Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CAC Document Number 210
CCTC-WAD Document Number 6508

Research in Network Data Management and Resource Sharing

Final Research Report

September 30, 1976

CAC Document Number 210
CCTC-WAD Document Number 6508

Research in Network Data Management and Resource Sharing

Final Research Report

Prepared for the
Command and Control Technical Center
WWMCCS ADP Directorate
of the
Defense Communications Agency
Washington, D.C.

under contract
DCA100-75-C-0021

Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

September 30, 1976

Approved for release: Peter A. Alsberg
Peter A. Alsberg, Principal Investigator

TABLE OF CONTENTS

Executive Summary

Background

This document summarizes the progress we have made during FY76-7T on a number of research problems in network data management and resource sharing. The goal is to develop techniques applicable to the World-Wide Military Command and Control System (WWMCCS) Intercomputer Network (WIN). The work is supported by the WWMCCS ADP Directorate, Command and Control Technical Center, of the Defense Communications Agency. The work has been structured into two separate but interacting programs:

1. the design and implementation of an experimental distributed data base system, and
2. basic research on alternative techniques.

A companion report documents the progress to date of the experimental system effort. This report deals with the basic research.

The Research Program

There are two aspects to the basic research program. These are

1. a mathematical modeling and analysis task, designed to provide tools needed to evaluate system design alternatives, and
2. the investigation of specific research areas. The specific areas studied during FY76-7T included deadlock, synchronization, resiliency techniques, network file allocation, and optimization of query strategies.

Results of both phases of the program have been extensively documented. A total of five technical reports and notes describing the research effort [2,3,4,6,7] have been delivered to CCTC-WAD during the contract year.

Summary of Results

This document as a whole provides a complete summary of the investigations that were carried out and the results obtained. Here we list only a few results - those that we consider substantial contributions to the state of the art.

Modeling. Models for cost and availability were developed. These provided the tools for investigating such questions as whether remote storage in a network can be cost effective and how much data base availability can be improved by having a backup copy. We discovered, for example, that a readily accessible (e.g., on disk), up-to-date backup copy can improve availability to nearly 100 percent. On the other hand, for a wide range of realistic environments, a backup copy stored on tape, (a common practice in the WWMCCS) does not improve availability. This result is of immediate importance to the WWMCCS community.

Synchronization. This was a major research area during the year. The various methods for process and data access synchronization were surveyed and their applicability in a distributed environment assessed. In addition, an in-depth study of the update ordering problem was undertaken. The problem is to ensure that updates are applied to all copies of the data in the same order. Otherwise the copies may become inconsistent. A new scheme for maintaining update order, the "reservation center", was developed. Improvements were added to old schemes. An extensive analysis has been carried out to pinpoint advantages and disadvantages of the several schemes. We are close to being able to provide definitive recommendations as to the best scheme to use in any particular circumstance.

Resiliency. It was essential that techniques be developed to ensure resiliency in distributed operations. It is impossible to provide for recovery from all possible combinations of failures. In order to establish a criterion for an acceptable level of resiliency, we introduced the concept of n-host resiliency. That is, in order for service to be disrupted, n hosts must simultaneously fail in a critical phase of service. We worked out a scheme for two-host resiliency in some detail. Two-host resiliency is sufficient to achieve failure intervals measured in centuries. Implementation of the scheme seems feasible and not too costly. The only problem is that enough hosts must be standing by ready to play a role in the resiliency scheme so that at least two are nearly always up. In most circumstances, a total of three or four hosts should be adequate.

Network file allocation. This is the problem of determining where copies of the files of a distributed data base should be located. Formulating this question as an optimization problem mainly requires careful definition of how the files are used and where costs are incurred. However, solving the optimization problem is computationally expensive. The work required doubles with each additional network site. Months or years of computation are required to calculate optimum allocations for networks of sizes existing today (40-50 sites). In order to reduce this work, we developed three rules which determine a priori that certain sites should, or should not, be included in any optimal allocation. These rules can reduce the file allocation problem for large networks to manageable size. In some simple - but realistic - situations, the rules can provide a complete solution to the problem.

Optimization of query strategies. If copies of data exist at several sites in a network, there are many ways to choose the particular sites that will be used to respond to a given query. It is straightforward to generate the mathematical description of the optimum choice of sites. Unfortunately, the calculation of the optimum requires knowing the size of intermediate files that will be generated during the query. Theoretically, it can be harder to obtain this information than to respond to the query in an arbitrary, possibly inefficient, way. We have devised a statistical sampling scheme to obtain the needed information. That is, a sample data base is formed by selecting - in some statistically valid way - a subset of the records in the entire data base. The query is then run against the sample, and the result is used to deduce the best way to apply the query to the entire data base. The results of tests carried out to date show the method to be very promising.

Mathematical Modeling and Analysis

Introduction

During the contract year, a large effort was put into the development of mathematical models and of analytical tools. The motivation for this development was as follows. In order to determine the relative merits of various distributed data management strategies, objective, quantitative evaluations are necessary. Subjective arguments can go on interminably and lead to no convincing conclusions. But quantitative evaluations are not easy to carry out. They require good models of system behavior. Sometimes a model can consist of a simple algebraic formula that evaluates some system aspect in terms of known - or measurable - system parameters. In other cases, simple, deterministic formulas are not appropriate, and sophisticated analyses (e.g. involving queueing theory or stochastic analysis) must be carried out.

In the preceding contract year (FY75) we identified three main aspects of a distributed data management system which require modeling. These aspects are

1. availability,
2. cost, and
3. response time.

Preliminary models for each of these areas were reported on in [5]. During the current contract year, work has continued on these three areas. Two documents have been produced [4,6] and further reports will be forthcoming. We here summarize overall progress for the year.

Availability

The problem. The availability of a data base may be simply defined as the fraction of time that the data is available to users.

Many things can cause a data base to become unavailable in a network setting. If the data base is stored at the same location as the user, the system through which the data must be accessed may fail, or the device on which the data base is resident may crash. If the data base is located at a remote site on the network, the remote site or system may fail, the network may partition so that the remote site cannot be reached, or some local failure may make the network inaccessible to the user.

In most of these cases, availability can be considerably improved if a backup copy of the data base exists. If copies of the data base exist at two sites in the network, the danger of losing access because of network partitioning or site failure is reduced. Furthermore, if a local device holding all or part of the data base crashes, data may be destroyed. It is likely to be much faster (as well as more reliable) to ready a locally archived copy of the data for usage than to try to recover the lost or degraded data from audit trails, etc.

How much the existence of a backup copy improves availability depends on a number of factors. For example:

1) How available is the backup copy? (Is it stored on disk for immediate access? If it is stored on tapes, a sizeable delay may be incurred while the tapes are located, mounted, and loaded onto a rapid-access device.)

2) How up-to-date is the backup copy? (Are all updates applied to the backup copy as rapidly as possible? Is there a long backlog of updates that must be processed before the data base is really ready for use?)

3) How often is the site (or device) holding the data base likely to fail? (If failures are infrequent, the backup copy may provide little improvement in availability.)

Even small improvements in availability can, of course, be important. Availability can be over 0.99 and still be disastrously low if, say, the data is unavailable for one 24-hour period during a year and that period happens to be during a crisis. It is important, therefore, to understand thoroughly how availability is affected by the factors discussed in the preceding paragraph, and hence by the strategy used for backing up a data base.

The model. We have developed simple algebraic formulas for availability as a function of the factors listed above. Additional parameters were incorporated to model the delay incurred in initiating the process of readying the backup copy, the rate at which updates are generated, and the rate at which updates are processed. In the work reported in [4], we assumed the existence of a single backup copy, and studied the improvement in availability that the existence of a backup provides over single-copy availability. The formulas were kept simple by using average values for parameters that are actually random variables. For example, we used the "mean time between failures" in the availability formula, while system failure is actually a random process. The validity of this simplification was investigated. We concluded that its affect on computed availabilities is, in most realistic situations, to make them appear only slightly larger than they actually would be.

Adequacy of two copies. Since much of our earlier work [4] was based on the assumption of a single backup, we subsequently felt that it was important to look into the adequacy of a single backup. We therefore investigated this question and found that in general a thrid copy is not very useful. Our approach and principal results are summarized here. For a description of the model on which this analysis is based, the reader should consult [4].

