEL code into a C program with appropriate calls to Ingres. It follows from this that Ingres can really be invoked in two very different ways, either interactively through the use of a database name or through the execution of a program in C that was created through using the precompiler.

The Ingres database is essentially made up of four types of files:

1. Administration Catalog Files

2. System Catalog Files

3. Database Administration Relation Files

4. User Relation Files

The Administration file contains only the user ID of the Database Administrator and information required for initialization of the system. A System Catalog file

exists for every database created and consists of predefined names for each relation. As with every relation they must be "owned" by someone, in this case the Database Administrator (DBA). Only Ingres itself or the superuser may perform direct updates on these relations, although anyone may do reads on them. The system catalog is really made up of six catalogs:

1. Relations Catalog
2. Attribute Catalog.
3. Index Catalog
4. Protection and Integrity Catalog
5. View Catalog
6. Graphics Catalog

It is the Relations Catalog and Attributes Catalog that have information about the ownership, names, type and size, and related information required to determine structure and contents of each tuple.

The Index catalog is used if secondary indexes exist for a relation. The Protection and Integrity catalog is used for exactly that purpose, to indicate access and integrity constraints. The View catalog provides the information required to provide particular views of a relation consistent with a particular request. The Graphics catalog contains information related to graphics processing.

The actual storage and retrieval of data from a relation is done using a system called Access Method Interface (AMI). AMI is composed of a set of very UNIX like calls. For example the call sequence to open a relation for access has the form of:

- OPENR(descriptor, mode, relation-name)

These calls are the interface between Ingres and the underlying Unix operating system which does the actual storage and retrieval operations. Unlike System R/Star, the designers of Ingres decided against building a file handling system as part of Ingres and decided instead to leave these functions up to Unix.

There are five methods of storage in Ingres. These are hashed, compressed hashed, ISAM, and compressed ISAM. When a tuple is stored using one of these methods its position in the relation is determined by the value of the domains used as keys.

Every relation is stored in a separate file with each file consisting of one or more pages. The first is a primary page consisting of five major areas. If this primary page is of insufficient size to hold all the data that needs to be stored, more primary pages are created for farther storage. If the relation grows due to additions and more storage pages are required, overflow pages are created and linked to the primary page with which they are associated through the use of pointers. One key restriction is put on all tuples, this is that a tuple must reside entirely on a single page. This however does not appear to be a major impediment in most cases. In the event that the ISAM storage method is used, the directory for accessing the primary and overflow pages is kept on separate pages following the primary or overflow pages.

As noted earlier every relation is stored as a separate file. This is not to imply however that a relation may not exist as more than one file, for this is exactly what is allowed. Horizontal fragmentation of a relation is supported with the *Distribute* command which creates a new relation containing the name of the fragmented relation

and the site on which that fragment resides. Replication of relations is also supported. With replication one of the copies is designated as a primary copy. These conditions apply to normal data relations as well as the systems catalogs. It can easily be seen that this ability to store systems catalogs at another site can lead to tremendous advantages in transaction speed if remote data is required, although this overhead does complicate updates and concurrency.

When a table is created by a user it is the user who becomes the administrator of that table and makes the decisions related to fragmentation and replication. The user's decision to replicate a relation at another site does not need to take into account costs related to storage capacity, communications or queries and updates. The only restriction that may be placed on users in an effort to restrict or prevent this haphazard replication and distribution of files, is to remove their ability to issue the related QUEL commands. This may be done by the system DBA. Possibly due to the user's ability to either replicate relations or remove relations there appears to be no reason for the system to do this. Replication of relations by the system would certainly be advantageous however, for if it was discovered that a great number of queries were generated at a site that did not have a particular relation, this relation could then automatically be replicated at this particular site without a user having to do so manually. As was the case with R/Star, Distributed Ingres has the functions required to gather statistics on relation usage but a facility to dynamically move and re-allocation relations does not exist.

The Unix File System

When Ingres was initially designed it was decided by the design group that Ingres should appear to Unix as just another process and not to make any changes to Unix at all. Due to this it is up to Unix to do all the actual file storage and retrievals as need be.

When a new file is created, the Unix system assigns to it a unit called a disk inode. The inode contains the following information, owner, group, file type, permission settings for read, write and execute, last accessed, last modified, last written, size in bytes and disk addresses. The areas of file type and disk addresses will be addressed here as they are of special interest. File types are regular, directory and special file. A normal data file would be named regular. When a request is processed by the kernel, for the opening of a file, the file path name is actually converted to an inode which then allows access to the file. A number of disk accesses may be required at this point. First to retrieve the disk inode from disk and move it to main memory buffers. Second to access from this inode the disk address of the blocks required for the actual data. These addresses are then used to do further disk accesses of the actual data block which are then also moved into system buffers. It should be noted that the original request produced by the kernel came from Ingres and before an actual search can be done for the specific data, one more move is required. The data in the system buffers must now be moved to the Ingres memory area or Ingres cache. In an effort to reduce the number of disk accesses and so improve file access times, the latest version of Unix 4.2 BSD has allowed blocks to increase from 512 byte to 1024 byte pages. Support for commercial database systems is provided through the System V variant of Unix which has file and record locking facilities that were not

available on earlier Unix version. File locking is the ability to give exclusive access to a particular process for Reading and Writing until this process is finished with this file, while record locking allows this for a subcomponent of the file, in the case of Ingres, a tuple.

Unix Processes and Communications

A process may be considered as a number of instructions designed for execution. Processes may be created or destroyed as required. A process is created through use of a Unix instruction called a "fork". A fork operation splits the original program into two copies. The only difference between the parent and the child process is their actual process identification number (PID). When Ingres is invoked through Unix it causes four processes to be created in a form as shown in figure 3.2 [39].

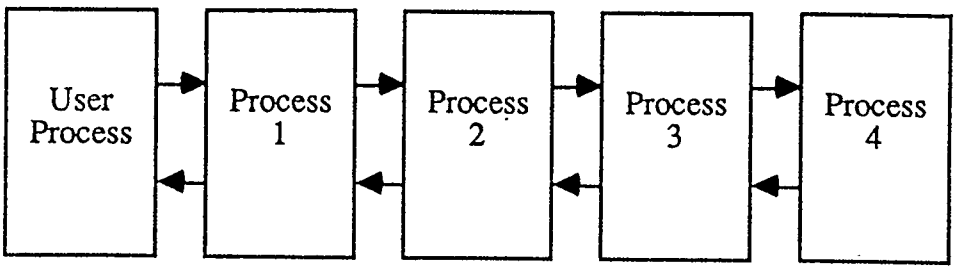

Figure 3.2: Ingres process structure

Process one is a terminal monitor allowing user interaction. If an EQUEL program had been executed to invoke Ingres, then process one is a C program. It should be noted that use of the EQUEL precompiler over interactive usage of Ingres does not result in an increase in performance. All queries are interpreted, one interaction at a time regardless of whether they occur once or a number of times. Process

two contains a lexical analyzer, parser, integrity control routines and query modification routines. Process three has query decomposition routines. Process four has utility routines required to activate Create, Index, etc, as well as a deferred update processor.

The reason that process two, three and four are created instead of just one larger process is due to hardware constraints as well as the possibility of generating too many page faults. It should be remembered that at the time Ingres was implemented, the hardware was a PDP 11/40. There are, however, costs involved in having three processes instead of just one. If one process is used, it could incorporate subroutine calls instead of the "piping" system required by Unix for communicating between processes. In the four process example it takes a minimum of eight such piping calls to completely execute even one Ingres command [39]. The option of one process instead of three separate ones is a real possibility today. The idea of using subroutines instead of communications is unrealistic for separate machines regardless of how close they might be, especially in a truly distributed system.

As distributed Ingres is essentially a number of single site Ingres implementations supplemented with communication, it would seem a relatively straightforward implementation. This has not been the case. The only hardware and software available to the designers in 1978 was a communication system called Berknet, essentially a multiplexer for RS232 communications lines. It can be seen that the undertaking was no trivial task in that it was not completed until early 1983 [39] . The need to provide a separate communication system outside of standard Unix would appear to circumvent the original aim of having Ingres appear as just another process to Unix. The Unix shortcoming of not providing intersite or intermachine communi-

cations had to be overcome, and distributed Ingres did this by providing its own communications system. This lack of intermachine communication has been alleviated on later versions of Unix. With the release of Unix 4.0 BSD there has been the implementation of a new system facility called a "socket". Sockets allow for both communications between processes residing on a single machine as well as processes residing on separate machines. When a socket connection is created within the UNIX domain it may be used for interprocess communication within a single site. This is very much like the "piping" function that existed previously. In actuality pipes are now implemented as the connection of two sockets. There is a major advantage to the use of sockets over the original piping system within a single site. This is due to the fact that pipes are uni-directional while sockets are bi-directional. When a socket connection is created using the INTERNET domain it is designed to be used for interprocess communication between separate sites. This internal communication facility of sockets is essential for truly distributed Ingres while at the same time maintaining standard Unix.

Chapter 4

Dynamic File Allocation Model

4.1 Introduction

User response time in a distributed database is among other things highly dependent upon the distribution of program and data files within the system. In many systems the decision as to where to place files is made by the Database Administrator. This is often done in a static manner where files are assigned to a site and then left there. Often the only time that any attempt is made to reconsider these decisions is when one of two things occur: where someone wants to use a file and is unable to access it due to some sort of restriction or when a particular file or set of files is accessed so frequently that the site or system throughput starts to suffer dramatically. In an effort to prevent access changes from reaching a point where it causes the system to degrade to an unacceptable level the area of dynamic file allocation holds a great deal of promise.

