DISTRIBUTED COMPUTING RESEARCH AT PRINCETON - 1985

Robert Abbott, Rafael Alonso, Daniel Barbara, Ricardo Cordon, Hector Garcia-Molina, Philip Goldman, Mark Karol, Jack Kent, Boris Kogan, Frank Pittelli, Peter Potrebic, Stuart Scheartz, Patricia Simpson, Annemarie Spauster, Michael Sotessl, Sergio Verdu

Department of Computer Science &
Department of Electrical Engineering
Princeton University
Princeton, NJ 08544

CS-TR-029-86

DISTRIBUTED COMPUTING RESEARCH AT PRINCETON — 1985

Robert Abbott, Rafael Alonso, Daniel Barbara, Ricardo Cordon, Hector Garcia-Molina, Philip Goldman, Mark Karol, Jack Kent, Boris Kogan, Frank Pittelli, Peter Potrebic, Stuart Schwartz, Patricia Simpson, Annemarie Spauster, Michael Stoessl, Sergio Verdu

Department of Computer Science and
Department of Electrical Engineering
Princeton University
Princeton, N.J. 08544

1. Introduction

In this note we briefly summarize the distributed computing research we performed in the year 1985. In general terms, our emphasis was on studying and implementing mechanisms for *efficient and reliable* computing and data management. Our work can be roughly divided into nine categories: mutual exclusion, dynamic vote reassignment, the implementation of a highly reliable database system, the implementation of a high availability database system, a survey of reliable distributed data management, the optimization of shadow recovery mechanisms, load balancing, caching and prefetching in information systems, and protocols for multiple-access packet-broadcast channels.

Due to space limitations, we concentrate on describing our own work and we do not survey the work of other researchers in the field. For survey information and references, we refer readers to some of our reports.

2. Mutual Exclusion

A number of distributed algorithms require that at most one connected group of nodes be active at a time, and this is achieved by assigning each node a number of *votes*. The group that has a majority of the total number of votes knows that no other group can have a majority, and can thus be active. (It is also possible that no group has a majority and is active.) Such a mechanism can be used, for example, to manage a file that is replicated at various sites. Voting ensures that at most one group can update the file at a time, avoiding potential conflicts.

The way these votes are dispersed among the nodes can affect in a critical fashion the reliability of the system. To illustrate what we mean, consider a system with nodes a, b, c, and d. Suppose that we assign each node a single

This work has been supported by NSF Grants ECS-8303146, DMC-8505194, and ECS-8351616, Naval Electronic Systems Command contract N00039-84-C0485, New Jersey Governor's Commission on Science and Technology Contract 85-990660-6, and grants from DEC, IBM, NCR, and Perkin-Elmer corporations.

vote. This means that, for example, nodes a, b, and c could operate as a group (3 is a majority of 4), but a and b by themselves could not.

Next, consider a different distribution where a gets two votes and the rest of the nodes get one vote. Call this distribution Q and the previous one P. It turns out that Q is superior to P because any group of nodes that can operate under P can operate under Q, but not vice-versa. For example, a and b can form a group under Q (3 is a majority of 5) but not under P. Thus, if the system splits into group $\{a,b\}$ and group $\{c,d\}$, there will be one active group under Q but no active group if P is used. So clearly, no system designer should ever use distribution P, even though it seems a very "natural" choice.

In the past we have studied vote assignments, developing a theory for their enumeration and evaluation [Barb85]. More recently, we studied other problems that are closely related. Specifically, notice that the formulation of the problem given at the beginning of this section involves a single operation or *mission* (e.g., updating a file). In other cases, there is more than one type of competing mission. For example, one mission can be to commit a pending transaction, while another mission might be to abort the same transaction. The rules for mutual exclusion are different from the single mission case. A commit mission does not want to exclude other commit missions from executing in other groups, and similarly, aborts do not conflict with each other. However, commits and aborts are mutually exclusive: we do not want to have a situation where some groups decide to abort a transaction while other groups decide to commit it.

We have studied various types of multi-mission mutual exclusion scenarios [Barb86a], again developing a theory for them. Modified voting mechanisms can be used to enforce the new mutual exclusion rules [Skee82]. In the commit and abort example, we can require that a group have A or more votes before it aborts a transaction, and that it have C or more votes to commit. If $A + C$ is greater than the total number of votes, then mutual exclusion between aborters and committers is guaranteed.