Our study took the approach of trying to answer the following four questions.

1. What events might make a third copy useful?
2. How often will they occur?
3. When they occur, how long is the expected downtime when there are only two copies?
4. Can the third copy be readied in time to be of use?

In answer to question 1, we identified one basic event to be investigated. This is the possibility that the backup site fails before the primary site has been repaired. This includes as a subcase the even more drastic event that the backup site fails before its copy can be gotten ready.

The basic probability P_f that the backup may fail before the primary is repaired was worked out in [4]. The formula is

$$P_f = 1 - \exp(- T_M/F),$$

where T_M is the total time to repair the primary and get its copy of the data updated and ready for use, and F is the mean time between host failures. (F is assumed to be the same for all sites.) If T_M/F is small, P_f can be approximated by

$$P_f \approx T_M/F.$$

Thus, to a good approximation, T_M/F gives the fraction of the time that, after the primary has failed, the backup will also fail soon enough to leave the user without any copy. Suppose, for example, that $T_M/F = 0.1$. (This is a reasonable ratio; it holds for, say, $T_M = 2$ hrs. and $F = 20$ hrs.) With failures of the primary occurring once every 20 hrs., and the backup also failing one time in ten, it seems that a third copy would be put into use once every 200 hrs., or about eight days. This begins to look like the third copy may not be worthwhile. But question 3 is also important. How long a period of downtime is expected? Offhand, one might suspect that usually the primary is "almost" ready when the backup fails, so that little is gained. However, our analysis shows that this is not the case. If the backup fails within the repair time T_M of the primary, on the average this failure will occur at time $T_M/2$. Thus, there will be an average time period of $T_M/2$ (one hour, for the parameters of our example) when primary and first backup are both down and a third copy would be useful. One hour of use every 200 hours may not be enough to make a third copy worthwhile. However, that hour could occur at a highly critical time.

Finally, we need to investigate whether the primary is likely to be repaired and ready before the third copy can be readied. If so, the third copy is not of much use. In order to analyze this question, we assumed that the repair time for the primary is not a constant but has a probability distribution as described in appendix 2 of [4]. Using 1/2 hour for the time to ready the third copy and realistic values of the other parameters, we found that the probability that the primary is repaired and ready before the third copy can be gotten ready is over 0.7. That is, more than two times out of three readying the third copy will have been a waste of time. Thus, instead of being useful about every eight days or so, the third copy will only be useful about once a month. (The details of this analysis will be reported in a forthcoming technical note.) Furthermore, the reader should remember that, with primary and backup down for an hour and the third copy requiring 1/2 hour to ready, "useful" means elimination of 1/2 hour of downtime. It is highly questionable whether it is worthwhile to maintain a third copy just to eliminate less than an hour of downtime once a month.

Conclusions to date. Including results from our earlier study [4], we may summarize our conclusions on availability as follows.

1. A backup copy can improve the availability of a data base by as much as 5 to 10 per cent. To put this result into more concrete terms, suppose that a single copy is likely to be down for two hours per day (availability = .917). A 5 percent improvement would produce an availability of .963, or a reduction of probable down time from the original two hours to about 54 minutes.
2. If the backup copy is readily accessible and kept reasonably up to date, the availability is very close to 1. On the other hand, if the backup copy is stored on tape, so that it is relatively out of date and locating it is a time-consuming process, availability may be little better than was provided by a single copy. (This is because one can probably repair the original system about as rapidly as one can ready the backup.) Indeed, a backup of this sort tends to be mainly useful for recovery from some accident which destroys data in the original data base.
3. A third copy is unlikely to increase availability by any significant amount.

It should be emphasized that these conclusions are based on the assumption that failures are nicely random. In examining data on failures and repair times that have recently become available from MIT Multics, we have observed that this assumption is questionable. There appear to be occasional clusters of two or three failures, followed by rather long failure-free periods. Often the failures in a cluster stem from one root cause which is only located with difficulty. Thus it may be wrong to assume (as we do) that individual failures are independent, random events. We plan to analyze failure data over a long time period to see if our model needs revision and, if so, whether our basic conclusions are affected.

Cost

In developing models for the cost of distributed data management, a number of difficult questions must be answered. Where are the costs of processing a query or performing an update actually incurred? How should system costs be pro-rated over time, or among programs in a multi-programming environment? Should costs be simply dollar costs, or should they be broadly defined as general weighting factors and assigned values on the basis of, for example, scarcity of resources? Because of the complexity of the problem, there have been few documented attempts to develop cost models with much detail. One such attempt was that made by Lum and co-workers at IBM Research to model data retrieval from a memory hierarchy [12]. In our early work on cost modeling, we built on Lum's model. This work has been reported fully [6] and we give only a brief summary of it here. More recently we have attempted an ambitious, *ab initio* cost model. The emphasis has been on computing the cost of a job or process from the point of view of the individual user. This seemed to be the most valid basis for making comparisons of various detailed strategies for handling data. (An attempt to compute the global cost of maintaining a complete data management system would probably yield less useful information with far more work. It could be done, however, by computing results for various "typical" usage patterns and averaging over these.) This work is still in progress and will be reported in a later document. In this report we briefly indicate the progress to date.

Preliminary model. The model developed by Lum et al. [12], describes a data staging process in a hierarchical memory. That is, the data is assumed to be stored on a slow, cheap storage device when not in use and transferred to a rapid, expensive device for accessing. What attracted us to this model was its fineness of detail and the ease with which we felt we could extend it to a network situation by including (as part of the hierarchy) devices at a remote site. We did, however, have some reservations about the model. Nevertheless, we decided that the questions we had about it might be more easily and more rationally resolved after we had experimented with it and better understood its limitations, as well as its good points. We therefore began with Lum's cost formula, with its terms for storage, data transfer, and accessing, and added network terms - including costs for data transfer to, from and over the network, as well as protocol costs. In adding these terms, we felt that, if only for consistency, we should follow the spirit of Lum's model. Hence the extended model also has questionable terms, such as those involving costs of "lost" CPU idle time. The model also has serious limitations, such as no provision for remote processing.

In spite of its limitations, we believed that the preliminary model was adequate for an initial study of the key question: Is it ever more economical to store data at a remote site (instead of locally) and bring it over the net when needed? We used the model to study this question and arrived at the basic conclusion - which we believe to be valid for real systems - that heterogeneity is a necessary requirement for remote storage to be cost effective. In fact, the cost differential due to heterogeneity must be sizable - not small percentages, but orders of magnitude.

Framework for better models. After working with the preliminary model and getting to understand its weaknesses, we believed that we could proceed to build a more useful model. The preliminary model did not lend itself to evaluation of many important network strategies, such as remote processing and front-ending. To avoid these limitations we have taken a different approach and developed a general framework within which a wide variety of models may be represented. Within this framework, each functional part of the system configuration or strategy under consideration is represented by a module. These modules are then strung together to represent the entire system configuration to be evaluated. This approach will provide a sufficiently flexible framework to allow consideration of a wide variety of system configurations and software architectures for models of different aspects of computer networking. Thus, the framework should be not only good for cost modeling but also for modeling response and other features.

Each module, no matter what its function, is assumed to have a standard format consisting of four sections:

1. the means by which requests are made to it,
2. the means by which support from other modules is requested,
3. the nature of the computation, and
4. the characteristics of its use of secondary storage.

Each of these sections can be characterized by two sets of parameters:

1. those that characterize the hardware the function resides on, and
2. those that characterize the function itself.

Thus, if one wishes to construct a model (such as a cost or response model) within this framework, it is necessary to develop the general equations for each of these four sections in terms of the parameters of that model, and then determine the proper values or range of values for the parameters which describe the hardware and the functions under consideration. In some modules, one or more sections may be vacuous. For example, only a few functions require access to secondary storage.

Most of the models that will be developed within the basic framework may be described by the following scenario:

> Inputs or requests of some length are submitted to the first module. It performs its "function", which requires utilization of the sections of that module. Communication is established with the next module, and requests for support (possibly more or less than were input) are generated. (Note that it is not necessary for each function to use all sections of its module.) Requests continue to be generated to subsequent modules until the original task is completed.

Such a scenario is followed, for example, in responding to a query. The user types his query into a terminal (the first module) which transmits it to a host. After proceding through some host software (other modules), the query is handed to the data management system (a module with access to the data base in secondary storage). The data may then be transmitted through a succession of modules, which may perform complex data analyses, before the response is finally returned to the user.