Access frequency is often measured or estimated in an attempt to determine where best to allocate files. These accesses are then often combined with known values such as the number and speed of devices such as disk drives as well as processing power of a particular site in an effort to determine the best allocation of files. This must of course be combined with some sort of balancing of files among sites, as it is obvious that when too many files are stored on a fast device or more powerful site, the waiting time to finish processing might turn out to be longer than if a file were stored on a

slower device or less powerful site.

Fundamental to all of the above is the need for some underlying trend in the way files are accessed. Access frequencies, queuing delays and system throughput values have been known for sometime as basic indicators of system performance. It is access frequency however that primarily determines the rest and is fundamental in determining file allocation. Changes in file accesses would also indicate that a different allocation of files may be required. It is this reallocation of files to improve efficiency that should be given primary consideration. This brings out a number of questions. For example how much change must occur in the way files are accessed in order to allow successful reallocation of these files? Is it possible to reallocate files by determining an underlying access trend from a number of small sets of access frequencies, each of which in themselves would be unusable? The following sections will attempt to answer these questions and their relationship to dynamic file allocation.

4.2 Research Objective

In an effort to improve system efficiency when changes in access frequency are encountered, the area of dynamic file allocation will be explored. This will be done through the use of a heuristic algorithm and the simulation of events. It should be noted however that this is not a pure simulation of events. The area of communications between nodes of the system has actually been implemented within the simulation code. The reason for doing this is that there was no need to simulate communications when it was actually possible to implement it. It is also possible in this way to make the step from simulation to prototype much easier. Unlike the

area of actual data files and user requests which had to be simulated since "real" values were not available, actual communications and their related values could be obtained.

Extensive monitoring of a real system's behavior may be impractical or impossible due to cost or time constraints, yet often small sets of data related to access frequencies are obtainable. A lower level of system monitoring associated with the smaller access frequency sets should result in less system overhead and so contribute to savings in time and cost.

It will be shown that by using a number of small access frequency sets (history) combined, file allocation can be determined dynamically. Factors such as throughput and queuing delays will be used in an effort to show that this reallocation of files is possible. It is suggested that significant improvements will result in query response time and system throughput when files are allocated based on the above mentioned factors.

The remaining sections of this chapter may be summarized as follows:

- Section 4.3 — A description of hardware used for the simulation.
- Section 4.4 — Contains information on file structure, communication between sites, as well as data collection and reduction.
- Section 4.5 — Describes the algorithm used in determining file reallocation.
- Section 4.6 — Simulation Verification.
- Section 4.7 — Simulation Results.
- Section 4.8 — A discussion of conclusions reached.

- Section 4.9 — Future Research

4.3 Environment

The hardware on which the simulation was developed and run is as follows. It consists of four Sun 4s as nodes as well as two Sun 3s as file servers communicating on a 10 Megabyte per second Ethernet spine as shown in figure 4.1. The Operating System is Sun OS Release 4.0E. The figure of 10 Megabyte per second on the Ethernet is a theoretical figure, for the actual transmission rates are known to be substantially less on Ethernet systems, which are CSMA/CD (carrier sense, multiple access, collision detection) systems.

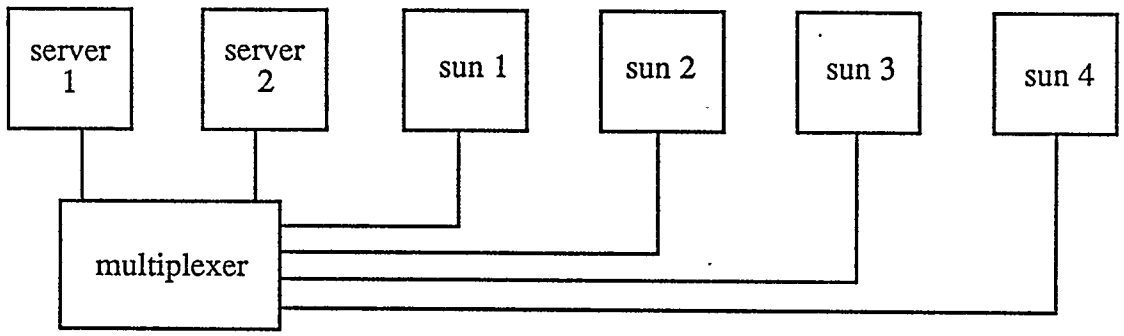

Figure 4.1: System Configuration

4.4 Model Description

A system that is to include dynamic file allocation would be very complex and the use of analytical evaluation seemed impractical since the problem has been found to

be intractable [19] [23]. It is for this reason that a simulation was done. In most simulation models an effort is made to emulate the system being modeled as closely as possible. The difficulty in this is that in an effort to completely or closely mirror the characteristics of a system, the simulation model may become too complex. A small number of sites and files will be used in an effort to limit complexity and yet retain enough of the problem to make the results useful.

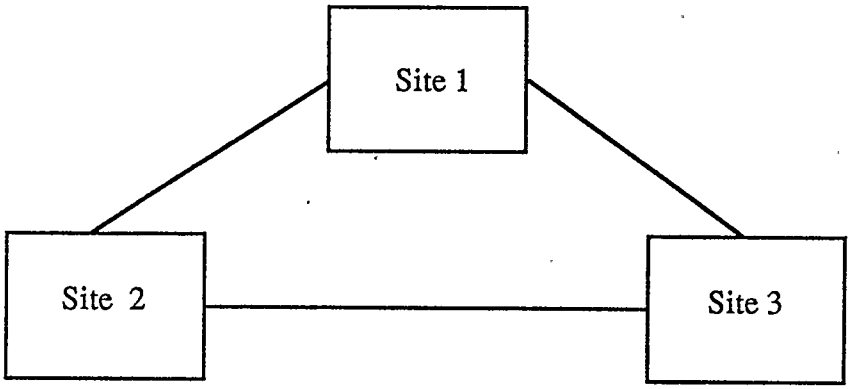

Figure 4.2: Model Network

The chosen model topology is that of an arbitrary number of sites in a completely connected network as shown in figure 4.2. Each site in the system is completely independent, containing its own processor, communications system, database and query file representing a set of user queries. In an attempt at minimizing the amount of communication traffic required, when remote requests are serviced, a completely connected network was chosen. Each site in the network contains software designed to maintain a table indicating locations of files that are either currently residing at this site or were created at this site, thereby keeping the maximum number of sites that need to be contacted in order to access any file at 2. In Local Area Networks it

is quite common to have completely connected networks due to the close proximity of most nodes and increased system efficiency even though there may be added cost and complexity in both hardware and software.

In an effort to prevent the simulation from becoming more complex than is necessary, certain simplifying assumptions are made. It is assumed that a request requires access to only one data file. Although in an actual system requests or queries needing access to more than one data file are not unusual, such requests need not be an essential part of the simulation. There are really two reasons for this decision. The first is that queries that require single data files are quite realistic and in certain types of databases make up the majority of requests. An example of this might be interactive queries concerning a bank balance at a local branch or an auto parts dealer request on the availability of a specific part. There are also certain multi-file queries that can be treated as a sequence of single file queries. An example of this may be seen in the following SQL like statement:

select employee_number from employees where city is in

(select city from offices where office = 'head_office')

The second reason is not quite as obvious but is still equally valid. It would seem reasonable to start out with an elementary situation and build to the more complex. If it can be shown that file reallocation is worthwhile for simple file queries then it would certainly be justifiable to continue the research for the more complex case of queries that require more than one file, but it is best to prove the elementary case first.

It is assumed that a query can be completely processed at one site and needs only to return a result that is smaller than the actual data file accessed. This is

based on allowing only one file to be used, for if two or more files are allowed then it is very easily possible to achieve results that are larger than simply the sum of the data files.

Taking into account the above simplifying assumptions, the events in a Distributed Database would likely take the following form. The user enters a query. The applications program or system then evaluates the query and determines what data are needed to derive the result. If the required data reside on the host machine's database, the data required are retrieved, the result determined and returned to the user and the next query is then processed. When it is determined that the data required do not reside on the host machine, it must then be decided where the required data exist and the query is sent there. Update queries are quite straight forward in this case but when retrieval queries are considered two major possibilities exist. The first possibility is that of retrieving the data from the remote site, transferring it to the querying site from the remote site and processing it at the querying site. Once processing is completed and the result of the query displayed, the next query may now be submitted at this site. The second possibility is that of processing the query at the remote site and sending back only the required answer to the querying site. The introduction of parallel processing into the structure seems quite appropriate considering the increasing use of networks. It would seem reasonable to assume that if a querying site has a relatively high percentage of processor usage compared to a remote site, then, if possible, it would be better to process the query on the remote site rather than move large amounts of data to the querying site where processing is already higher. The introduction of large data transfers in the communications system will certainly tend to slow down the communications system. In this second

possibility, when processing is complete and a result returned to the querying site via the communications system, the next query may now be processed at this site. *In this simulation model processing will be done at the site where the data file resides.*

In the simulation the above sequences take the following form. For each available site, a series of values is generated representing requests for data files. These values are stored in a file referred to as the query file. As will be shown later these values may be adjusted to simulate a wide range of requests, from totally random requests to the other extreme of values generated to simulate requests for only one data file repeatedly. A file designed to simulate actual data files is also created for each site. This database file contains values designed to represent data files of differing sizes as well as an index table to be used for determining on which site a particular file now resides. Section 4.4.1 will give a more in-depth explanation as to how these data values are generated as well as how the index table is used.

The first value is taken from the query file and evaluated for processing. If the requested data exists on this site, execute the query. If the data does not reside on this site then send the query to the remote site where the data file resides and evaluate the next query. The remote site will upon completion of processing immediately return the result of a query to the site where the query originated. It is therefore possible for a querying site to process a number of quick requests before the result of a more time consuming remote request returns. The sequence of events may be viewed as shown in figure 4.3.