Just as in the single mission scenario, the vote assignment plays an important role in system availability, i.e., in the likelihood that a transaction could be committed or aborted. And as in the single mission case, there are non-voting mechanisms that may be superior. For example, suppose we define the sets

$$A = \{\{a,b\},\{c,d\}\}$$
$$C = \{\{a,d\},\{b,c\},\{a,c\},\{b,d\}\}.$$

Our rules are that a group can abort a transaction if it or a superset appears in A, and it can commit if the same holds for C. For example, if nodes $\{a,b,c\}$ form a group, then they can either abort or commit a transaction. Group $\{a,b\}$, on the other hand, can only abort transactions.

Notice that sets A and C enforce the mutual exclusion rules we need for commits and aborts, and yet, cannot be implemented by the voting technique we outlined earlier. Hence, the set mechanism could be superior in some cases. In [Barb86a] we have studied such sets, their properties, and techniques for selecting good ones.

3. Dynamic Vote Reassignment

Even though voting mechanisms may not be as "powerful" as set oriented ones, they do have an important advantage: they are very simple. In particular, not only can we select a good assignment at system initialization time, but we can also dynamically change the assignment as failures and repairs occur. This allows the system to adjust to the current topology, reducing the probability of encountering a state where no group can perform a mission. We suspect that such an adaptive strategy would be harder with a set oriented mechanism.

To illustrate, consider a system with four nodes a, b, c and d that is executing a single type of mutually exclusive mission. Say we have initially assigned the following votes: $v_a = v_b = v_c = 1$ and $v_d = 2$, where v_i represents the votes assigned to node i. Assume that a partition separates node d from nodes a, b, and c. Nodes a, b, and c can still collect a majority of votes, while d cannot. However, if a second partition occurs, separating node c from a and b, the system will be *halted*, i.e., no group will have a majority and no group will be able to execute a mission. However, we can reduce the likelihood of halting if we increase the votes of group $\{a,b,c\}$ *before* the second partition occurs. That is, after any failure, the majority group (if any) dynamically reassigns the votes in order to increase its voting power and increase the system's chances of surviving subsequent failures.

In our example, nodes a, b, and c may opt for reconfiguring the votes during the first partition. For instance, a new vote assignment could be $v_a = v_b = v_c = 5$. Node d is unaware of the change and remains with $v_d = 2$ votes. (As a matter of fact, since d is not in a majority group, it *cannot* change its votes.) In this way, the second partition will find nodes a and b with 10 votes out of a total of 17, forming a majority group that can continue to execute its mission. After the second partition, the new majority group $\{a,b\}$ could reassign itself new votes of $v_a = 15$ and $v_b = 5$ in order to tolerate even a third partition. When the partitions are repaired, the nodes that have proportionately less votes (e.g., d) can attempt to increase their votes. Equivalently, nodes that have increased their votes may decrease them to recapture the original assignment.

Notice that in our approach, nodes operate quite autonomously, without requiring group consensus. Group consensus may select a better assignment but requires tighter coordination among the nodes (e.g., a central coordinator may have to be selected). On the other hand, autonomous vote changes are much simpler and more flexible. Each node decides independently what its new vote value should be. The node does not need complete or accurate information about the state of the system. In a sense, the node makes an educated guess about the best number of votes to have, with its primary goal being to claim for itself all or part of the voting power of a node (or nodes) that have been separated from it.

There are two problems to solve in implementing the ideas we have sketched. The first involves the selection of a new vote value by a node that has detected the disconnection of another node. We refer to the mechanism for such a selection as the *policy.* The policy takes into account the current state of the system (who is up, who is down, who has how many votes), as determined (perhaps not accurately) by the node making the change, and outputs the desired votes for this node. There are a number of different rules that can be used for the policy and these are surveyed in [Barb86b]. The efficiency of the various policies is compared through a detailed simulation that evaluates the probability of arriving at a halted state. The autonomous policies are also compared to a group consensus strategy. We have found that the better autonomous policies provide almost as much availability as the consensus method, even though they make a less accurate reassignment relative to the new topology. Since autonomous policies are simpler to implement, we are encouraged by these results.

Once a node chooses a new vote value, it cannot vote with it right away. The second problem is to ensure that the node is part of a majority and authorized to make the change. The mechanism to do this is called the *protocol.* The basic idea is to collect acknowledgements from nodes with a majority of votes, and in the process inform them of the change. In doing so, we must make sure the protocol does not get confused between "old" and "new" votes. We have studied one such protocol and have shown that it operates correctly [Barb86c]. The protocol is efficient and relatively simple.

4. A Highly Reliable Database System

One of the goals of our research is to understand reliable data management, so a natural question to ask is: how reliable can we ever hope to build a data management system, and at what cost? In an attempt to answer this question, we have embarked on the actual implementation of an ultra high reliability system.