Given this kind of a scenario, there are a few general things we can say about what sort of parameters will enter into the equations to describe the various sections of the module. The intercommunication sections (which receive or transmit requests) will tend to depend on the number of separate communication set-ups, the number of requests per unit time (some models may require an interarrival time), and the average length of a request. The utilization of the secondary store will generally depend on the nature of the function in terms of the average number of accesses and number of opens generated per request. It should be noted that the module structure makes no assumptions about the physical secondary store. The structure definitely does not imply that each module has a separate physical secondary store. Different kinds of secondary storage or the finiteness of storage can be taken into account by the equations and boundary conditions built into a particular model. The computation section is, in a sense, the heart of the module. Equations for this section describe, in more or less detail, the function of the module. Parameters fall into two general classes, "application" and hardware". Application parameters include not only those describing the internal workings of the section, but also information transmitted from other modules. For a high-level model of a well-specified process, the externally generated information may consist primarily of numbers of requests received and lengths (or complexities) of those requests.

Use of the framework for cost modeling. We have developed detailed cost formulas for the storage access and intercommunication sections of a typical module. The formulas are lengthy and contain many parameters. We will therefore not write down the formulas here (they will be given in a forthcoming document), but will qualitatively describe what goes into them.

Accessing secondary storage. The cost of accessing secondary storage is broken down into four components

1. the static cost of storing data,
2. the cost of opening a file,
3. the CPU cost of servicing the access, and
4. the channel costs.

These last three are dynamic costs, incurred for each access. They must be multiplied by an access rate to get a cost term compatible with the first term. Costs of both user and system buffers are taken into account in the storage cost and the buffer allocation strategy plays an important role.

We made the tentative decision to include the cost of opening a file as a single, measured quantity. We felt that the mechanics of a file open are sufficiently different on various systems that any attempt to represent an open in finer detail would introduce considerable complexity without contributing significantly to the model.

To determine CPU costs, we examined in detail the steps that take place in satisfying a typical disk access. The steps include the system call, decoding of the request, queueing of the request (if the data is not already available), and (later) a disk interrupt and process switch to transfer the data to a user area. The process switching and queue handling which the system must carry out are time-consuming and add important components to the CPU costs. Some of the factors in the CPU cost term necessarily involve probabilities - probability that the process blocks, probability that a disk access is required, etc. Calculation of such probabilities would require rather complex stochastic submodels, involving both program and system characteristics. It might be more reasonable to attempt to measure these probabilities.

Finally, the model must account for the channel costs that come from moving blocks of data from the disk to the primary store. This term involves the data transfer rate and block size, as well as the actual cost of using the channel.

<u>Interprocess communication</u>. The costs incurred in interprocess (i.e. intermodule) communication will depend upon the form of communication linking the modules. We have worked out cost equations for four basic types of interprocess communication:

1. message queues,
2. shared core,
3. shared files, and
4. hardware channel.

Although the detailed scenarios that must be modeled differ, the cost formula for each of these contains such basic factors as CPU costs for process switching and system calls, as well as message sizes and cost of the data transfer medium. For example, the cost of the shared-core mechanism must include the cost of the storage area in core reserved for this process.

Cost of computation. The equation for this section of a module will depend upon the specifics of the module's function. However, cost factors that will contribute in general are CPU cost to carry out the function, the cost of page faulting, and the cost of user primary storage (working space).

Current and future work. The basic framework for sophisticated cost modeling has been constructed. The major effort now and in the future is to apply the framework to the development of models to study specific aspects of distributed data management - e.g. remote processing of queries, front-ending, etc. The determination of the parameters to be inserted into the models will require extensive work. Some parameters can be measured; others might require the development of submodels. Nevertheless, this work is necessary if reliable conclusions are to be reached for real systems.

Response Time

In a complicated computer system, the time it takes to complete a process is not easy to determine theoretically. Not only the actual processing time must be included, but also all the time that the process spends waiting in queues. The latter depends upon both the process of interest and its interaction with all the other processes concurrently in the system.

The higher the level of the model, the less one need be concerned with these difficult details. For this reason, our preliminary work (reported in [5]) assumed a simple linear dependence of response time on system load. Essentially, we looked into the question of when, because of load differentials, a local user can get a faster response from a remote site (including network delay) than he can get locally. We were suspicious of the linearity assumption, but further study during the current contract year appeared to verify its validity. We therefore felt that, as a high-level model, that reported in [5] was reasonably correct and complete; hence we did not pursue that line of work further.

Clearly, a lower-level, more detailed model was needed to answer less trivial strategy questions. In looking into the literature, we found little in the way of response-time analysis dealing directly with questions of data management. However, both time-sharing systems and multiprogramming systems have been analyzed. Both of these types of systems are characterized by competition for shared resources. Several jobs reside in the system simultaneously and must occasionally wait for processing, I/O, etc. The natural mathematical models to describe the progress of jobs through such a system of waiting lines and processors are those of queueing theory. Accordingly, queueing theory has been heavily and successfully used to develop formulas for response time in such systems.

It should be possible to develop models of this type which are specific to data management - i.e. in which the concurrent jobs are queries or updates. An attempt at developing such a model was made by Chu [9] in his work on network file allocation. Chu's work was simplified by his view of a query/update as a simple file transfer. Chu further neglects all queues but one (that for sending the file to the user) and hence treats what is a very difficult problem as a single-server queueing system. To make even this problem tractable, Chu makes other simplifying assumptions, notably that all files have the same length and that there is only a single copy of each file in the network. We have attempted to relax these assumptions, in the hope of obtaining a more realistic model without making the mathematics completely intractable. This work is still in progress and will be reported later.

As another approach, we are using the general modeling framework discussed above to model response time as well as cost. The formulas need a few modifications. Terms such as storage costs may simply be deleted. The CPU cost terms are of the form of a product of the time it takes to carry out some process, times the CPU cost per unit time. This last factor is omitted to yield a time term. It is necessary to insert additional terms giving the time spent waiting in queues. And obtaining this waiting time may require a queueing analysis. However, we believe that we understand what is needed to develop a response-time model by using the modular framework. Working up response-time models for specific applications is expected to be a bit more difficult than for cost models, but still feasible.

Introduction

In parallel with the development of mathematical models and analytical tools, we made in-depth studies of a number of research areas. These areas are:

1. deadlock,
2. synchronization,
3. techniques to ensure resiliency in distributed operations,
4. optimal allocation of files in a network, and
5. optimization of query strategies.

Much of our progress in these areas has been reported in detail in previous documents. Here we summarize overall progress to date and emphasize results not previously reported.

Deadlock

The deadlock problem and techniques for treating it were discussed in detail in a previous technical report [2]. To summarize, we found that this problem has been very heavily studied over the last six years. The basic techniques for handling deadlock fall into three classes:

1. Detection and recovery.
2. Avoidance, or elaborate schemes for assigning resources to processes only when it is "safe" to do so.
3. Prevention, or the imposition of some discipline on the processes (e.g., requiring them to ask for needed resources in a prescribed order) which precludes the circular blocking configuration characteristic of deadlock.

We assessed the applicability of these techniques in a distributed environment, particularly with reference to data base access control. We concluded that avoidance is probably too expensive and time-consuming to be practical. The prevention schemes also may impose too much of a burden on the system or, alternately, on the application programmer. Simple detection schemes may be feasible in a network. We proposed one such scheme which, unlike one previously discussed in the literature, does not require a central monitor.

There is also the possibility that deadlocks occur infrequently enough that the problem may be ignored; i.e., that a deadlock may be handled like any process failure. Several studies exist in the literature on the expected frequency of deadlock. As one might intuitively expect, as data bases grow larger, contention for the same items (and hence the probability of deadlock) should decrease.

Existing computer systems tend to use a combination of techniques for handling deadlock. Different schemes or disciplines are applied to handling different types of resources. Some possible sources of deadlock are even ignored. The practicality of combining techniques in worth keeping in mind, since there appears to be no single "best" way to handle deadlock.