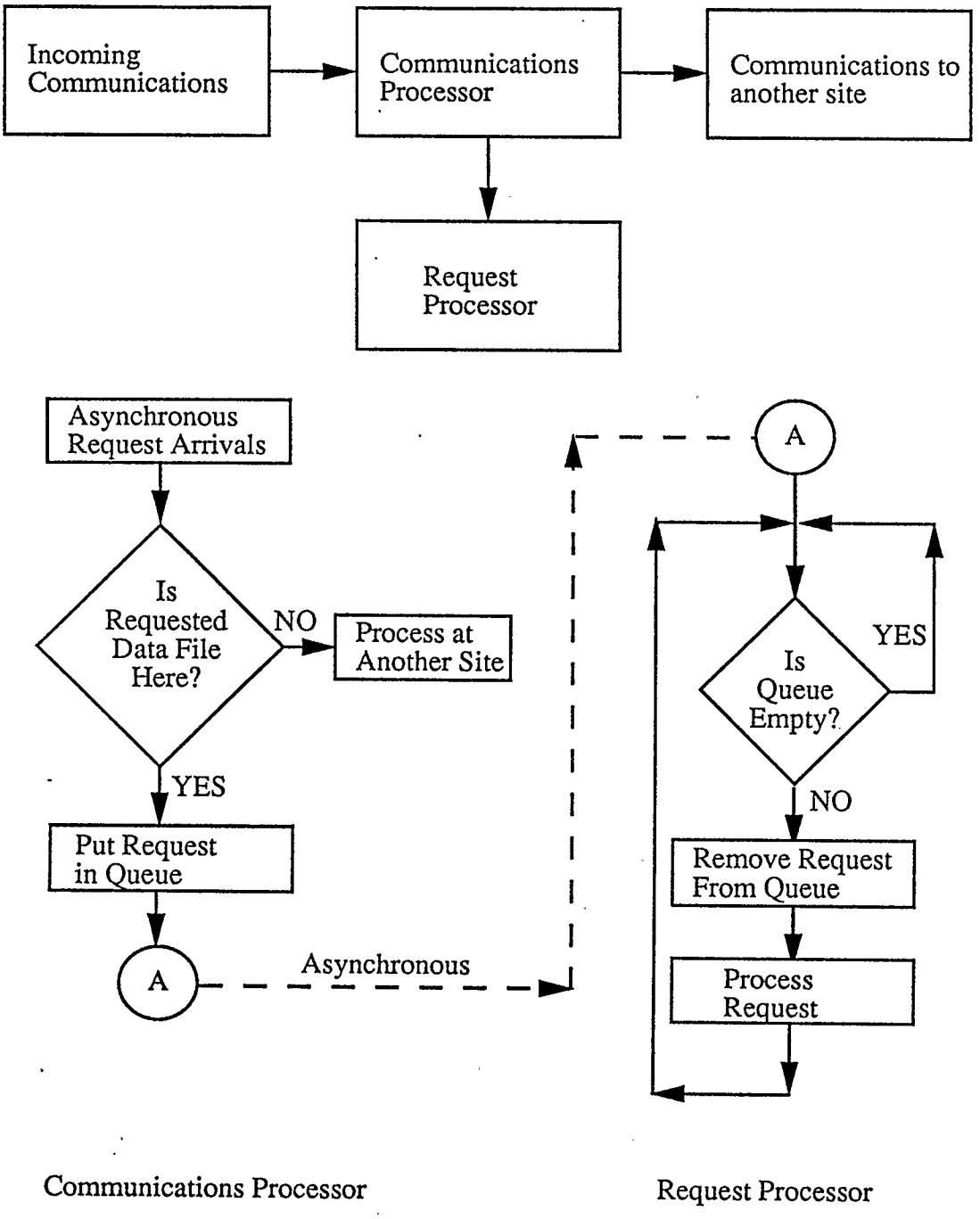

Figure 4.3: Single Node Interaction

The idea of a host site with a query file, a database and a means of communicating with remote sites which also have databases and processing power can now be taken a step further. Each of the remote sites itself also has its own query file that will be executing concurrently. This is shown in figure 4.4.

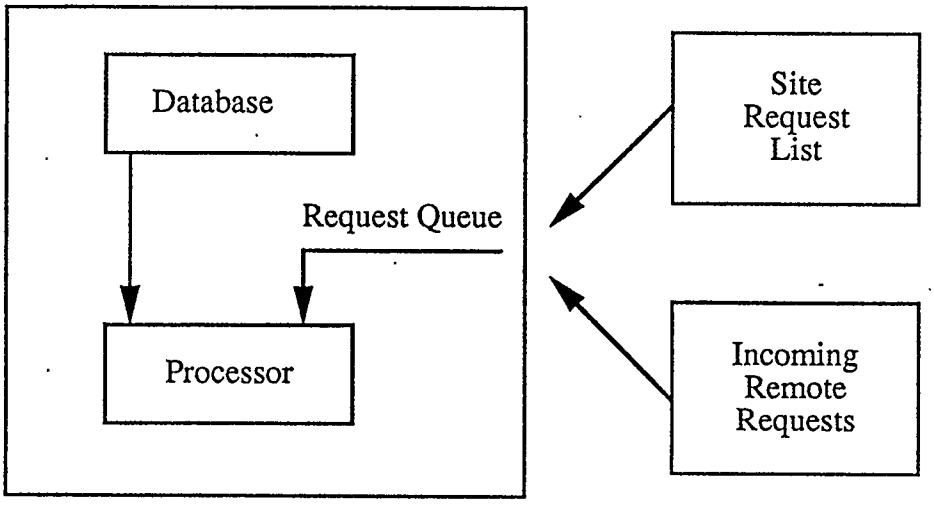

Figure 4.4: Multiple Node Interaction

The creation and running of a site is best described as follows:

1. A main process is created and remains active until all queries from the query file for this site have been completed.

2. The main process on its initial creation, itself creates a child process which is designed to handle all queries for files at this site, whether from this site or queries from other sites for data files on this site.

The basic process structure for a site is shown in figure 4.5.

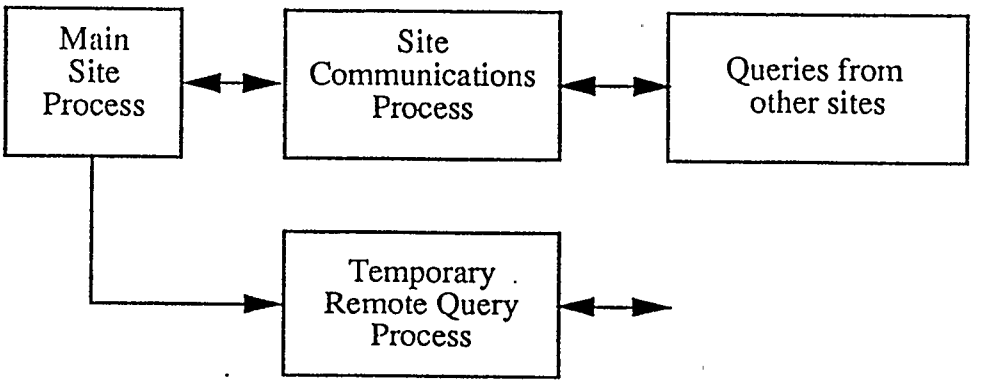

Figure 4.5: Individual Site Process Structure

The actual sequence of events for an arbitrary number of sites is as follows. After the initial creation of the main process and remote communications process, each site takes the first entry in its query file and evaluates it. A table used to determine file locations is maintained at each site. Section 4.4.1 is used to provide a full explanation of how this table is used and maintained. If the file does not reside at the requesting site this request is sent to the site where the requested data file was initially created, and an immediate check is done to see if the requested data file still exists at this site or has been transferred to a different site. In the event that the data file has been moved the query is immediately routed to the site where the data file now resides. At most two sites must be visited for any query, since the site where a data file was initially created can always supply the name of the site where the file now resides. This is much like the R/Star system where at most two sites need to be visited to answer a request for a file. Arguments can be made that all sites could maintain a record of the location of all files and this may be quite valid in a situation where there are few file re-allocations. In systems where there is a great deal of file movement, the duplication of each file's location in every site's table will slow down the system

because all tables must now be updated every time a file is moved.. When considering these points, in addition to trying to maintain site autonomy it was decided to have each site only retain information directly related to it.

Upon determining that the data file resides on this site, the query is entered into the queries pending queue along with the value representing the requested data file's service time. A complete explanation of a data file's service time is given in the following section.

Query entry into the queue follows a Poisson distribution. It would seem reasonable to treat all requests the same and yet have each totally independent of each other. Arrivals that are randomly distributed and independent of each other appear to closely mimic the actual operations of independent sites in a network [27]. Upon entering the queue of queries pending an immediate calculation is done to determine the amount of time this query must wait in this queue. This is done by using the data file service times that were entered in the queries pending queue with every query when it entered the queue. By adding up the service times of queries preceding this query in the queries pending queue the total wait time for the latest entry is calculated. If the site is not busy the queries pending queue is checked for available queries and the leading query is processed following a First-Come-First-Served principle. Any queries arriving either from this site or other sites must then wait in the queries pending queue until the processor is again available. Upon completion of a query, statistics are collected concerning that query. The process of evaluating queries, both local and remote, continues until all queries have been evaluated for each site in the system and statistics gathered for all sites. The methods used to create both data and query files as well as their actual contents are shown in the

following section.