The key to our system is a failure model that captures the worst possible behavior by a computer. In this model we have a collection of $2n + 1$ computers, and we assume that at most n of them fail. We make no assumptions about how a computer fails, and in this sense, the model is the most general possible. We call a failed node *insane* since it can send any message out, including misleading ones, it can refuse to send required messages, and it can even collaborate with other insane nodes in an attempt to subvert the entire system. We assume that an insane failure is eventually detected and repaired.

If we wish to reliably perform a task in this environment we must execute the task on all $2n + 1$ computers and look at all outputs. The output that $n + 1$ or more nodes agree on must be the correct one. This approach is well known and is called N-modular redundancy (NMR). It has been used successfully at the hardware level for many years. In our system, however, we are using NMR at a different level, the database level.

There are a number of advantages to operating at this level. The communication overhead can be greatly reduced. Instead of exchanging and comparing the result of, say, every addition or memory reference, the system operates at a much higher level, mainly comparing the outputs of user transactions. Since remote sites in essence have backup copies of the database, processing at each site may be streamlined, eliminating local logging and dumping, for example. Actually, the database processing at each site is quite simple. Note that nodes execute each transaction independently from the rest of the system. Locks do not have to be requested from other nodes, and there can be no global deadlocks. In addition, our system is built with off-the-shelf equipment, as opposed to the specialized hardware usually required by NMR.

So in light of these advantages, and of the extra reliability provided, the cost of database NMR may not be as high as one may initially imagine. It will never be cheap, but it may be a desirable alternative for critical applications where human lives or money is at stake, or where users are simply tired of dealing with "temperamental" computers that lose or destroy their data.

Database NMR has not been implemented before (as far as we know), and there are a number of challenging problems to be solved. Typically, each node of a NMR system contains very little "state" information, so recovery after a failure is simple. However, in our case the "state" is the entire database. Bringing it up to date after a failure has been detected involves copying a snapshot of the database from remote nodes, followed by execution of any transactions missed during the snapshot copy operation. Since we cannot allow a failed node to halt the entire system, the recovery has to be performed without stopping the nodes that have not failed. It may also be possible to identify the portions of the database that were corrupted during the failure, in order to avoid copying the entire database.

There is also the problem of distributing incoming transactions to the nodes for execution. For the NMR scheme to work, not only do all nodes have to execute exactly the same transactions, but they have to execute them in the same order. This can be achieved with a so-called Byzantine Agreement protocol [Garc84]. Even though these algorithms have been studied extensively in the literature, they have seldom been implemented (in addition to own own implementation, we know of only one other one, at IBM Almaden Research Laboratories).

Our highly reliable database system has been implemented on a network of SUN and VAX workstations and is currently undergoing extensive evaluation. The structure of the system, as well as some preliminary results are presented in [Pitt85]. Specifically, a 3 node, fully redundant system is compared against a 2 and 1 node system. The smaller systems have reduced hardware and overhead, but of course, provide less protection against failures. Thus, our comparison gives us a chance to evaluate the cost of various degrees of reliability.

In another report [Pitt86] we study the tradeoffs involved in scheduling transactions. In particular, several options like batching together transactions and using null transactions to speed up agreement are studied and evaluated in detail.

5. A High Availability Database System

There are two important aspects to reliability: correctness and availability. The objective of the project described in the previous section is mainly correctness of the database operations, even if there is a cost in terms of availability. For instance, if one computer (or even a group of n computers) is cut-off from the rest of the system, it will be unable to process transactions because this could compromise the correctness of the results and of the database. On the other hand, if availability were the main objective, then it may be desirable to continue processing at the isolated node. After all, it does have a copy of the database and is capable of processing transactions. Of course, we probably do not want to compromise correctness "too much," i.e., we may only be willing to tolerate certain types of inconsistencies between the isolated copy and the rest of the system.

There appears to be a spectrum of alternatives for reliability. At one end are systems that stress correctness (like the one we discussed in Section 4), and at the other end we have both implemented a simple system for very high availability and commenced the design of an improved system. Such systems could be used in applications where data availability is paramount and halting transactions at some nodes may not be acceptable. Typically, in these applications either network partitions are relatively common or they occur at critical moments when access to data is imperative.

The hardest problem in a high availability system is coping with the data inconsistencies that arise when nodes that cannot communicate properly continue to execute transactions. Drawing the line between the desired availability and the undesired inconsistencies is usually hard.