An important point to notice is that the choice of synchronization mechanism (see below) will impact on the deadlock problem. Synchronization techniques need to be studied with one eye on their potential for causing deadlock. If synchronization is handled properly, the need for a separate concern with techniques for handling deadlock can be minimized. For this reason, we did not extend our study past the basic state-of-the-art assessment and the preliminary development of a distributed detection scheme (i.e., past the work reported in [2]). Sophisticated technology is available for use. For any distributed data management system, the problem of synchronization must first be solved. Then one must determine where deadlocks can arise and how these potential deadlocks are best handled. The problem must be solved within the context of a particular system. There seems to be no quick, general solution. Barring an unexpected breakthrough which yields such a solution, we believe that in the future useful work in this area will be limited to engineering (i.e., application of known techniques) and experimentation.

Synchronization

Synchronization mechanisms are needed to ensure that concurrent processes do not interfere with one another. In the context of data management, the need is particularly acute. Concurrent updates may interfere with one another in such a way that the data becomes garbled or inconsistent. Updates may interfere with queries and cause the user to get a nonsense answer.

Semaphores and locks. In our previous report [2], we surveyed existing techniques for synchronization. First, there are *semaphores* by which processes communicate with one another. Semaphores are essentially variables which can be accessed (read and altered in a strictly defined way) by all the processes to be synchronized. By modifying the values of such variables, a process can communicate its state to the other processes. Conversely, by examining the values of the semaphores, a process can determine whether or not it is safe to proceed.

Second, there are *locking mechanisms*. Similar to semaphores, locks are, however, directly attached to data (records, blocks of records, etc.). They give the state of the data (essentially whether or not it is in use) instead of the state of a process. Thus, by locking the relevant data, an update program can ensure that no concurrent program will interfere with it.

Problems with semaphores. We identified a number of difficulties which semaphores may encounter in a distributed environment.

First, semaphores reside in shared memory; all the processes to be synchronized must have ready access to the semaphores. Second, even if it is feasible for the semaphores to be stored at one reliable site on the network, the time delays involved in network communications are likely to make synchronization prohibitively slow. Third, the time delays, which may vary considerably, have other adverse affects. They make the problem of races much more severe than it is within a single system. Furthermore, extraordinarily long delays may be virtually indistinguishable from failures. Serious trouble can be caused if a process attempts error recovery on a (remote) process which is still working properly, albeit slowly. Fourth, problems of errors become more severe. Much care needs to be taken to see that communication among processes is resilient to lost and delayed messages, and to other common errors.

Resiliency. We have addressed this last problem - the resiliency problem - in some detail. Essentially, our approach is to back up each critical message and to wait until at least two sites have received it before taking (or allowing the user to take) any action. The approach, as applied to synchronization primitives, is sketched in [2]. Further development and analysis of resiliency techniques were reported in a later document [3]. This work will be summarized briefly in a later section.

Assessment of locks. Although locking mechanisms are well understood, certain problems have required further study. One of these is the data "level" at which exclusive use is assigned. It may be that usage is assigned on a record-by-record basis. Or, at the other end of the scale, only one user at a time may be allowed access to the entire data base. The choice of level is a difficult decision. In [2] we discussed some of the tradeoffs which should be considered.

A problem specific to distributed environments is that of using a mechanism which minimizes network traffic and network delay. In particular, there is a need to minimize the number of synchronizing messages which must be sent across the network. Looking at current single-site synchronization techniques in the light of this requirement, we found [2] that the frequent, explicit setting of semaphores or locks appears to be too time-consuming. The most promising techniques seem to be those in which only the transaction requests themselves, with perhaps some information on the order in which they should be performed, are transmitted across the network. The local data managers are then responsible for setting locks or enforcing a usage protocol that avoids interference. More work needs to be done in this area, especially in conjunction with experimentation.

Maintaining consistency among multiple copies. In order to maintain consistency among multiple copies of data, it is important to ensure that the updates are applied to each copy in the same order. For example, if one update adds 10 to a field and a second increases the same field by 10 percent, interchanging the order of the operations will change the final result. There is very little in the literature on this problem. We have therefore made this topic the subject of a major research effort. In our report [2], we briefly indicated our preliminary findings. A final report on this work is forthcoming. Meanwhile, we will here discuss our progress to date in some detail.

There are essentially two different ways to maintain the update sequence. One way is by seeing to it that all sites receive the updates in the same order. The second is by explicitly attaching a sequence number to each update. The first scheme runs something like the following. An ordering of the sites holding a copy is established.

Each update is first sent to the host designated as the primary. The primary then sends the update to the next host, and so forth. The precise sequence of the updates is therefore determined by the order in which they arrive at the primary. Some safeguard must be taken to see that updates from the same user will arrive in order (if that order is important) and that the primary (or some later host) does not somehow mix up the ordering. To ensure that no mix-ups occur, it is advisable that the primary attach a sequence number to the update before sending it along to the backups. This consideration then leads us to emphasize the second approach.

The second approach is to attach explicit sequence numbers to the transactions. If each update in the network, no matter where it was generated, has a unique sequence number attached, then every data base manager can use these numbers to order the operations. The question that then arises is how the sequence numbers should be assigned. Three different techniques are presented here. These are

1. centralized assignment (primary host),
2. partially distributed assignment (the "reservation center,") and
3. completely distributed assignment.

Centralized assignment: The most obvious way to assign sequence numbers is to have a distinguished host (the primary) from whom sequence numbers are requested. In this scheme, the host generating the update sends it to the primary, where a sequence number is attached. At this point, two variations are possible. (1) The update, with the assigned number, may be returned to the originating host for transmission to the various data base copies. This scheme envisions the number-assigner as a very simple piece of software which does nothing but hand out numbers on request. (2) The primary may have the responsibility of broadcasting the update (with its sequence number) to all the copies. This scheme is functionally equivalent to the approach (discussed above) which orders the sites. Adding sequence numbers can be thought of as one way of providing some of the needed resiliency; i.e., of ensuring that none of the sites destroys the order of the transactions. A centralized assignment scheme with updates broadcast by the primary has been discussed in some detail by Bunch [1]. It is this alternative that we will be discussing here as the "centralized" or "primary host" scheme.

Centralized sequence number assignment has several apparent defects. First, the primary might be a bottleneck, because it must assign a sequence number to each transaction. However, compared to the work involved in applying the update to the data base, the additional work of assigning a sequence number is quite small. Second, each transaction can incur delay because it must wait for the sequence number to be assigned. The transaction delay should be negligible (a few hundred milliseconds) at the primary site itself. However, the host generating the update might find it inconvenient to wait for a number to be returned from a remote site before applying the update.

Partially distributed assignment: A more distributed scheme that we have developed is the socalled "reservation center" (RC) scheme. In this scheme sequence number assignment is a two-step process. First, the reservation center, a distinguished host, gives each host a block of sequence numbers. These numbers are valid only during a limited time interval. All numbers issued are ordered. When a host wishes to initiate a transaction, it takes the next unused sequence number from its block of numbers for the current time interval. Some strategy must be devised to handle the case when more numbers are requested than are available. A simple way to handle this is for each host to have two sets of numbers.

When the first set is used up, the host begins to use the second and simultaneously notifies the RC that it needs a new set.

Some of the problems of the central number assigner are ameliorated by using the reservation center. For example, when the number assigner fails and this task is transferred to a new host, there may be some difficulty in determining the "next" number to be assigned. There is no such problem if the reservation center fails. Instead, sequence numbers from the next time interval are assigned. They will be greater than any previously sent numbers. The reservation center is also less likely to be a bottleneck because its response is not critical for the continuation of an update, as is the case with a primary copy. The workload of the reservation center is more periodic and predictable. Every time interval it must make up and transmit new blocks of sequence numbers. There is a minor problem in determining exactly which sequence numbers have been used. This is necessary to detect a failure in which updates have been lost.

We noted above that there is a delay in applying locally generated updates to the local copy when the site must wait for the sequence number to be returned from the primary. This is not the case in the RC scheme, since the local site applies a sequence number from the list it already has. It is then ready to apply the update - although it may have to wait for an update with an earlier sequence number to arrive and be applied first. However, since blocks of sequence numbers expire after a fixed, brief time interval, this delay will necessarily be fairly short.