4.4.1 Database Files

Each site has its own file designed to simulate a database. This file retains information only on simulated files that were either initially created at this site or simulated files that are currently residing here. If a file was created here but now resides at another site, the new site ID. is paired with the file number *at the creation site.* If in the future the data file is moved to yet another site, this new site ID. is entered at the file's creation site but all references at the intermediate site are removed. Sites only retain file information that is directly related to them.

Each simulated data base file therefore consists of a matrix of three columns and an appropriate number of rows to represent every file currently residing at this site as well as the location of every file initially created at this site. The first column has the site number corresponding to where the file listed in column two now resides. The values in column three are based on the following assumption. In an actual system, queries to certain types of files, accessed through indexes or hashing, often take very similar times. If this is then complemented by each file having similar but different data, similar response times could be expected. It is therefore not unreasonable to assume that all queries to a particular file take equally long to process. It is on this assumption that the values in column three are based. Column three contains a value representing the time required to evaluate the query needing the file listed in column two of the same row. Table 4.1 shows part of a database file and it's corresponding values.

Site Number	File Number	Processing Time
2	10	11.534991
2	11	1.686346
1	12	0.000000
2	13	1.044262
2	14	1.678562
2	15	1.186478
2	16	5.610621
2	17	10.942223
2	18	2.026231
2	27	1.362914

Table 4.1: Simulated Database File

In table 4.1, the third row down shows an example of a file that was originally created, or started out at this site, but now resides on site number 1. The processing time value related to the third row has been changed to zeros at this site to indicate no corresponding data exists here for this file. The values in column three representing query evaluation time are generated using a probability density function designed to create exponential service times. This function is derived from one presented by Deitel [16]:

$$f(t) = \lambda e^{-\lambda t}$$

and thus the probability distribution

$$y = P(T < t) = F(t) = 1 - e^{-\lambda t}$$

where T is a random variable and small t a given value of T.

Solving for t in terms of y gives

$$t = ln[(1-y)^{-1/\lambda}]$$

where $0 < y < 1$ and y is the probability of an arrival in a set time period and λ is equal to the arrival rate and where $0 < t < \infty$.

In real systems it is often the case that there are a great number of small service times and a few occasional larger service times; an example of this might be a neighborhood bank. In an attempt to simulate this behavior it was decided to use exponential service times and so achieve a Poisson distribution. This also greatly simplifies the validation process when comparing analytic and actual results.

4.4.2 Query Files

A site's query file, the file used to simulate queries for data files, takes the following form. It is a simple list of values used to simulate requests for data files either residing on this site or for data files on other sites. The arrival at the required site of each of these queries is determined by a function similar to the one used to create query evaluation times for the data files. A Poisson distribution, similar to that used in the creation of data file values, was used in an effort to simulate as closely as possible actual site arrivals. As was mentioned in the introduction, if queries were truly random in nature and had no trend of any sort, then attempting to reallocate files on different sites would be a fruitless venture. The underlying assumption is that there is some order in queries and that this can be simulated and so may be used in determining the allocation of files. There appears to be an intuitive feeling for

this as well as considerable empirical evidence to support this. In economic circles considerable research has been done to support the premise that approximately 80 percent of the work is done by 20 percent of the workers. In the computer science field similar values appear appropriate as has been proposed by [25][31]. Grosshans [25] states that "20 percent of the functionality will be used 80 percent of the time" and that it is important "to find and identify that critical 20 percent of the functionality which most users want". With these ideas as background it was decided to design the code that generates values for the query file to allow different sets of values to be generated. The design is flexible enough to allow the generation of values ranging from completely random requests for all files and sites by any site, to the other extreme of completely non-random requests, where a particular site's request file would consist of requests for only one data file. This is done by adding parameters to the generating function so that an increase in the parameter value would correspond to an increased probability of a particular site being requested. A similar parameter exists to allow an increase in probability that a particular file or group of files will be requested.

4.4.3 Network Communication

As was explained in a previous section, each site upon starting up creates a child process to act as a file server. This server has two functions. One is to act as an access device to the database and so retrieve and verify values, and the other is to act as a communications link between this site and all requests for data on this site. On the Sun OS 4.0E communication between processes are most often done using a facility called a socket. Due to the requirement for intersite communications an

Internet Domain was chosen over a *Unix Domain* which is restricted to local process communication. Each Internet Domain socket must be identified by a unique "port number". The port number in this case is the site number, given a particular site, since all site numbers are unique. Each communications server process in this way has a unique address at which it is constantly waiting for incoming messages from any sender. A sender or client need only know the server's port number in order to communicate. The client need only create its own socket connection and then using the server's port number as the destination address can in this way send its message. The server's process socket remains open until all requests from all sites have been answered. The client's socket only exists until the messages related to a specific data file request are completed. An argument can be made that this constant process creation is quite wasteful in time and expense. A full connection strategy would likely be faster than dynamic creation of socket connections which tend to take in the range of .2 to .7 seconds to complete, as well as the actual cost of process creation. Arguments can be made that to maintain a process once created has a much smaller overall cost than creating a new one. When this project was initiated, a system of machines and communications existed that would have allowed the actual implementation of the communications between machines in the system. Due to hardware and operating system changes that occurred during the project initial testing and implementation, the communications aspects became extremely difficult to do. It was for this reason that the complete simulation was actually done on one machine still using the actual communications system however. Unknown at the time of this forced decision to move from a number of machines to one machine was a technical constraint that restricted the number of active processes to 50 for

any one user. It was this final constraint that forced the decision to create processes and sockets dynamically as required and so reduce the total number of processes that might exist at any one time.

4.4.4 Data Collection

Upon start-up of the simulation and the subsequent movement from one event to the next, the following elementary information is gathered. Each site monitors all queries of data files at that site regardless of whether these queries are local or remote. With the arrival of a query, the following information as shown in Table 4.2 is gathered.

Calling_Site	File_Num	File_Time	File_Wait_Time	Queue_Len

Table 4.2: Structure of Initial Data Collected

The arriving query packet contains both the calling site number as well as the data file number required for the query. A search of the database is done to find the appropriate data file and in this way obtain the value representing the query processing time. Although the term "file time" is used in table 4.2, this value is meant to represent the amount of time required to actually process this file or determine an answer to the query presented. The value, file_wait_time, is calculated by adding up the file_times of any file queries preceding this query in the queries pending queue. This file wait time and the latest query's file time is used to determine a query's total service time. The value queue_length is simply a count of the number of queries in

the queue wanting service before this query. For example in the event of 100 queries at this site, a data file is made into which the values for each query are appended as the query is processed, in this case 100 rows would result, each in the format presented in table 4.2.

4.4.5 Data Reduction

Data reduction, although a component of the evaluation algorithm, is presented here to give continuity between data collection and the reduction of this data to a form usable for input into subsequent parts of the evaluation algorithm. The initial step in data reduction may be done at each individual site and so has each site creating a data file of related facts. Processing may also be done at one designated site to which all previously collected data files are first moved before this initial reduction takes place although it is immaterial which site is chosen. It is this designated site scenario that will be followed.

The data reduction step may be initiated in one of two ways, either at reaching a set number of queries processed or by triggering a designated amount of change in system performance. At this point it may be said that data collection has been completed and that data reduction can begin. A new file is created into which is entered a unique row associated with each file/site combination; this is shown in table 4.3.

File #	Calling_Site	File_Time	Wait_Time	Accesses	Avg_Wait	Q_Len

Table 4.3: Intermediate Level of Data Reduction

If a site made only one query for a data file then the only difference between this and the initial data collection file is that accesses and average wait time have been added. Accesses in this case would be equal to one and average wait time equal to wait time. In the event of a site querying the same file more than once, the access column would register the appropriate number of queries. The wait time column would indicate the sum of the file wait times for each request. The average wait time is the total wait time divided by the number of accesses. In this way every data file at every site has associated with it a list of associated information related to queries it serviced.

A further tallying of information is done for each site. A file is constructed containing the following information. See Table 4.4.

1	2	3	4	5	6	7	8
Site_num	T_L_Call	T_R_Call	L_Call	R_Call	Q_len	Avg_Wait	T_Wait

1	2	3	4
Avg_Loc_Res_Time	Avg_Rem_Res_Time	Avg_Site_Res_Time	Tot_Sys_Time

Table 4.4: Final Level of Data Reduction

In this file each site will have only one corresponding row which will contain its values.

1. Column 1 contains the site number.

2. Column 2 is the summation of file and wait times of all data file queries by this site querying data residing at this site.

3. Column 3 is the summation of all file and wait times for all other sites querying data files at this site.

4. Column 4 is simply the count of the number of local calls.

5. Column 5 represents the number of calls from all other sites.

6. Column 6 average queue length encountered by all queries to this site.

7. Column 7 is average_wait_time, calculated by

$$\frac{\text{total\_wait\_time}}{\text{total\_local\_calls} + \text{total\_remote\_calls}}$$

8. Column 8 is the summation of all queries wait_time. Wait time is time a query is required to wait before this particular query is serviced.

Having tallied the values for each site and creating a single row corresponding to each site, some of these aggregates are now used to determine four different total and average values for the complete network. These additional four are shown below and are those indicated in the second row of information in table 4.4

1. Value 1 is average_local_response_time. This is calculated by

$$\frac{\text{local\_call\_file\_time}}{\text{total\_local\_calls}}$$

2. Value 2 is average_remote_response_time. This is calculated

$$\frac{\text{remote\_call\_file\_time}}{\text{total\_remote\_calls}}$$

3. Value 3 is average_site_response_time. This is determined by

$$\frac{\text{local\_call\_file\_time} + \text{remote\_call\_file\_time}}{\text{total\_local\_calls} + \text{total\_remote\_calls}}$$

4. Value 4 is total_system_time. Determined by

$$\text{total\_local\_time} + \text{total\_remote\_time}$$

So as a result of running the simulation, three different files are created as shown in tables 4.2, 4.3 and 4.4. It is the values in tables 4.3 and 4.4 that will be used in the input portion of the evaluation program leading up to possible file reallocation. In subsequent runs of the simulation new files are created; previous ones are not discarded but are kept so that they may be used as a history of what has occurred before. This comparison of present to multiple levels of previous events is vital in the actual evaluation section of the algorithm.