A number of solutions to the problem have been suggested [Davi86]. The approach is generally to allow transactions to execute anywhere in the system at any time. When inconsistencies are detected, then either conflicting transactions are backed out (e.g., if two meetings were scheduled at the same time, one is canceled and the organizer is notified), compensating transactions are run (e.g., if a customer overdrafts account, temporarily close it and send dunning letter), or differing values can be integrated in an ad-hoc fashion (e.g., if one system reports a ship to be at position X and another at position Y, then install the latest value in all copies).

We have implemented a simple system (on a network of SUN and VAX computers) that allows execution of transactions at all times and then integrates inconsistencies using simple rules provided by the database administrator. The approach we used is called *datapatch* and has been described in an earlier report [Garc83]. The system we built is described in detail in [Stoe85].

Although such a solution can be useful in some applications, we have discovered two important drawbacks. First, solutions like datapatch and others do not generalize easily. As the application becomes more complicated, it becomes more difficult to handle inconsistencies. Second, it is difficult to determine what *general* properties the system guarantees. Conventional concurrency control mechanisms guarantee serializability; what do the new mechanisms guarantee instead?

In the design of a new high availability system we have tried to avoid these problems by making two simple but important observations:

(a) Not all users of (or transactions in) the system have the same availability requirements. A bank customer may want to be able to deposit and withdraw funds at any branch any time, but there is no reason why interest payments have to be executed at any branch at any time.

(b) For high availability, proper design of the application data structures and transactions is crucial. If the application is not structured properly, then the detection and integration of inconsistencies can become difficult. Thus, not only do we aim to build a system, but we also want to develop a methodology for designing the application.

The basic idea in our approach is quite simple. The database is divided into *fragments*, each one having an *agent* that controls the execution of transactions affecting it. Fragments can be replicated at various nodes. An agent is like a central concurrency control mechanism for the fragment, authorizing all updates to the fragment. However, there are two important differences. First, an agent may be associated with a physical person or object and can move from node to node, including nodes that temporarily cannot communicate (electronically). This way, a user that requires access to a fragment can be made its agent, therefore getting access whenever he needs it. Second, if a transaction needs to read fragments it does not modify, it can read them at any copy. Because of communications failures, such copies can be out of date. Hence, the agents do not guarantee serializability. However, agents do provide control that is crucial in resolving conflicts. Specifically, if we ever encounter two versions for the same data item, it is easy to resolve the conflict: the one its agent produced later is the correct one.

In an airline reservation example, each customer can have his or her own fragment representing the requests for reservations that have been made. Since the customer is the agent, he can modify his request no matter what node he is communicating with. At a central site, requests for reservations are read and generate real reservations, stored at a different fragment. The agent for this fragment is at the central site. For each flight there is a fragment representing the seat assignments for it. The agent for these fagments is located at the airport where the flight originates. There transactions read the request and reservations fragments and decide who actually gets on the plane. Priority is given to customers with reservations, but if there is space, then customers that requested a reservation but have not gotten one (maybe because the central site is down), are considered.

The coupling between fragments is not tight, and this is what gives the higher availability. We can think of each fragment as a separate database. Transactions can read other databases, but that data is viewed simply as input data to the transaction. Within each fragment, however, serializability is guaranteed. Hence, we call this *fragment-wise* serializability.

Fragments and agents are very simple ideas, but we believe they provide the right mix of control and flexibility for high availability applications. Furthermore, they provide a spectrum of interesting solutions. With a few additional restrictions, fragment-wise serializability can become full serializability, without sacrificing availability entirely. In the other direction, we can relax the rules for agent migration and obtain a property weaker than fragment-wise serializability.

We are currently studying and designing a system based on the notions of fragments and agents. A preliminary discussion can be found in [Koga86].

6. Survey of Reliable Distributed Database Management

We have been invited to contribute a survey paper to a upcoming special issue of the IEEE Proceedings on Distributed Databases. We decided this was a good opportunity to summarize and understand current developments in one of our areas of interest: reliable data management.

Our survey [Garc86b] assumes little prior knowledge. We start by reviewing some of the basic concepts of centralized data management such as consistency, serializability, and logging. We also outline the most common failure models for distributed systems. Throughout the paper, we assume that network partitions do not occur. (An earlier paper of ours surveyed protocols for partitioned systems [Davi86].)

The key to reliability is the replication of resources. Hence, the rest of the survey is divided into three areas, according to what is replicated. The simplest case occurs when no data is replicated. Although data availability is low is this case, the system must still ensure that transactions are executed correctly, i.e., as atomic actions. Therefore, we study how the transaction processing strategies of a centralized system can be extended to a distributed one and made robust.