Completely distributed assignment: If we let each host generate its own sequence numbers, we have a completely distributed scheme. One such scheme has been described by Johnson and co-workers

[10,11]. Their scheme was more motivated by a concern for maintaining the temporal sequence of updates; i.e., maintaining the precise order in which the updates are generated by the users. They tried to achieve this by generating the sequence numbers partly from the local clock time. The required uniqueness of the sequence numbers may be achieved by appending host id, user id, etc., to the timestamp. There is an interval of uncertainty because the clocks on the various hosts may not agree. One would expect this interval to be small in the WIN environment because the military already requires good time coordination. Since this is not true in all environments, we have developed techniques for detecting and correcting an inaccurate clock. Essentially, timestamps on updates arriving from remote sites should agree with the local time plus an adjustment for network transmission delay. A more or less sophisticated analysis can be used to determine the probability that a particular clock requires adjustment, and by what amount.

There is, however, a similar interval of uncertainty in the other schemes considered. In a centralized scheme, network delays may cause updates from different sites to be ordered quite differently than their precise time of generation would indicate. The reservation center approach was in large part developed to formalize the viewpoint that the precise sequence of updates within a certain time interval doesn't matter. It appears that overconcern with applying updates in their "real" order is something of a red herring. Even for a single-site time-sharing system, the order of transactions from different users may sometimes be determined as a random outcome of the terminal polling process.

In briefly discussing the schemes for sequencing updates, we have touched on some of the more obvious advantages and disadvantages.

There are many more points which need to be considered in deciding which of the schemes is best used in a particular distributed environment. We here list and briefly discuss several of these.

1. Storage requirements. In Johnson's timestamp scheme, as well as in the reservation center scheme, the timestamp of the last update applied to each record (or even each field of each record) is appended to the record. This is done so that the data base manager can look at the record and determine whether an update was applied out of order. If there is a central sequence number assigner, all updates theoretically are applied in the assigned order, so that this enormous storage overhead is not necessary.

2. Operations supported. This is a key problem that we have studied extensively. If the operations which one is allowed to perform on the data are severely restricted, almost any synchronization scheme will work. For example, Johnson and his co-workers assume that only assignments are allowed. This solves some tricky problems. Superseded assignments may be simply thrown away. On the other hand, if only increments and decrements are allowed (as for an inventory system) the order in which these operations are performed doesn't matter, and no sequencing scheme at all is needed.

In a realistic, multi-operation system, assignments, increments, decrements, etc., will all be allowed. If an update arrives at a site out of order (as determined by explicit sequence number), it may cause earlier arriving updates to have to be redone. That is, the system must have some provision for undoing and redoing operations. We have looked in some detail at the work that might be involved in carrying out such a process. If the updates which get out of order are independent, in the sense of having no effect whatsoever on each other, then there is no problem. But in general out-of-order updates may not only affect specific field values (as in the example given at the beginning of this section) but also may alter the applicability of other updates. Consider the following simple example. Suppose that, in a military personnel file, update u_1 promotes a number of persons to the rank of captain. Suppose that update u_2 increases the salary of all captains by $2000. Now if the updates are applied in reverse order (u_2, u_1), then all of the newly promoted captains will (alas!) miss out on their pay raise.

We have analyzed the problem of two out-of-order updates (rearranging u_2, u_1 to u_1, u_2) in some detail. As many as eight distinct cases (types of interaction between the updates) must be considered. If we procede one step further and try to rearrange u_2, u_3, u_1 to u_1, u_2, u_3, we find the problem to be so complex as to be virtually intractable.

As an alternative to rearranging out-of-order updates, the system could wait for "missing" sequence numbers - an approach that is feasible only if all numbers are assigned centrally and all sites receive all updates. We have also done some investigation of a delay pipeline as a flexible way to impose a waiting period. The idea is that as each update arrives at a site, it enters a pipeline, or queue. After a certain time delay, the system is ready to take the update from the pipeline and apply it. Before doing so, the system checks to see whether there is any other update in the pipeline which should be applied first. If so, the updates are applied in the correct order. Such a pipeline could take care of updates which get out of order by small amounts due to the normal fluctuation in network transmission delays. However, the scheme is not adequate to handle seriously out-of-order updates. Serious disordering will inevitably occur when hosts go down or the network partitions.

3. Application delay at remote sites. In discussing the various schemes, we have noted that they differ in how rapidly an update may be applied at the site which generates it. They also differ in how rapidly updates may be applied at the remote sites. Again, the timestamp and RC schemes have the advantage, since updates may be almost immediately broadcast to remote sites. The process of going through a primary copy necessarily causes some delay. And in the implicit ordering scheme, in which updates percolate down the list of copies, the delay to reach the last one may be considerable.

4. Other considerations. A number of less critical features have been studied. For example, one scheme may cause less network traffic than another. Detailed analysis of the schemes - and perhaps even of particular implementations - are required to give fine comparisons on this basis. No reason for large differences among the schemes is evident. There is, however, a difference in the sensitivities of the schemes to network traffic. If the traffic into the primary is very high, increased network delay may adversely affect the primary-host scheme. Similarly, an overloaded primary host could be a real bottleneck.

Table 1 summarizes our ranking of the three schemes (primary host, reservation center, and timestamp) with respect to various considerations. A "1" indicates "best", a "3" worst. The implicit scheme discussed at the beginning of this section is not listed separately, since it has the same general properties as the primary host scheme. The latter, however, has advantages with respect to resiliency, since lost updates or dead hosts can be identified through missing explicit sequence numbers.

Basis of Comparison	Timestamp Scheme	Reservation Center Scheme	Primary Host Scheme
Maintenance of "real" update sequence	3	2	1
Local applicaton delay	1	1	3
Remote application delay	1	1	3
Storage requirement	3	3	1
Variety of operations supported	3	3	1
Sensitivity to host failures	1	2	3
Sensitivity to network traffic	1	2	3
Sensitivity to host load	1	2	3

Table 1

Assessment of three schemes for assigning sequence numbers. A "1" indicates that a scheme is the best on this basis; a "3" indicates that it is the worst.

In attempting to make an overall judgment as to the best scheme, we have to weight the various considerations with what we believe their relative importance. In addition, we have to consider the magnitudes of the differences indicated by the rankings of table 1. For example, all the schemes are rather sensitive to host failures. The timestamp scheme has only a small advantage, and that disappears if the primary host scheme is made more resilient. On the other hand, the problem of supporting complex operations seems insurmountable for the timestamp scheme. After weighing all the evidence, we favor the primary host scheme. We feel that its advantages far outweigh its disadvantages. Since this opinion represents a change from that of several months ago, we hesitate to claim that now we have the ultimate answer. More analysis and experimentation is needed to make the case for the primary host scheme completely convincing.

Resiliency Techniques

Early in our study of multi-copy management, we identified two problems which required in-depth study before we could proceed with further work. One was the synchronization problem, which has been discussed above. The other was the resiliency problem. A recent technical report [3] contains the details of our work on resiliency. The following is a brief review, largely abstracted from that more complete report.

Need for resiliency. The current protocols on the ARPANET (and similarly in its "copy" PWIN) operate under basic assumptions that are at best questionable in a production networking environment. It is assumed that all hosts correctly obey protocol. It is assumed that no host is malicious. It is assumed that messages are not lost in the network. It is assumed that when a host fails, it will fail at a "convenient" point in the execution of a protocol sequence. In fact, all of these assumptions are commonly violated every day in the ARPANET environment. What is required for production networking are resource sharing strategies which are as resilient as possible to protocol violations, malicious attack, communication failures, and host failures.

Characteristics of resilient service. "Resiliency" is a term which might mean different things to different people. It is clearly unrealistic to expect a resilient service to be completely free of errors. In our work, we have assumed that a resilient service should have four major attributes.

1. It should be able to detect and recover from a given maximum number of errors.
2. It should be reliable to a sufficiently high degree that a user of the resilient service can ignore the possibility of service failure.
3. If the service provides perfect detection and recovery from n errors, the (n+1)st error should not be catastrophic. A "best effort" is made to continue service.
4. The abuse of the service by a single user should have a negligible effect on other users of the service.

In short, the user of a resilient service should not have to consider the failure of the service in his design. He should be able to assume that the system will make a "best effort" to continue service in the event that perfect service cannot be supported. Finally, the system should not fall apart when the user does something he is not supposed to do.

It is important to establish criteria for acceptable resiliency. We have introduced the concept of n-host resiliency. That is, in order for service to be disrupted, n hosts must simultaneously fail in a critical phase of service. It may be possible for n or more hosts to fail outside of such a critical phase without disrupting service.