4.5 The Evaluation Algorithm

As was mentioned previously, the evaluation algorithm takes as initial inputs the files generated in the last stage of the data reduction section as were shown in tables 4.3 and 4.4.

A maximum of three files representing the latest as well as previous data reductions were kept, with the latest referred to as profile1, the previous as profile2 and the one before the previous as profile3. It is these profiles that are a major force in determining whether a file should be moved or remain were it is. In the case where this is the first evaluation to occur, the latest profile is copied into profile2 and profile3. A comparison of profile1 to profile2 and profile3 is used first of all to determine if progress is being made with respect to overall system response time. If the latest profile shows an improvement in response times when compared to previous profiles then it is very likely that at least some previous file movements were beneficial.

On the initial input of profile values two calculations are made with respect to times. Using previous profiles, averages are calculated for previous average site response times and previous system times.

4.5.1 Determination of File Reallocation

The actual evaluation section of the simulation has the following structure.

- Input values tabulated in the data reduction section
- Input values collected in the data collection section

- Initialize parameters of: entry-point, accuracy-value, home-threshold, remote-threshold.
- Sort sites in descending order according to site average_wait_time (AWT)
- Sort data files in descending order according to number of accesses

```
While there are still sites to evaluate and the site's AWT > entry-point
        get the current site's values
        if the current site's AWT > system AWT * accuracy-value
                while there are still file/site combinations
                        get input value for next simulated data file used
                        call Single Site Requests (see section 4.5.2)
                        if data file used by more then one site
                                call Multiple Site Requests (see section 4.5.3)
                end while
                look at next site
end while.
```

A search is done on the latest profile to find the site with the largest average wait time. This site will be looked at first in an effort to determine if any data files should be moved. Entrypoint can take any of a range of values, the simplest being 0. In this case all sites with an average wait time greater then 0 will be looked at. This appears quite reasonable especially when initially allocating files but is a poor choice when files have been reasonably well distributed. This initial entry condition, into the actual evaluation part of the algorithm, of average wait time compared to

entry-point can take a number of different forms as will be shown in the Weightings section.

The results shown in subsequent sections were achieved by giving each profile's average wait time equal weight with respect to another profile's average wait times. If it were decided that any change in usage should be emphasized the latest profile could be given extra importance by giving it more weight with respect to previous profiles. This decision would likely be very dependent on the types of system users and was not used in an effort to minimize the number of variables. Using the fact that the site currently being processed has a larger average wait time than any of the sites yet to be processed the next major value to be looked at is how this site's average wait time compares to overall average wait time for all sites.

By putting a weighting factor on this function, the evaluation can be forced to consider only cases where this site's value exceeds the norm by a certain amount, for example 10 percent. It was discovered during experimentation that when accuracy-level was assigned values greater than 20 percent system performance changed little between evaluations. When values of less than 10 percent where assigned the accuracy-level system thrashing often occurred due to excessive file movement. The results derived in the actual simulation were obtained by giving values of 1.1 or 1.2 to accuracy-level. This value is multiplied by this site's average wait time and so in this way only sites that have, in the case of 1.1, average wait time greater then 110 percent of the average of all sites average wait times will be evaluated.

4.5.2 Single Site Requests

After it has been determined that an evaluation is in order, due to the site being processed having an average wait time that is greater than the system average wait time * accuracy-level, the second major step is as follows. A search and evaluation is done of this site's query log file to determine if there are any data files residing on this site and queried only by one site. Although the number of data files requested by only one site may be relatively small their importance can not be overlooked. In settling these cases first, the number of possibilities for reallocating the remaining data files is substantially reduced.

- If a data file is only used by one site

  If the calling site's AWT * home-threshold < system AWT
  move data file to calling site
  add additional time to current site's AWT
  Else if site with smallest AWT * remote-threshold < system AWT
  move data file to smallest AWT site
  add additional time to that site's AWT
  Else leave data file at present site

An underlying premise is that if a file is only queried by one site it is likely best for it to reside there. This may not necessarily be true in all cases but appears to be for the majority. Although further evaluation may prove this to be incorrect in a particular instance, it does appear to be at least a reasonable starting point. If during this initial part of the evaluation a data file is found that is only requested

by one site it must then be decided whether to move the data file to a new site or have it remain where it currently is.

The first step is to compare the calling site's level of average wait time to the system's average wait time. The weighting, home-threshold in this case, is done to allow for file distributions that although not perfect are considered adequate. Through experimentation it was found that an assignment of 10 percent to home-threshold and 20 percent to remote-threshold worked well.

The rationale for the weighting can be seen in the following example. Let the calling site be A, with an average wait time x, equal to average system wait time and on which resides file 1. Let another site B exist with a wait time equal to 98 percent of average system wait time. Movement of file 1 from A to B may now cause B to have an average wait time greater then the average system wait time and possibly causing A to have a time equal to less than 98 percent of average. It may be best to leave the file on site A even though completely equal site times are not achieved. If it is determined that the calling site has a small enough workload and that it should also house this additional file, then the file is moved to this site and search tables updated.

A second possibility is this: the file's present site is doing more work than most other sites and the calling site is also doing above average work. In this case there may, however, be a third site that is doing below average work. What is then looked for is the site that has the smallest average wait time as a possible resident site for the data file in question.

Sites with smaller wait times are looked at as possible locations for files that should be moved from both their present site and yet not to their calling site. Here

again the need to put weightings on the wait time value arises. Two distinct possibilities exist, the second of which has been implemented in the simulation.

The first is to look at a site's average wait time and determine from the wait times and work done at a site that a data file should go to a new site. This may be calculated as simply as looking at, for example, the wait time and work done as a certain percentage of the average.

The weighting may also become a more complex function as the following example shows. A data file is still assigned as in the previous example but now the recipient site's average wait time is increased by a value directly related to file query time, so that if in this present evaluation another file may be looking at this site for residence, the average wait time will be longer then it was for the previous file assignment. It was determined through experimentation that a file with a query time twice that of another file had more than double the effect on query response time. A number of different functions were used in an attempt to capture this effect. The functions tested had the format $v = Q^e/N$, where v is the value to be added to the recipient site's average wait time, Q is the query response time for a particular file, e is a number in the range 1 to 5 and N is an integer between 1 and 10. The best results were achieved when the estimated value that is added on to a file's newly designated site is the square of the query time divided by 2. From this point on the average wait time is no longer only actual time but now incorporates a value estimated to reflect by how much the wait time is likely to increase due to the addition of the new file. This new value might be more properly called estimated future wait time. It is in this way that an effort is made to take into account the previous file that was moved to this new site.

4.5.3 Multiple Site Requests

In the previous section data files that were queried by only one site were dealt with. Although it is a normal occurrence to have a single data file queried by only one site, an equally normal situation is where two or more sites query the same data file in a set time period. The following pseudo code explains the algorithm used when dealing with multiple requests. Let A be the site that requests the data file most often and let B be the next largest requesting site.

- Assume global initialization of home-threshold

  Find the site that requests this data file most
  If A has AWT * home-threshold < system AWT
  move data file to A
  add additional time to A's AWT
  Else if B has AWT * remote-threshold < system AWT
  move data file to B
  add additional time to B's AWT
  Else look at alternative sites starting with site with smaller AWT

As shown in the pseudo code, a site querying a data file more than any other site as well as doing less work than other sites, will be assigned the data file. This assignment is again dependent on weightings put on as parameters. If this assignment would increase the recipient site's average wait time by too much, then a site with a smaller average wait time and higher throughput will be looked for in the same manner as when dealing with single requests. For both of the above cases the recipient site's

average wait time is increased by a value indicating not just a single site's request but that of multiple requests by this site as well as other sites. After evaluating a number of different functions, as was done for the single site case, the value is calculated in the following manner. If Q is the query time for a particular file and A is the number of accesses to this data file then $v = (Q^2/2) * (A^2/2)$ where v is the value to be added to the recipient's average wait time. Complicating this effort to estimate future average wait times based on data file access frequencies is the problem of actual data file processing time. If a newly assigned data file has a very large or very small processing time the actual processing time may differ substantially from the value calculated to be used in estimated future wait times. This particular problem is addressed in the next section but requires much more research than is presented here.

If it happens that two or more sites query a data file an equal number of times, the decision as to where to store the file is again based on each site's average wait time and each site's work load and in this way the file is then assigned to the appropriate site. Again if none of the calling sites are appropriate a search is done of other sites with smaller wait times and less workload in an effort to find a site.

4.5.4 Weightings

Weights are used in the evaluation, both in determining how to proceed when comparing a site's present wait time to an other site's wait time in the same time period as well as when comparing a site's wait time to that of a previous time period. When comparisons are done between different sites as well as over different time periods a number of possibilities present themselves. Some of these are shown in the following

examples.

If a straight difference is done between this site's average wait time and the overall average wait time, then even a slightly higher value for this site can be used to reassign files. This may not be beneficial in certain cases as it will only result in needless thrashing.

It was stated as one of the initial assumptions that there must be some sort of underlying trend in data file requests. It would then also be reasonable to assume that information collected in previous time periods in regard to these trends could then be valuable in evaluating present conditions.