The next case is data replication. The basic protocols of the previous case can be used for replicated data, but they do not exploit the potential for high availability. (For example, a transaction may still have to lock at all copies before updating. If one copy is down, the transaction must halt.) So, in this section we study how higher availability can be achieved without sacrificing serializability. Incidentally, in another invited paper [Garc86a] we discuss the general tradeoffs involved in data replication. We also present some strategies that may make replication less costly in the future.

Finally, we consider full data and transaction replication. This yields systems like our own NMR Highly Reliable Database System (Section 4). In this last section we study the principles behind such systems.

7. Optimizing Shadow Recovery

Many of the distributed mechanisms we are interested in require a reliable database store at one or more of the nodes. Thus, in conjunction of our other efforts we have been studying strategies for building *efficient and reliable* single node data storage facilities.

Earlier we constructed an experimental testbed system for implementing and evaluating various storage mechanisms [Kent84]. Out of these studies came a new idea for significantly improving the performance of one of the most common mechanisms: shadow recovery. This idea is presented in detail in [Kent85a], where we also evaluate it analytically.

In a conventional shadow mechanism, the database (or any file) is broken into disk pages. The page table contains pointers to all these pages, and all database accesses are through the page table. When a transaction wishes to modify one or more pages, it does not change the existing copies. Instead, it creates new versions that are private to the transaction. When the transaction completes, it atomically updates the page table so that it points to the pages just created.

Shadows are a useful mechanism, and as a matter of fact, our experiments showed that shadows outperformed logging (another popular strategy) in cases where transactions updated large numbers of contiguous pages. However, in many other cases logging performed better. We observed that the principal reason why shadows were not performing as well was excessive page table IO. That is, the page table is usually too large to fit in main memory. Thus, to access a database page we may have to read into memory parts of the page table (requiring one or more disk block reads).

Our new shadow recovery technique effectively "shrinks" the size of the page table, while keeping the size of the database fixed. Unlike other schemes, this is achieved *without* compromising disk space utilization. Moreover, our mechanism can be customized for a desired disk utilization and application. We believe that this mechanism, with its added flexibility and improved performance, extends the range of applications where shadowing is the prefered alternative for crash recovery.

Very briefly, the basic idea of our mechanism is to have two page tables, the primary and the secondary one. The primary one is small enough so that most of it fits in memory. Since the number of mapping bits per database page is limited in the primary table, each logical database page can be mapped to only a small subset of locations on disk. If the disk is not too full, then chances are that the database page will indeed be in one of these locations. In this case, the page can be retrieved directly, without extra IO cost.

If the mapping from logical database page to physical location cannot be done with the primary table, then the secondary one must be accessed. It is similar to a conventional page table, and there will be an IO cost associated with accessing it. However, if the system is designed properly, this should not occur often. Thus, the primary table acts like a filter, performing a majority of the mappings very cheaply.

To support our claims, we have developed a probabilistic model to study the location of logical database pages on disk and the requests made by transactions [Kent85a]. Our results indicate that the IO overhead can be significantly reduced, while keeping memory size and disk utilization at reasonable levels.

8. Load Balancing

Many of the currently existing computing environments consist of a heterogeneous collection of of workstations and mainframes connected by a high bandwidth local area network (LAN). One of the main benefits of working in such an environment is being able to share scarce resources with other users of the network; but one resource that is often not shared is the processing capacity of the network nodes. In many systems, the scheduling of user jobs is individually carried out by each processor, and the computations of the users logged on at any one machine are performed locally. The decentralization of CPU management, coupled with large differences in the numbers (and types) of users connected to each of the nodes in the network, often can lead to situations where there are great disparities in load among the machines on the network. For example, at our local computer center, as the due date for a class assignment approaches, the processor assigned to the students in that class becomes heavily loaded, while other machines are underutilized. Although users can determine by themselves that an imbalance exists, and remotely log onto another computer, we feel that, in order to prevent a chaotic situation and to aid naive users, it is best to develop strategies that can solve the load balancing problem in an automatic way, much in the way that users now depend on virtual memory techniques to manage their memory space. We are currently exploring the implementation of such load balancing strategies for LAN environments.

Our initial work consisted of designing and implementing a prototype load balancing mechanism [Alon86]. Our measurements show that, even under conditions of relatively small load imbalance, sizable performance gains can be achieved. For example, we have observed improvements of as much as 70 percent in the turnaround time of a simple compilation task. The overhead involved in running our system is fairly small (for both users and non-users of our mechanism), and our implementation involved no changes to the underlying system software. Furthermore, our performance improvements were obtained with a naive load metric (load was defined as the average number of jobs in the ready queue) and with a simple-minded decision policy (always send jobs to the least loaded machine).