Techniques for achieving resiliency. In [3], we describe several alternative schemes for implementing two-host resiliency with respect to communication system and host failures. A linearly ordered set of hosts is assumed to have the responsibility for resiliency. In one scheme, users send critical messages to the primary host, which relays them to the first backup. The backup then sends acknowledgments to the primary and to the user, as well as relaying the messages to the second backup. Thus, both the primary and the first backup have knowledge of a message before the user gets an acknowledgment and may proceed further. It is so unlikely that both primary and backup will fail at a critical point in this process that two-host resiliency is sufficient to achieve failure intervals measured in centuries.

Feasibility. The resiliency criterion is a question of service integrity. If the criterion is met, the service will almost certainly be functioning correctly. Unfortunately, large service hosts are down for substantial periods of time during the day for both scheduled maintenance and unscheduled failures. A possibly large number of service sites may be required, simply in order that there be a reasonable expectation that at any time at least two of them will be up. This raises an important issue related to resiliency - the availability of resilient service.

We have analyzed this question in some detail. Results are tabulated and graphed in [3]. To summarize, we found that when resilient service must be supported in a WWMCCS-like network, it appears that three or four service hosts must be supplied. This will support a service availability in an acceptable range during non-crisis periods.

In a crisis situation, by judicious management of down-time parameters, the same three or four service hosts should be capable of providing significantly improved availability for the duration of the crisis.

If only one service host is available, it would be possible to discard the resiliency scheme and to operate with only the single host. We have investigated the question of what allowing this degradation in resiliency would cost us in the way of service integrity. In a single host environment it is possible to have undetected errors creep into distributed resources like data bases. The probability of service errors is greatly dependent on the frequency of the errors that would normally be detected by the resiliency scheme. If, for example, these normally detected errors occurred at the rate of one every one to four hours (a reasonable assumption based upon ARPANET experience), then a two-host service which was permitted to operate in degraded one-host mode would experience undetected errors at the rate of one every day or two. If three service hosts were permitted to degrade to single-host service, then the errors could occur every three to nine days. If four service hosts were permitted to degrade to single-host service, then undetected service errors would occur approximately every month. (To get these quantitative results, realistic values were assumed for failure and repair rates. See [3] for details.) There is then a tradeoff to be considered. Suppose there are three hosts in the scheme. Is it better to cut off service for the 1/2 hour or so a day when two hosts are not available, or to take the chance of introducing an error once or twice a week? This is a management decision.

<u>Applications</u>. We feel that our report [3] contains the essential information required for implementing a resilient service. Future work needed in this area consists mainly of experimentation - to

see how well, in fact, the various schemes work - and of application to specific distributed system functions. These applications might include the following.

1. Synchronization. A two-host resilient scheme for handling synchronization primitives was sketched in [2]. Further work and experimentation would be useful.

2. Directories and data access. In a distributed environment the problem of accessing and updating network virtual file systems and their associated directories is difficult. For example, consider the problem of a single network-wide tree-structured file directory scheme. Each host on the network must be able to determine, in some reasonably transparent fashion, where individual files are stored. If each site in a large network is required to keep the entire directory structure, the cost for updates and synchronization of access to all of those directories (whenever they are updated) would clearly be prohibitive. It is relatively straightforward to use a scheme where the very highest levels of the directory structure are fixed and replicated on all hosts. Alterable directories and files are at lower levels of the tree. A list of potential service hosts is stored at the point where the hierarchy becomes variable. These service hosts are coordinated via a resiliency technique to provide access to files below that point. This approach has the advantage of partitioning the hierarchy in such a way as to minimize the number of hosts required to cooperate in an update.

3. Load sharing. Automated load sharing requires that multiple processors be controlled in a resilient and transparent fashion to provide processing services to requesting hosts. The resiliency technique can be applied in a straightforward way to coordinate the offering of that service. Any potential service site can receive a request for

service and pass it on to the primary for determination of an optimum processor for the work. Once the task has been successfully forwarded to the primary it would not matter if one of the service hosts involved in the task were to die. Adequate information would be maintained to support the automatic recovery of the service host.

Network File Allocation

A report [7] on this research area was written late in the contract year. The following summary of our work in this area is therefore largely abstracted from that document.

The problem. The file allocation problem is simply stated. If we want to set up a distributed data base in a network, where should we put copies of the various files? To make the problem precise, we must first define carefully the file usage process which we want to take into account in the decision on allocation. That is, we must develop a model for how files are expected to be used. The next step is to define what is meant by best (or optimal). In most studies in the literature, optimal is defined as least cost, where "cost" can be broadly defined to include more than dollars. That is, "costs" can be weight factors imposed by management to reflect, for example, the scarcity of certain resources. Alternatively, it might be worthwhile to minimize response time instead of cost.

Once we have a model, and have decided that (say) cost is to be minimized, the next step is to develop a cost formula, reflecting the model and including all essential parameters. Additional formulas may need to be developed for constraints, in case it is not realistic to assume that only cost need be taken into account.

Finally, the optimization problem that has been defined must be solved. In general, this is a straightforward, but very large, computational problem.

Models. We have identified three aspects of file handling that must be included in any file allocation study. Basically, the file must be

1. stored,
2. updated, and
3. queried.

The precise impact on cost of these basic processes will depend on how they are carried out. In the models which one usually sees discussed in the file allocation literature (see [8], for example) the cost formula is of the following form:

Cost = Cost of Storage

+ Cost of Sending Queries to Nearest Copy

+ Cost of Sending Updates to All Copies

Notice that as the number of copies increases, the cost of updates and storage will also increase. But the cost of querying decreases as closer sites become available to send the queries to. We therefore have a classic tradeoff, which makes the problem of finding the cheapest allocation a nontrivial one. In case the reader feels that not all relevant costs appear to be included in this formula, we remark that only a slight modification is needed for this basic scheme to reflect such additional features as the costs of processing queries and updates. Furthermore, parameters can be interpreted broadly. The "nearest" copy is not necessarily nearest in the geographical sense, but the one that can respond to the query most cheaply.

However, we identified a more serious difficulty with this scheme. The assumption is made that all sites generating updates send them independently to all the file copies. In our parallel work on synchronization and resiliency (see above), we identified many difficulties with such a decentralized model for updating. We believe that a primary-copy scheme, in which all updates are first sent to a "primary" has many advantages. The primary plays a role in ensuring both resiliency and proper synchronization of the updates. For a primary-copy scheme in which the primary subsequently broadcasts the updates, the cost formula will be something like the following:

Cost = Cost of Storage

+ Cost of Sending Queries to Nearest Copy

+ Cost of Sending Updates to Primary

+ Cost for Primary to Send Updates to Copies

Surprisingly, although this formula looks like it would probably lead to higher cost than the broadcast formula, this is not necessarily the case. That is, in many situations lower-cost file allocations can be obtained for a primary-copy scheme than for a broadcast scheme. Intuitively, this can occur when the primary is more centrally located than the sites generating the updates.

There are a number of constraints which one might want to impose on the problem. The four most useful ones that are discussed in the literature are:

1. Forbidden or prescribed sites. For historical or policy reasons, a copy of a data base might be reqiured at a given site. Conversely, security considerations could forbid certain locations.
2. Prescribed number of copies. Primarily for reasons of increased availability [4,9], it may be desirable to specify that there be, say, two copies of the data base, or of certain files. One might also specify a minimal number of copies.
3. Limited storage. It may be necessary to take into account the limited storage capacity of certain sites.

4. Upper bound on response time. If the access process is modeled in considerable detail, and if enough information on system parameters is known, the expected time to access each file may be computed. The constraint can be imposed that this time be less than some given bound.

Of these four constraints, the first two appear to be the most relevant to WWMCCS needs. Fortunately, they turn out to be very easy to impose. Straightforward conditions on number of copies or on file locations reduce the options and hence tend to reduce the computation necessary to find an optimum.

On the other hand, the third and fourth constraints actually tend to increase the difficulty of the problem. This is because these constraints couple together the impact of all the files in the system. Storage limitations obviously must take into account all the files. In response-time constraints, the file locations interact in more subtle ways, such as in increased network traffic and resulting delays. Without such coupling, the best location for each file (or group of files) can be determined independently. It turns out to be much easier to solve a number of small allocation problems than one gigantic one.