When comparing present wait times to a previous time period's wait times a number of factors must be considered. For example should a change in values in the latest profile, profile1 as compared to profile2 be given more importance than the difference between profile2 and profile3? There appears to be no one answer to this, for it is very situation dependent. If there is much continuity in requests, then it would seem that to give equal importance to each profile might be best. It is however also quite possible that the latest change could be an indicator of major changes in access patterns and so if large changes occur it may be best to emphasize these changes.

Consider two profiles and the average wait time of a particular file in each of these profiles, then the comparison process can be made more sensitive to larger changes by a formula that uses the square of the difference in these average wait times. In this way larger changes are given emphasis and so can more easily be given attention.

This might be even further enhanced by giving data files a tag designating their importance relative to other data files. This second possibility could be used in a

new function which would then be the weighted square of the difference in average wait times. In this way specific data files could be given more or less emphasis.

When file weights are used it becomes much easier for the evaluation algorithm to determine estimated future wait time for there would then be an indicator showing how much processing time a particular data file requires. In the present algorithm a change in queries and the related change in wait times is evaluated as a simple difference function. This approach was taken due to the fact that the primary objective is to show that dynamic file allocation is viable and to this end the comparison of wait times past that of a simple difference function was considered unnecessary. This does not intend to minimize the importance of how the functions presented affect the reallocation of files, although there appears to have been little research done in in this area.

4.6 Simulation Verification

There can be little doubt that the ideal situation would have been to have real data and a real data base to work with. This was not possible however, and so a simulation was done as a substitute for an actual data base. In order to verify that the simulation is a reasonably good model, a number of simple tests were done using different inputs but with no file movement. These same inputs were then also used in determining analytical solutions. A comparison is done between analytical results and simulation results using request sets of 250 and 500. As was mentioned in an earlier section, average arrival time as well as the time required to process a request were created using a function that followed a Poisson distribution. Analytical results were obtained using common formulas presented by Deitel [16]:

$$\rho = \lambda E(s)$$

as well as

$$W = \frac{\rho E(s)}{1 - \rho}$$

Where ρ is server utilization, λ is average arrival rate and $E(s)$ is the expected service time for a request. The results shown in column one of table 4.5. were obtained by substituting the values shown in the third and fourth columns of table 4.5 into the formulas shown above.

Analytic Queue Time	Simulation Queue Time	Average Request Time	Mean Arrival Rate
11.9	8.59	4.89	6.91
14.1	11.90	5.06	6.91
19.9	29.03	5.68	7.26
13.2	13.61	5.85	7.26
9.9	7.18	4.89	7.26
11.4	11.90	5.06	7.26

Table 4.5: Comparison of Analytic and Simulation Results for Model Verification

Chi-Square results show the difference between the analytical and simulation queue times is not significant at a 10 percent level ($\chi^2 = 5.91$, d.f = 5). More and larger groups could have been included but seemed unnecessary as the simulation achieved results that corresponded well with the analytical results obtained. It is not entirely unexpected that the results were similar, as this had been the goal, but it should be noted that the simulation does include actual site to site communication and so this similarity is more significant than it may have been otherwise.

4.7 Simulation Results

Full scale simulation results were obtained using the following parameters. Simulations were done with the number of sites ranging from 2 to 10 and with the number of simulated data files assigned to each site ranging from 10 to 50. The number of requests generated per site were done in groups of 100, 150, 250, 500, 1000. The

majority of testing was done using 3 sites in an effort to meet the desirable feature of the smallest number of sites that would still constitute a distributed data base and yet allow a choice of more than one other site when reallocating data files. When a larger number of sites, and a larger number of associated files is chosen, a change in file distribution of a small number of files does not tend to give the same relative effect as when a small number of files is redistributed on a smaller number of sites. When a larger number of sites was used, for example 10, the results corresponded quite closely to the 3 site situation except that each simulation took much longer to finish. For similar reasons the number of data files used was set to a total of 30. The effect of a single file moving in a large number of files has less effect on the total system than if the number of files were less. Since it is precisely these changes that must be measured the majority of testing was done using a simulation of 3 sites and a total of 30 files.

The following discussion and testing results assumes this situation of 30 files shared among 3 sites.

In testing when fewer than 100 requests were generated per site, an even distribution was difficult to obtain due to the limited number of requests. When requests in groups of greater than 100 were generated is was possible to get requests for data files on a particular site that, while generated randomly, closely fit the probability distributions assigned to them.

The probability of requests from one site was used as a parameter, with the probabilities of requests from other sites then set to being equal. Thus, if the probability for 1 site is p, the probability for each of the other sites is $(1-p)/(X-1)$, where X is the number of sites.

Each site is thus assigned a probability which is then used to determine requests for specific files, in the 3 site case this probability was set to .33, to simulate no preference for any particular files, while probabilities of .40, .45, and .50 were used to simulate differing degrees of preference in a site's requests for specific data files.

As can be seen in the following examples, a small percentage change in a site's assigned probability can have a marked effect on the probability of a particular file being requested. Let p be the the difference between the value representing no preference for any particular file and the site's present assigned probability and let f be the number of files. If q is the probability of requesting a particular file then $\frac{p}{f} = q$ may be used to show how a small change in p dramatically changes q. The following examples are given to more clearly show these changes.

It can easily be seen that for a site requesting 10 data files a change from .33 to .40 probability results in only increasing to probability of requesting a particular data file by .007. The probability of .40 was chosen because the difference between .33 and .40 although small, may still be enough to indicate the minimum probability that needs to be assigned so as to enable a trend to be established.

When the probability of a particular site requesting 10 data files is increased to .50 from .33 it can be seen that this gives a marked change in the probability of requests for a particular data file. In this case from .007 to .017.

Files were initially distributed in two ways:

1. Evenly distributed.

2. Randomly among the sites.

The reasons for choosing these 2 distributions are as follows. It is obvious that putting all files on one site and leaving the remaining sites completely empty and having their processors idle must be the poorest allocation possible. This is much like the centralized systems of one machine with a number of terminals attached to it. A distributed database is defined as files distributed at different sites or nodes. It would seem that at the very least a reasonable starting point would be that of allocating an equal number of files per site. This is not meant to infer equal work or accesses. Another starting point might be that of random distribution. This is quite reasonable because it is quite possible that certain sites may have created more new files than other sites, either initially or since the last evaluation. Using these 2 file distributions as starting points, requests where submitted at each site in groups of 100, 150, 250, 500, 1000. The probability of each site requesting particular files was set to .33, to simulate no preference for any particular files, as well as .40, .45 and .50 to simulate different levels of preference for particular data files. When the probability of file selection by one site is set to for example .50, the remaining probability is equally divided among the remaining sites. Let p be one site's probability and let N be the number of sites. Then $\frac{(1-p)}{N}$ would be the probability of each of the remaining sites.

Results in all cases were averaged over a total of 10 simulations run for each combination of probabilities and requests. In the case of 3 sites and 1000 requests this would then result in a total of 30,000 requests being processed.

Tabulated results for the simulation of 3 sites with 1000 request per site with different probabilities of files being selected and before any reallocation of files are shown in table 4.6 for even distribution and in table 4.7 for random distribution. The values listed under Probability are meant to indicate the probability assigned

to one site's set of data file requests while values listed under Throughput are meant to reflect the number of requests served per unit time.

Probability	Throughput	Avg_Q_length	Avg_Wait_time
.33	.0658	2.12	11.84
.40	.0572	2.16	14.15
.45	.0579	2.79	12.60
.50	.0544	2.56	14.61

Table 4.6: Simulation Results Using Even Distribution of Files

Probability	Throughput	Avg_Q_length	Avg_Wait_time
.33	.0706	1.89	12.16
.40	.0361	3.93	25.42
.45	.0467	3.39	21.05
.50	.0482	3.41	19.31

Table 4.7: Simulation Results Using Random Distribution of Files

Having obtained results for these distributions without any file movement the simulation was then run in exactly the same manner except now to include file movement and the use of profile information. Using the starting points of Evenly and Randomly distributed data files, the evaluation algorithm was run after each group of requests had been completed. Groups of requests were identical to those

used when no file movement was included. The probability of particular data files being selected by any one site again ranged from .33, .40, .45, .50. The sequence of events can be described as follows:

1. Run simulation for set probability and request set.
2. Reallocate files as need be using information gathered from previous simulation.
3. Repeat steps 1 and 2 until no further reallocation takes place
4. Calculate Throughput, Queue length, and Wait times.

This sequence was then repeated 10 times for each combination of probabilities and request sets with the final result of combination being an average of the 10 runs. For example, if reallocations took place 5 times before the simulation considered the allocation correct, this would mean that for the 3 site 1000 request combination, a total of 150,000 requests were processed to get the final result for this one combination. The sequence described above took place for both even and random distributions. The results from each of these were almost the same in respect to the number of files moved in total or the number of evaluations required. System throughput as well as average queue length and average wait time were also very similar. It is for these reasons that the results using file reallocation for both even and random distributions as starting points were combined and presented as one. The results shown in table 4.8 were obtained for the 3 site 1000 request per site combination.

Probability	Throughput	Avg Q_length	Avg Wait_time
.33	.0614	2.35	11.65
.40	.0655	2.03	10.62
.45	.0663	2.14	10.89
.50	.0723	1.81	9.69

Table 4.8: Simulation Results After Reallocation of Files

The relationship between the probability of a particular site being selected and the number of times file redistribution was required to achieve either a 10 percent or 20 percent level of difference between sites is shown in figure 4.6.