Our prototype currently runs in a laboratory facility composed of 7 Sun workstations of various types and configurations. All the processors are connected by a stand-alone 10 Mbit/s Ethernet. The operating system for the workstations is Sun UNIX 4.2 Release 2.0, which supports *NFS*, a network file system.

The software that we have developed for our prototype implementation consists primarily of a shell that executes commands either locally or at a remote site (the site selection depends on the load balancing strategy being used); this shell is aided in its task by ancillary daemons which communicate load information across machines and facilitate remote execution of processes. If the command is to be run locally, the shell interprets it in the usual way. However, if a remote host is chosen (and the connection to that host is successful), the system transfers the command line (as well as other relevant information) to the remote site. At this point, the shell again has to coordinate with the remote daemons, in order to to ensure that I/O flows between the command process and the user, and to clean up after the command terminates. It should be noted that all the operations just described are transparent to the user; from the perspective of a user employing our shell, most commands will proceed to execute as if they were being executed by one of the standard UNIX shells. In particular, commands can be combined by the use of pipes, have their input and output redirected, and access environment variables.

Our current work involves a systematic study of different load balancing strategies, which involves exploring the performance of various combinations of load metrics and decision policies. The selection of a load metric requires a careful definition of what is meant by the load of a processor. It seems clear that load should be defined, at least partially, in terms of a set of performance indices (such as CPU utilization or mean number of I/O requests), but it is less clear that two different processes should use the same definition of load; for example, an I/O intensive job will probably perceive load in a different way than a CPU-bound job. And since load metric information must be broadcast, the rate at which such broadcasts are made also needs to be studied. In selecting a decision policy, questions such as the stability of distributed decision policies, the behavior of the algorithms under old or incomplete load information, the cost of making the decision, and so on, are of crucial interest. Lastly, we are also interested in determining the impact of system topology and network bandwidth on the different load balancing strategies.

9. Caching and Prefetching in Information Systems

During the last few years, a number of companies have begun to offer a variety of public information services. These services typically consists of providing access to a large database containing information useful to a sizable number of clients, who pay for the ability of using the database. The database designers usually assume that users will employ a terminal and a modem in order to access the service through the phone network. However, with the increase in the numbers of personal computers owned by the general public, many clients now contact the database by using a workstation; yet, most information systems still assume the same terminal-based model of user access, neglecting to utilize the added functionality that a computer offers over a terminal.

Our work in this area consists of exploring how best to utilize those added capabilities, in order to integrate more closely the user's computer (typically a personal computer with limited storage capacity) and the information services' computational resources (usually a large mainframe with a large secondary storage). In particular, we are investigating the performance improvements possible by using the storage and processing capabilities of the user workstations in order to cache at the workstations the information services data, prefetch information on behalf of the user, and perhaps even migrate some of the computational tasks from the mainframe to the workstations.

Initially, we are examining the issues described above by means of a simulation study, although, eventually, we expect to use the results of the simulations to guide an implementation of an information system (using the facilities of the workstation laboratory described in the previous section). Currently, our model consists of a single central computer containing the entire data bank, connected through a network to a set of personal computers. With this model we are now studying the reference patterns that make caching pay off, the processing and storage capacities that are required from the user machines, and the impact of the communication channels on the performance of the system.

The next step in our work will be to study the tradeoffs involved in prefetching data, i.e., in retrieving data before a user even requests it. We expect that personal work profiles will be developed for each database client. These profiles can be determined either by user selection (i.e., the user will provide information as to his normal activities and areas of interest), or compiled based on the user's access patterns. Based on these profiles, the user's machine can request data from the central database before that data is referenced. As in the caching study, we will attempt to identify the conditions that make prefetching profitable (both with and without caching). Our results may also suggest desirable user interfaces that give to the system the information that is most useful for caching and prefetching.

We will also modify our models to allow user workstations to connect to the database through another computer. The motivation for this is that some installations will wish to consolidate the requests from their multiple workstations, and access the database through a single (probably high speed) line. Thus, clusters of user workstations will be connected to medium size computers (the cluster controllers), which will in turn communicate with the database. The controller nodes may cache (and prefetch) data for their installations, and coordinate main database requests. There are many issues to be explored in this situation. For example, the data to be cached may be determined by each workstation's requests (workstation-driven caching) or by the cluster's data demands (cluster-driven caching). Furthermore, the data may be cached in either the workstations (individual caching) or in the controller (group caching). Similarly, prefetching may occur based on the access patterns of a single workstation or on the combined needs of the cluster, and the prefetched information may reside in either the workstations or in the controller. We also need to develop mechanisms that will enable the cluster to reach a consensus on what data will be cached or prefetched. These mechanisms may be centralized (i.e., the controller decides) or distributed (i.e., the cluster members vote). Another issue to be explored arises from the fact that the cluster controller will ask for more data than will the workstations connected directly to the database (i.e., the personal computers). Thus, issues of communication channel allocation and fairness of the access scheme are relevant.