One further point should be made about storage constraints. It is not only difficult to impose them, but it may be unrealistic to do so. If it is otherwise worthwhile to store a file at a given site, there seems to be no good reason why additional storage capacity could not be added to accommodate that file. That is, "storage cost" in the cost formula can just as well reflect the cost of new storage capacity.

If one takes this point of view, then the only classical constraint which is troublesome to impose is a bound on response time. Further study may be useful to determine whether response-time constraints can be handled in some simple fashion - and also whether they are really worthwhile in most environments. It may be that some more straightforward constraint - for example, on network traffic between various sites - would be easier to apply and at the same time have the same effect on system performance as the response-time constraint.

Finding the optimal allocation. If there are n sites in a network, and for each site there are two possibilities - either it has a copy or it doesn't - then there are 2^n different file allocations to be considered. (This number becomes 2^n-1 if we omit the null allocation, i.e. no copy.) Computationally, there are no foolproof shortcuts to finding the optimal allocation. As the network increases in size, the work to solve the allocation problem will in general grow exponentially, doubling with every additional site. Consider the following example. Casey [8] notes that a program run on the 360/91 took less than 10 seconds to solve six different optimal allocation problems for a network of 19 sites. (This time included the Fortran compilation.) Thus, if we assume that it takes about 2 seconds to optimally allocate a single file in a 19-site network, it will take over an hour in a 30-site network, about 48 days in a 40-site network, and about 136 years in a 50-site network. These figures must then be multiplied by the number of files to be allocated.

It therefore becomes imperative to reduce the computational work in any way possible. There are some useful theorems reported in the literature, but it seemed to us that it was possible to make further progress in this area. We therefore developed three theorems which determine _a priori_ that certain sites should, or should not, be included in any optimal allocation. Each such _a priori_ determination reduces by a factor of two the size of the remaining allocation problem. We feel that our theorems can be enormously useful in reducing the file allocation problem for large networks down to manageable size.

Applications to special situations. Following another line of research, we investigated certain simple models and related allocation questions which may be particularly useful in the WWMCCS environment. First we assumed that network transmission cost is the same for all site pairs. However, we added the costs of processing queries and updates to the model, and we assumed that these may differ from site to site. Here are the questions we investigated, with summaries of our results.

Question 1. Suppose that all updates are generated at a single site, but queries may originate anywhere in the network. What is the best allocation?

> In a homogeneous network, the optimal allocation often turns out to be a single copy, located at the site generating the updates. Specifically, this happens when it costs more to maintain (store and update) a copy elsewhere than it does to answer remote queries from the updating site. In a heterogeneous network, more tradeoffs enter into the picture. We present two simple strategy rules for this case.

Question 2. Suppose that there are only two sites under consideration - a remote site holding a copy of the data base and responsible for updating it, and a local site generating queries for the data base. Is it worthwhile to have a local data cache?

> A simple inequality suffices to answer this question. If the savings per query, multiplied by the number of queries in a given time period, is more than the local storage cost of the data cache for that time period, then it pays to have the local cache. Otherwise it does not.

At this point we feel that the basic file allocation problem is well understood. Future work is likely to be more along the lines of applying the known principles to other special situations and particular network environments. A practical problem which must be addressed before real optimal allocations can be made is that of collecting the data on usage patterns and "costs" which go to make up the parameters in the formulas.

Optimization of Query Strategies

When distributed data management systems become a reality, many more options will be available to the user than exist in a simple single-site system. If multiple copies of data exist, there is a choice of sites from which to retrieve needed data. The different sites may have the data organized in different ways or may have different indexes. Some sites may be busier than others. In short, a particular query might be answered very rapidly (and cheaply) at one site but very slowly (and expensively) at another. Even without multiple copies, complex queries may require combining data from several sites, with consequent transmission of large amounts of data across the network. How much data need be shipped depends upon the strategy for handling the query. One could hardly expect the individual user to have enough information (or time) to decide on a strategy for each of his queries. The system must be able to make the strategy decisions. One would hope that then this decision could be made optimal, or near optimal. However, the development of algorithms to find the best strategy in various situations is a difficult research topic. Our work to date has been only preliminary. The conclusions drawn -and even the approaches taken - should be considered tentative. We present only a brief summary of the work here. Since it was carried out in conjunction with the experimental system design effort, a more thorough account of the work may be found in the Experimental System Report, which is being prepared concurrently.

Models for query strategies. In simple cases, models for handling queries can be formulated in much the same way as those for file allocation. In the latter case, the question is whether or not a file should be placed at a certain site. Relevant costs can be computed and the total costs of alternate allocation strategies compared. In the case of modeling query strategies, the question is whether or not a given data operation should be carried out at a certain site. Again, associated costs can be computed and total costs of rival strategies compared.

One model that we have developed for studying query strategies contains the following basic parameters:

1. The cost C_{ij} of performing a given operation (i) at a specific site (j) which holds a copy of the relevant data. Costs may vary from site to site because of pricing policies, differences in system software or hardware, or differences in how the data are stored or indexed.
2. The amount A_i of output from a given operation (i). The decision on where to carry out an operation can be affected by whether or not this output must be transported over the network. For example, suppose sites 1 and 2 both have copies of the relevant files, and the results of the request are needed at site 1. Then, even if site 2 can respond to the request (i.e., carry out the necessary operations) more cheaply than can site 1, the cost of shipping the results (which may be a large set of records) from site 2 to site 1 may make total costs lower if the operations are performed at site 1.

Responding to a query may involve a sequence of operations (perhaps with some parallelism) to be carried out on different files. We have found that the order in which these operations are performed can have a considerable effect on total cost, but the problems which arise in trying to optimize over all possible orderings of operations are enormous.

Taking the order of operations as given, a formulation of the cost optimization problem as a zero-one programming problem has been developed. The variables Y_{ij} to be determined (and that take on only the values one or zero) indicate whether or not a given operation (i) is performed at a given site (j). The total cost formula consists of two terms:

1. the cost of carrying out the operations, i.e.,

   $\sum_{i,j} C_{ij} Y_{ij}$, and

2. the cost of the network traffic incurred; i.e., the sum over all operations i of the cost of transporting A_i if the operation is carried out at one site and the result is then needed at another.

If there are m operations to be carried out in fulfilling a request and n sites (on the average) at which each may be carried out, the number of variables Y_{ij} is mn. Hence there is likely to be a rather large programming problem required to optimize the handling of each query. Clearly, solving large programming problems in this setting is much more impractical than it is in file allocation, where allocations are optimized only infrequently. There is too much overhead involved even in heuristic methods for finding near-optimal solutions. Furthermore, an enormous amount of *a priori* information on database content is assumed in the presumption that one knows the parameters A_i - and even the costs C_{ij}. That is, unless there is some simple way to generate good guesses for these parameters, it would appear that optimizing retrievals in this way requires more work to get the parameters than the actual retrieval (even if done inefficiently) should take.

Application of sampling theory. In an attempt to solve the problem of obtaining valid estimates for the parameters A_i, we have looked into using the techniques of statistical sampling for that purpose. In order not to confuse our results by including too many extraneous factors, we considered only one small aspect of the overall strategy question. Suppose we have a simple query involving two files at two different hosts. Suppose that, to respond to the query, a subset of the records in each file is first selected. To make this precise, suppose that all of the records satisfying property "r" are selected from file F_1 at site 1, and all those satisfying property "q" are selected from file F_2 at site 2. Further suppose that these subsets must then be combined in some way. An example of such a query is the following. Suppose site 1 contains a personnel file F_1 of air force officers and their detailed job qualifications. Suppose site 2 contains a general military personnel file F_2 with information on such things as eligibility for retirement. The search for an officer for a certain command post might involve selecting those with the appropriate job qualifications from F_1, selecting those from F_2 who are not scheduled for imminent retirement, and then finding, by a combining operation, the officers common to both subsets.

To carry out the combining operation, one of the subsets must be shipped to the site holding the other. The question is, which one? We assume that the decision should be made strictly on the basis of which subset is smaller - that is, we want to ship the smaller set of records. In a simple case like this, the system manager might just ask each site to find the subset and report its size. Then the manager could determine which subset to ship. However, for more complex problems this sort of step-by-step monitoring may not be practical; an _a priori_ decision may have to be made. The results from investigating how well _a priori_ decision tools work out in this simple case should carry over to the more complex cases where they are really needed.