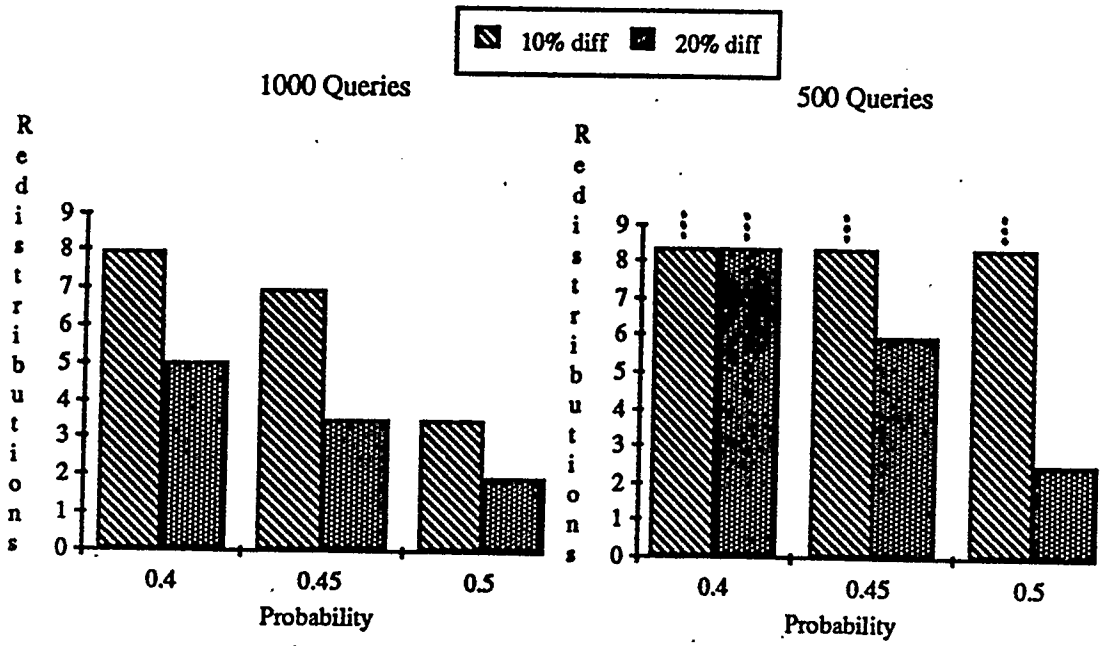

Figure 4.6: Comparison of Probabilities to Redistributions Required

Combinations for which no results were obtained are shown with open ended columns. When requests in sets of 1000 where dealt with, file redistribution made it possible to achieve differences in wait times of 10 percent or less between sites. When request sets of 500 where used, a 10 percent difference in wait time was not achievable with any of the given probabilities. When the wait time difference was set to 20 percent it was only achievable for probabilities of .45 and .50.

4.8 Conclusions

For file reallocation to succeed in reducing overall wait time there must be some sort of consistency in queries to a site. If queries are totally random and files either randomly or evenly distributed among the sites, reallocation would make little sense.

When a comparison of queue lengths and wait times is done between the results of queries to reallocated data files, as shown in table 4.8, and the results of queries to either evenly or randomly distributed files, table 4.6 and 4.7, it can be seen that the differences are quite dramatic. These are graphically shown in figure 4.7.

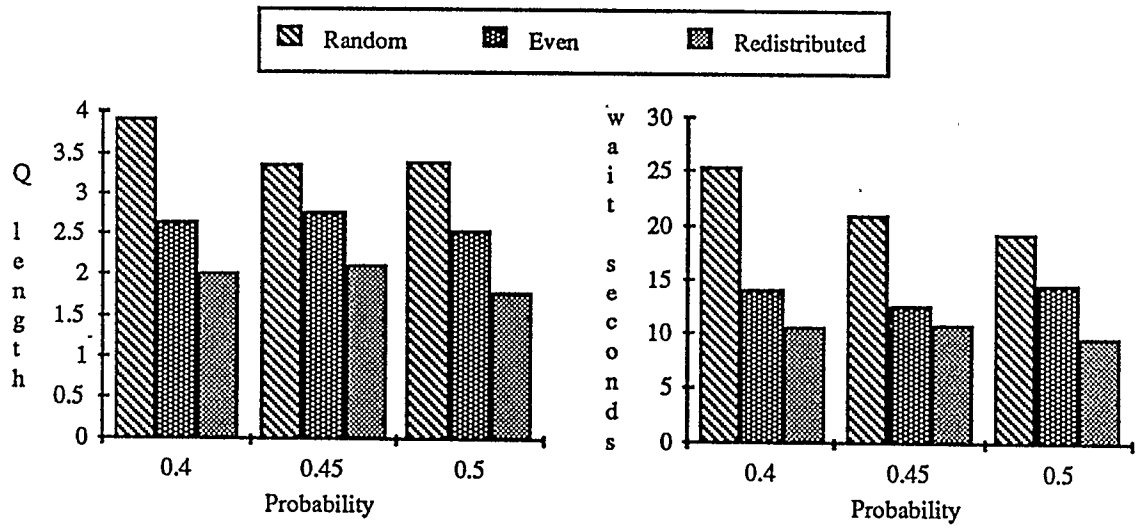

Figure 4.7: Comparison of Results Between Initial and Final Distribution of Files

One of the most dramatic changes is in wait times. The difference between simulations with reallocated files and when sites had files either evenly or randomly distributed with a probability of site selection of .50 is 34 and 50 percent respectively. Even with a .40 probability of site selection the difference is 25 and 59 percent

respectively. Not surprisingly the differences for random queries on both evenly distributed files or reallocated files is very small as indicated by throughput values of .0658 and .0706.

Queue length and wait times seem to increase and decrease in conjunction with each other. This however is on average, for during testing, situations did occur where a particular site's queue length was larger then a neighboring site's queue and yet the wait time less overall.

The closeness of throughput values for reallocated data files with probabilities of .33, .40 and .45 appears to indicate that once a good allocation of files has been achieved, the probability of site selection in the requests matters little. Only when the probability is increased to .50 from .45 does throughput increase by approximately 9 percent while wait time drops by 12 percent.

When 3 sites are involved, small changes in probability, for example .33 to .40, appear to be insufficient to allow the establishment of trends in requests. Larger changes in probability, for example .33 to .50, do allow a much quicker and more accurate reallocation of data files as can be seen in figure 4.6. As shown in figure 4.6, when reallocation is done with query sets of 1000 queries per site and a probability of .50, it took on average 3.5 evaluations and the resulting movement in files to achieve a resulting file reallocation that gave consistent wait times that differed less then 10 percent between sites.

It can also be seen that as the query probability for a site comes closer to random considering the same number of requests, it took more evaluations to reach a similar level of differences in wait times between sites. In some cases the smallest difference in wait time between sites that could be reached was 20 percent.

When attempts are made to reallocate files, and the request set became smaller or the probability assigned the site became less, it became increasingly difficult to get a proper evaluation and redistribution of files. The following example will show why this should be expected.

Let q be the probability that a given site randomly chooses a file from a file set residing there, and let Y be the number of files that a given site chooses from the file set residing at that site in n trials. Then the random variable Y has a binomial distribution with mean nq. Let $P(an < Y < bn)$ be the probability the Y falls in the interval (an,bn). Using the normal approximation to the binomial we know that the random variable

$$Z = \frac{Y - nq}{\sqrt{nq(1-q)}}$$

has approximately a standard normal distribution for n > 30. This may then be used to obtain

$$P\left[\frac{an - nq}{\sqrt{nq(1-q)}} < \frac{Y - nq}{\sqrt{nq(1-q)}} < \frac{bn - nq}{\sqrt{nq(1-q)}}\right].$$

If we let

$$Z_0 = \frac{an - nq}{\sqrt{nq(1-q)}}$$

and

$$Z_1 = \frac{bn - nq}{\sqrt{nq(1-q)}}$$

then the probability is

$$P(Z_0 < Z < Z_1).$$

If q = .50, a = .47, b = .53 and n = 500 this then gives values of $Z_0 = -1.34$ and $Z_1 = +1.34$. The probability is therefore $P(-1.34 < Z < 1.34)$. These values may now be used to read from a common standard normal distribution table to obtain an answer of $P(-1.34 < Z < 1.34) = .818$.

Results for q = .5 , .45 and .4 are as shown in table 4.9.

	q = .50	q = .45	q = .40
	a = .47	a = .422	a = .373
Requests	b = .53	b = .479	b = .427
1000	.944	.930	.922
500	.818	.790	.782
250	.660	.644	.594
100	.451	.428	.408

Table 4.9: Analytical Results in Support of Simulation

It can easily be seen in table 4.9 that even for q = .50 the probability of Y being in a reasonable range like .47 to .53 is unlikely for sets of under 250. These results appear to substantiate the inability of file reallocation past a certain point given too small a request sets. The history of a site, previous profiles, appear to be of little

help in the redistribution process except to add a degree of conservatism into file movements. This may however be due to the fact that once a file is moved to a new site its history becomes irrelevant and it takes a number of accesses in order to build a new history. The problem may also be due to the way facts on data files and data file movements were recorded. It may be that the inability of previous profiles to assist in redistribution is due to the totaling of data and that not enough specific information was retained on each particular file. It would appear beneficial if previous profiles contained information not just on the total number of local and remote accesses and time required per site, as was done, but on access frequencies and wait times per data file. In this way facts related to a data file could be compared between profiles to obtain further valuable information. It may be possible that summing accesses for a specific data file between different profiles would enable redistribution more easily for smaller request sets. This is certainly an area worthy of further study and research.