Finally, our models will be extended to allow updates to the database information. Although in the type of information systems we are studying updates will only originate at the central mainframe, these central updates could potentially result in cached and prefetched data invalidations throughout the entire network. It is clear that there is a tradeoff between the benefits of increased amounts of data caching (and prefetching) and the cost of invalidating that data, and those costs depend very strongly on the frequency and extent of the updates. We may consider different strategies for different kinds of data, since certain information seldom changes (i.e., movie reviews), other data can be slightly out of date without causing problems (i.e., phone numbers), and some data changes so quickly (i.e., the latest stock prices) that it may never pay to cache it.

10. Protocols for Multiple-Access of Packet-Broadcast Channels.

This component of our research project focuses on the study of distributed algorithms for scheduling transmissions by a group of users who share a common communication channel. In the hierarchical layer structure of the communication network, this is the part of our project which is closest to the study of the physical layer of the multipoint-to-point communication link. Recent advances accomplished by our research group include: a) a random-access algorithm which achieves the highest possible efficiency in its class [Karo85c], b) the development of a powerful method to analyze the throughput of collisison resolution algorithms [Verd85], and c) a distributed algorithm for allocation of channel resources among users having nonuniform requirements [Karo85a,85b].

We have discovered that substantial improvements over current protocols can be obtained by taking into account round-trip delay information. The effective use of this information, which is available in the vast majority of practical applications allows the design of random access algorithms whose efficiency (measured as the percentage of time the channel is successfully used) can be as high as 67%. This figure should be compared to the efficiency of state-of-the-art random-access algorithms which do not use round-trip delay information. The maximum efficiency achieved to date by this class of algorithms is 48%. The analysis of the collision-resolution algorithm which achieves the best known efficiency in its original form required the use of sophisticated and time-consuming procedures for sequential optimization of Markov decision processes. We have developed a new method which reduces the computation of the efficiency and optimum parameters of the algorithm to a simple optimization problem. In the last part of our project we have obtained one of the first multiple access protocols that provides fair channel access for a heterogeneous user population. This protocol has many additional nice features, e.g., it is adaptive to a time-varying mixture of traffic types; a priori knowledge of the number of users, arrival processes and message lengths is not required, and it supports a mixture of bursty and stream-type traffic. Because of these features, this new protocol is ideal in integrated voice/data networks, which are becoming increasingly important in the telecommunications industry.

11. Completed Thesis

Over the past year, four Ph.D. thesis were completed as part of our project. Daniel Barbara's thesis [Barb85] contains most of his work on vote assignments, including the material of Section 1 and earlier work. Ricardo Cordon completed his work on the use of semantic knowledge to improve transaction processing [Cord85]. His work was summarized in an eralier progress report [Alon85]. Jack Kent's [Kent85b] thesis contains the results of his experimental evaluation of single site crash recovery mechanisms, as well as the material summarized in Section 7. Mark Karol in his thesis [Karo85d] developed multiaccess protocols for a heterogeneous user population. He also utilized knowledge of round-trip delays to develop a very efficient collision resolution algorithm (summarized in Section 10).

12. References

[Alon85] R. Alonso, D. Barbara, R. Cordon, H. Garcia-Molina, J. Kent, F. Pittelli, "Distributed Computing Research at Princeton (1984)," *IEEE Bulletin on Database Engineering*, Vol. 8, Num. 2, June 1985, pp. 68-75

[Alon86] R. Alonso, Phil Goldman, Peter Potrebic, "A Load Balancing Implementation for a Local Area Network of Workstations," *Proc. 1986 IEEE Workstation Technology and Systems Conference*, March 1986, to appear.

[Barb85] D. Barbara, "Mutual Exclusion in Distributed Systems," Ph.D. Thesis, Department of Computer Science, Princeton University, October 1985.

[Barb86a] D. Barbara and H. Garcia-Molina, "Mutual Exclusion in Partitioned Distributed Systems," *Journal of Distributed Computing*, Inaugural Issues, Springer Verlag, to appear.