Our proposed decision tool is the following. Let $\hat{F}_i$ be a randomly chosen sample of records from file F_i. Find (by searching if necessary) the fraction of the records from $\hat{F}_1$ satisfying the given property "r" (denote this by p_{r1}), and similarly the fraction of the records from $\hat{F}_2$ satisfying property "q" (p_{q2}). Let S_1, S_2 be the number of records in F_1, F_2, respectively. (If the records in files 1 and 2 are of different lengths, it might be better to define S_1, S_2 as general file sizes (e.g. numbers of bytes).) If the inequality

$$S_1 p_{r1} \leq S_2 p_{q2} \qquad (1)$$

holds, we decide _a priori_ that, after the selections from the complete files have been made, the subset from site 1 will be shipped to site 2. That is, we hope that it is also true for the complete files that

$$S_1 P_{r1} \leq S_2 P_{q2}, \qquad (2)$$

where P_{r1}, P_{q2} represent the true fraction of the entire files having properties r, q, respectively.

We find that there are a number of ways in which statistical sampling theory aids in developing and understanding this approach. First, a standard formula relates the sample fraction p having a certain property to the fraction P of the complete population having that property. Parameters in the formula include the desired precision d ($= |P-p|$), the confidence level (i.e. with what probability $|P-p|$ is less than d) and the sizes of both sample and parent population. Using this formula, one can readily determine how large a sample is needed to give a desired precision (with reasonable confidence). It turns out (luckily for the political pollsters) that rather small samples give high confidence levels. For example, suppose we find that $p=0.100$ for a sample of 500 records out of 10^5. Then the probability is 0.95 that P is really between .074 and 0.126.

One might question whether the assumptions underlying statistical sampling theory are valid for properties of records in a file. We therefore tried the sampling technique on a test data base. The data base contained 10^4 records with information on fuel storage at military sites. Twenty-two queries were run against the full data base and against a sample of 100 records. In all cases, the sample predicted the actual query performance well within the theoretical limits.

Another problem that needed to be addressed is to determine how likely it is that the correct decision is made. Notice that this is a tricky problem. Even though our estimates are bad, as long as the inequalities (1) and (2) both hold (or both fail to hold) we make the right decision. Making standard probabilistic assumptions, we are able to compute confidence levels for the decision as a function of expected absolute difference in subset sizes:

$$D = |S_1 p_{r1} - S_2 p_{q2}|.$$

As this difference increases, the probability that we make the right decision rapidly approaches one. (Sample sizes, etc., enter in here too, of course.)

Finally, we have looked briefly into the question of how costly it is (on the average) to make the wrong decision. That is, how much extra data, on the average, will we have to ship when we make the wrong decision? The answer is that very little need be shipped. Intuitively, we obtain this result because we are most likely to make the wrong decision when the difference in subset sizes is very small. The larger the true difference the less likely we are to mis-estimate so badly that inequality (1) gives the wrong answer.

In summary, it appears that sampling will allow us to make the low level decision discussed here with good accuracy. Further investigation is required to determine if this technique can be applied to more complex operations. The good results of this preliminary study do, however, make us very optimistic of future progress in the optimization of query strategies.

References

1. Alsberg, P.A. et al. Preliminary Research Study Report. CAC Document No. 162 (JTSA Document No. 5509), Center for Advanced Computation, University of Illinois at Urbana-Champaign, May 1975.

2. Alsberg, P.A. et al. Synchronization and Deadlock. CAC Document No. 185 (CCTC-WAD Document No. 6503), Center for Advanced Computation, University of Illinois at Urbana-Champaign, March 1976.

3. Alsberg, P.A., Belford, G.G., Day, J.D., and Grapa, E. Multi-Copy Resiliency Techniques. CAC Document No. 202 (CCTC-WAD Document No. 6505), Center for Advanced Computation, University of Illinois at Urbana-Champaign, May 1976.

4. Belford, G.G., Schwartz, P.M., and Sluizer, S. The Effect of Backup Strategy on Data Base Availability. CAC Document No. 181 (CCTC-WAD Document No. 6501), Center for Advanced Computation, University of Illinois at Urbana-Champaign, February 1976.

5. Belford, G.G., Day, J.D., Sluizer, S., and Willcox, D.A. Initial Mathematical Model Report. CAC Document No. 169, (JTSA Document No. 5511), Center for Advanced Computation, University of Illinois at Urbana-Champaign, August 1975.

6. Belford, G.G. and Day, J.D. A Cost Model for Data Distribution. CAC Document No. 179 (JTSA Document No. 5514), Center for Advanced Computation, University of Illinois at Urbana-Champaign, November 1975.

7. Belford, G.G., Day, J.D., Grapa, E., and Schwartz, P.M. Network File Allocation. CAC Document No. 203 (CCTC-WAD Document No. 6506), Center for Advanced Computation, University of Illinois at Urbana-Champaign, August 1976.

8. Casey, R.G. Allocation of copies of a file in an information network. AFIPS Conference Proceedings 40, AFIPS Press, Montvale, N.J., 1972, pp. 617-625.

9. Chu, W.W. Optimal file allocation in a computer network. In <u>Computer-Communications Networks</u>, N. Abramson and F. Kuo (eds.), Prentice-Hall, Englewood Cliffs, N.J., 1973.

10. Johnson, P.R. and Beeler, M. Notes on Distributed Data Bases. Draft report available from the authors (Bolt, Beranek, and Newman, Inc., Cambridge, MA.)

11. Johnson, P.R. and Thomas, R.H. The Maintenance of Duplicate Databases. RFC #677, NIC #31507, January 1975. (Available from ARPA Network Information Center, Stanford Research Institute-Augmentation Research Center, Menlo Park, CA.)

12. Lum, V.Y., Senko, M.E., Wang, C.P., and Ling, H. A Cost Oriented Algorithm for Data Set Allocation in Storage Hierarchies, CACM <u>18</u>, pp. 318-322.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CAC Document Number 210 CCTC-WAD Document Number 6508	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Research in Network Data Management and Resource Sharing - Final Research Report		5. TYPE OF REPORT & PERIOD COVERED Research
		6. PERFORMING ORG. REPORT NUMBER CAC #210
7. AUTHOR(s)		8. CONTRACT OR GRANT NUMBER(s) DCA100-75-C-0021
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Advanced Computation University of Illinois at Urbana-Champaign Urbana, Illinois 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Command and Control Technical Center WWMCCS ADP Directorate, 11440 Isaac Newton Sq., N Reston, VA 22090		12. REPORT DATE September 30, 1976
		13. NUMBER OF PAGES 55
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Copies may be obtained from the
National Technical Information Service
Springfield, VA 22151

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

No restriction on distribution

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed data bases
Network resource sharing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report summarizes progress made and results obtained during the preceding fiscal year. Specific research areas investigated included deadlock, synchronization, techniques to ensure resiliency in distributed operations, optimal allocation of files in a network, and optimization of query strategies.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUC-CAC-DN-76-210	2.	3. Recipient's Accession No.
4. Title and Subtitle Research in Network Data Management and Resource Sharing - Final Research Report			5. Report Date September 30, 1976
			6.
7. Author(s)			8. Performing Organization Rept. No. CAC #210
9. Performing Organization Name and Address Center for Advanced Computation University of Illinois at Urbana-Champaign Urbana, Illinois 61801			10. Project/Task/Work Unit No.
			11. Contract/Grant No. DCA100-75-C-0021
12. Sponsoring Organization Name and Address Command and Control Technical Center WWMCCS ADP Directorate 11440 Isaac Newton Square, N. Reston, Virginia 22090			13. Type of Report & Period Covered Research
			14.

15. Supplementary Notes

None

16. Abstracts

This report summarizes progress made and results obtained during the preceding fiscal year. Specific research areas investigated included deadlock, synchronization, techniques to ensure resiliency in distributed operations, optimal allocation of files in a network, and optimization of query strategies.

17. Key Words and Document Analysis. 17a. Descriptors

Distributed data bases
Network resource sharing

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement No restriction on distribution Available from the National Technical Information Service, Springfield, Virginia 22151	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 55
	20. Security Class (This Page UNCLASSIFIED	22. Price