4.9 Future Research

As was shown in the previous section, a number of positive results were obtained through use of the simulation model. Among these is the fact that dynamic file allocation is possible but appears highly dependent on the data used. The area of obtaining actual system data must be considered. Values obtained through the monitoring of a real data base would allow verification of the simulation to be done and show whether or not the testing and evaluation done here does actually perform as it should. As might be expected in a project of this nature a number of areas of interest showed themselves, areas that were not at all obvious at the beginning and are certainly worthy of more research. Among these are the following:

1. The use of some sort of profile (history) needs further evaluation. It would appear that system's records generated during simulation were not accurate enough and therefore did not capture or retain all of the necessary values related to systems usage. It is important that information related to previous file queries is not lost but can be used in future evaluations.

2. With certain program files requesting the same set of data files repeatedly it would seem that considering these files as a unit merits further study. By moving file units, instead of individual files, if may be possible to reduce the number of possible locations that must be considered in evaluating file locations. An added feature may be that less communication is needed between sites because of this pairing.

3. File replication has not been used here and yet there is little doubt it holds promise. There is also little doubt, however, that even the simple duplication of files would require a much more complex evaluation algorithm. The duplication of data files introduces the need to determine which copy of a data file will be accessed by a requesting site. Among the possibilities are, does the requesting site choose the needed data file at the closest location, at the least busy site, or at the site doing the least amount of work at that time. These very interesting questions certainly merit future research.

4. A number of database systems now allow the use of a *join* between distributed portions of the database. Little if any work has been done however in determining the possibilities of automated or semi- automated file reallocation in these databases. It would appear that distributed databases are now at a point where this could now be considered and explored.

The listing of these points shows that the work presented in this thesis is by no means the only and final answer. The primary achievement of this thesis has been to provide a solid base from which answers to these areas may be achieved.

References and Bibliography

[1] Ames, J.E., Dynamic File Allocation in a Distributed Database System, PhD. Dissertation, Dep. Computer Science, Duke University, Durham N.C., 1977.

[2] Apers, P., Hevner, A., and Yao, S.D., Optimization Algorithm for distributed queries. *IEEE Trans. Software Eng.* SE - 9, 1(Jan), 1983, pp.57-68.

[3] Astrahan M.M., Chamberlin D.D., Lorie R.A., Price T.G., and Selinger P.G. Access Path Selection in a Relational Database Management System IBM Tech. Bulletin Vol.22 No.4 Sept. 1979.

[4] Astrahan M.M., et al., System R: Relational Approach to Database Management,*ACM Transactions On Database Systems*, June 1976.

[5] Bach, M.J., *The Design of the UNIX Operating System.* Prentice Hall, Inc., Englewood Cliffs, New Jersey. 1986.

[6] Badal, D.Z., The Effects of Concurrency Control on Centralized DBMS and Distributed DBMS based on Long Haul and Local Networks. In *Proceedings 2nd International Conference on Distributed Data Bases* , Berlin Germany, Sept. 1982.

[7] Bradley, James., *Introduction to DataBase Management in Business.* Second Edition. CBS College Publishing, New York. 1987.

[8] Brill David and Templeton, Distributed Query Processing Strategies in Mermaid, A Frontend to Data Management Systems. In *Proceedings International*

Conference of Data Engineering (Los Angeles, April 24-27). 1984.

[9] Chang Shi-Kuom, Liu An-Chi, A Database File Allocation Problem. In *Proceedings of the IEEE 5th International Computer Software and Application Conference* (Chicago Ill.,Nov. 18-20) IEEE, New York, 1981, pp.18-23.

[10] Ceri, S., Navathe, S., Wiederhold, G., Distribution Design of Logical Database Schemas. *IEEE Trans, Software Eng.* SE - 9, 4(July), 1984, pp.487-503.

[11] Ceri, S., and Pelagatti, G., *Distributed Databases: Principles and Systems.* McGraw Hill, New York, 1984.

[12] Chen Peter Pin-Shan, and Akoka Jacob, Optimal Design of Distributed Information Systems. *IEEE Transactions on Computers.* c-29, 12(Dec) 1980, pp.1068-1080.

[13] Daniels et al., An introduction to distributed query compilation in R*. In *Proceedings 2nd International Conference Distributed Data Bases.* Berlin Germany, Sept. 1982.

[14] Date, C. J., *An Introduction to Database Systems, Vol.1* (Fourth Edition). Addison-Wesley, Don Mills, Ontario, 1987.

[15] Date, C. J., *An Introduction to Database Systems, Vol.2.* Addison-Wesley, Don Mills, Ontario, 1985.

[16] Deitel, Harvey, M., *An Introduction to Operating Systems, Revised First Edition.* Addison-Wesley, Don Mills, Ontario, 1984.

[17] Drew, Maxwell, S. Scheduling parallel distributed relational data base query computations on a local computer network. In *Proceedings 2nd International Conference on Distributed Data Bases.* Berlin Germany Sept. 1982.

[18] Epstein, R. and Stonebraker, M., Analysis of distributed database processing strategies. In *Proceedings of the 6th International Conference on Very Large Data Bases.* (Montreal, Oct. 1-3), IEEE, New York, 1980, pp.92-101.

[19] Eswaran, K.P., Placement of Records in a File and File Allocation in a Computer Network. *Information Processing 1974.* IFIPS, North Holland Publishing Co., 1974.

[20] Ferrari Domenico, Lee Tzong-yu Paul, Modeling File System Organizations in a Local Area Network Environment. In *Proceedings International Conference of Data Engineering* (Los Angeles, April 24-27, 1984).

[21] Foster, D.V., Dowdy, L.W., Ames, J.E., File Assignment in a Computer Network. *Computer Networks 5*(Sept 1981) pp.341-349.

[22] Gardarin G., Valduriez P., Viemont Y., Predicate Trees: An Approach to Optimize Relational Query Operations. In *Proceeding International Conference of Data Engineering* (Los Angeles, April 24-27, 1984).

[23] Garey, M.R. and Johnson, D.S., *Computers And Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA. Freeman, 1979.

[24] Gavish Bezalel, Optimization Models for Configuring Distributed Computer Systems. *IEEE Transactions on Computers.* c-36, 7, (July), 1987, pp.773-793.

[25] Grosshans Daniel., *File Systems: Design and Implementation.* Prentice-Hall, Inc., New Yersey, 1986.

[26] Irani Keki, B., and Khabbaz Nicholas, G., A Methodology for the . Design of Communication Networks and the Distribution of Data In Distributed Supercomputer Systems.*IEEE Transactions on Computers.* c-31, 5(May), 1982, pp.419-434.

[27] Jensen Paul A.,*Students' Guide to Operations Research.* Holden-Day, Inc., Oakland, Ca., 1986.

[28] Laning Laurence J., and Leonard Michael S., File Allocation in a Distributed Network. *IEEE Transactions on Computers.* c-32, 3 (March), 1983, pp.232-244.

[29] Luk, W.S., and Black, P.A., On cost estimation in processing a query in a distributed database system. In *Proceedings of the IEEE 5th International Computer Software and Application Conference.* (Chicago Ill., Nov. 18-20) IEEE, New York, 1981, pp 24-32.

[30] Mahmoud, S., and Riordon J.S., Optimal Allocation of Resources in Distributed Information Networks. IEEE Transactions on Database Systems. Vol. 1 No. 1(March 1976), pp.66-78.

[31] Martin, J.T.,*Design and Strategy For Distributed Data Processing.* Prentice Hall, Englewood Cliffs, New Jersey, 1981.

[32] Mendelson, H., Pliskin, J.S., and Yechiali, U., Optimal Storage Allocation for Serial Files.*Commun. ACM* 22, 2(Feb 1979), pp.124-130.

[33] Morgan, H.L., and Levin, K.D. Optimal Program and Data Locations in Computer Networks. *Commun. ACM* 20, 5(May 1977), pp.315-322.

[34] Ramamoorthy C.V., and Chandy, K.M., Optimization of Memory Hierarchies in Multiprogrammed Systems. *Journal ACM* 17, 3(July 1970), pp.426-445.

[35] Ramamoorthy C.V., and Wah Benjamin W., The Isomorphism of Simple File Allocation. *IEEE Transactions on Computers.* c-32, 3(March), 1983, pp.221-231.

[36] Reiner D., (Guest Ed.) *IEEE Database Engineering Special Issue on Query Processing.* Sept, 1982.

[37] Sacco, G. M., Distributed Query evaluation in local area networks. *IEEE Data Eng.* 1984, pp.510-516.

[38] Smith Alan Jay, Long Term File Migration Development and Evaluation of Algorithms. In *Communications of the ACM.* 24(8) August 1981.

[39] Stonebraker Michael, *The Ingres Papers: Anatomy of a Relational Database System, edited by Micheal Stonebraker.* Addison-Wesley, Don Mills, Ontario, 1986.

[40] Tanenbaum, Andrew, S., and Mullender, Sape, J., Operating System Requirements for Distributed Data Base Systems. In *Proceedings 2nd International Conference on Distributed Data Bases.* Berlin Germany, Sept. 1982.

[41] Wah, B.W., File Placement on Distributed Computer Systems. *IEEE Computer.* Vol. 17, No. 1 (Jan), 1984, pp.23-32.

[42] Wah, B.W., and Lien, Y.N., The file assignment and query processing problems in local multiaccess networks. *IEEE Data Eng.* 1984, pp.228-235.

[43] Yu Clement T., Chang C.C., Templeton Marjorie, Brill David, Lund Eric, Query Processing in a Fragmented Relational Distributed System: Mermaid. *IEEE Transactions on Software Eng.* SE-11, 8(Aug), 1985, pp.795-810.

[44] Yu C.T., Siu M.K., Lam K., Tai F., Adaptive Clustering Schemes: General Framework. In *Proceedings of the IEEE 5th International Computer Software and Application Conference.* (Chicago Ill., Nov. 18-20) IEEE, New York, 1981, pp.81-89.