[Barb86b] D. Barbara, H. Garcia-Molina, A. Spauster, "Policies for Dynamic Vote Reassignment," *Proc. 1986 International Conference on Distributed Computing Systems*, May 1986, to appear.

[Barb86c] D. Barbara, H. Garcia-Molina, A. Spauster, "Protocols for Dynamic Vote Reassignment," Technical Report XXX, Department of Computer Science, Princeton Univeristy, March 1986.

[Cord85] R. Cordon, "Using Semantic Knowledge for Transaction Processing," Ph.D. Thesis, Department of Computer Science, Princeton University, October 1985.

[Davi86] S. Davidson, H. Garcia-Molina, and D. Skeen, "Consistency in a Partitioned Network: A Survey," *ACM Computing Surveys*, to appear, 1986.

[Garc83] H. Garcia-Molina, T. Allen, B. Blaustein, M. Chilenskas, and D. Ries, "Data-patch: Integrating Inconsistent Copies of a Database after a Partition", *Proc. Third Symposium on Reliability in Distributed Software and Database Systems*, October 1983.

[Garc84] H. Garcia-Molina, F. Pittelli, and S. Davidson, "Is Byzantine Agreement Useful in a Distributed Database?" *Proceedings Third SIGACT-SIGMOD Symposium on Principles of Database Systems*, April 1984, pp. 61-69.

[Garc86a] H. Garcia-Molina, "The Future of Data Replication," *Proc. Fifth Symposium on Reliability in Distributed Software and Database Systems*, January 1986, pp. 13-19.

[Garc86b] H. Garcia-Molina, R. Abbott, "Reliable Distributed Database Management," *IEEE Proceedings*, Special Issue on Distributed

Database Systems, to appear.

[Karo85a] M. Karol and S. Schwartz, "A distributed reservation protocol for integrated line- and packet-switched networks, " *IEEE 1985 Proceedings of the Fourth Annual International Phoenix Conference on Computers and Communications,* 1985

[Karo85b] M. Karol and S. Schwartz, "Multiple Access Protocols: Fairness in Heterogeneous Systems, " *IEEE Proceedings INFOCOM'85*,pp. 151-160, March, 1985

[Karo85c] M. Karol and S. Schwartz, "The First/Last Access Protocol," *IEEE Global Telecommunications Conference*, to appear, Dec. 1985

[Karo85d] M. Karol, "Multiple Access Protocols for Broadcast Channels," Ph.D. Thesis, Department of Electrical Engineering, Princeton University, November 1985.

[Kent84] J. Kent and H. Garcia-Molina, "Performance Evaluation of Database Recovery Mechanisms," *Proceedings Fourth SIGACT-SIGMOD Symposium on Principles of Database Systems*, March 1985, pp. 113-122

[Kent85a] J. Kent, H. Garcia-Molina, "Optimizing Shadow Recovery Algorithms," Technical Report CS-012, Department of Computer Science, Princeton University, October 1985.

[Kent85b] J. Kent, "Performance and Implementation Issues in Database Crash Recovery," Ph.D. Thesis, Department of Computer Science, Princeton University, October 1985.

[Koga86] B. Kogan, H. Garcia-Molina, "Achieving High Availability in a Distributed Database System," working paper, 1986.

[Pitt86a] F. Pittelli, H. Garcia-Molina, "Database Processing with Triple Modular Redundancy," *Proc. Fifth Symposium on Reliability in Distributed Software and Database Systems,* January 1986, pp. 95-103.

[Pitt86b] F. Pittelli, H. Garcia-Molina, "Reliable Scheduling in a TMR Database System," Technical Report TR-28-86, Department of Computer Science, Princeton University, March 1986.

[Skee82] D. Skeen, "A Quorum-Based Commit Protocol", *Proceedings Sixth Berkeley Workshop on Distributed Data Management and Computer Networks,* February 1982, pp. 69-80.

[Stoe85] M. Stoessl, "The Design and Implementation of a Highly Reliable, Highly Available Replicated Database System," working paper, April 1985.

[Ston77] M. Stonebraker and E. Neuhold, "A Distributed Database Version of INGRES," *Proceedings 1977 Berkeley Workshop on Distributed Data Management and Computer Networks,* 1977.

[Verd85] S. Verdu, "Computation of the Efficiency of the Mosely-Humblet Collision Resolution Algorithm: A simple method," *Proceedings of the IEEE*, to appear.

[Will82] R. Williams et al, "R*: An Overview of the Architecture," *Proceedings of the International Conference on Databases*, Israel, June 1982.