AD

TECHNICAL REPORT
GIT–ICS–81/15

LEVEL II

(12)

A SIMULATION TOOL FOR DISTRIBUTED DATABASES

By
Nancy D. Griffeth

(12) 32

Prepared for
OFFICE OF NAVAL RESEARCH
800 N. QUINCY STREET
ARLINGTON, VIRGINIA 22217

Under
Contract No. N00014–79–C–0873
GIT Project No. G36–643

September 1981

410044

GEORGIA INSTITUTE OF TECHNOLOGY
A UNIT OF THE UNIVERSITY SYSTEM OF GEORGIA
SCHOOL OF INFORMATION AND COMPUTER SCIENCE
ATLANTA, GEORGIA 30332

81 12 23 086

THE RESEARCH PROGRAM IN
FULLY DISTRIBUTED PROCESSING SYSTEMS

A SIMULATION TOOL FOR DISTRIBUTED DATABASE SYSTEMS

TECHNICAL REPORT

GIT-ICS-81/15

Nancy D. Griffeth

November, 1981

Office of Naval Research
800 N. Quincy Street
Arlington, Virginia 22217

Contract No. N00014-79-C-0873
GIT Project No. G36-643

The Georgia Tech Research Program in
Fully Distributed Processing Systems
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER GIT-ICS-81/15	2. GOVT ACCESSION NO. AD-A108814	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Simulation Tool for Distributed Database Systems		5. TYPE OF REPORT & PERIOD COVERED Technical Report 9/1/81
		6. PERFORMING ORG. REPORT NUMBER GIT-ICS-81/15
7. AUTHOR(s) Nancy D. Griffeth		8. CONTRACT OR GRANT NUMBER(s) N00014-79-C-0873
9. PERFORMING ORGANIZATION NAME AND ADDRESS School of Information and Computer Science Georgia Institute of Technology Atlanta, Georgia 30332		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research 800 N. Quincy Street Arlington, Virginia 22217		12. REPORT DATE 9/1/81
		13. NUMBER OF PAGES 24 + vi
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) same		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited.

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

18 SUPPLEMENTARY NOTES

The findings of this report are not to be construed as an official Department of the Navy position unless so designated by other authorized documents.

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

concurrency control; recovery; atomic action; deadlock; distributed database systems; locking; serializability; timestamps; two-phase commit; two-phase locking; transactions; reliability.

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

An experimental software tool for simulating the behavior of distributed algorithms is proposed. The primary motivation for developing the tool is to study distributed database algorithms. Also, a classification of techniques presently used for distributed database problems of concurrency control and recovery is presented. This classification will be used to reduce the experimentation necessary to compare the performance of alternative algorithms.

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

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

The study and development of distributed algorithms in general and distributed database algorithms in particular is behavior of distributed systems. Both intuition and present-day analytical tools are inadequate to characterize their behavior. Another barrier to understanding such algorithms is the complexity of their interaction, due to the potential lack of synchronization between nodes of a distributed system. Finally, it is not yet clear what "good" behaviors are reasonable to expect from a distributed system. As a result, a multitude of algorithms may exist for solving a single problem, but without more experience and analysis, their behavior cannot be well understood or compared.

This report describes an approach to providing the experience necessary for understanding the behavior of these algorithms.

ABSTRACT

An experimental software tool for simulating the behavior of distributed algorithms is proposed. The primary motivation for developing the tool is to study distributed database algorithms. Also, a classification of techniques presently used for distributed database problems of concurrency control and recovery is presented. This classification will be used to reduce the experimentation necessary to compare the performance of alternative algorithms.

The study and development of distributed algorithms in general and distributed database algorithms in particular is behavior of distributed systems. Both intuition and present-day analytical tools are inadequate to characterize their behavior. Another barrier to understanding such algorithms is the complexity of their interaction, due to the potential lack of synchronization between nodes of a distributed system. Finally, it is not yet clear what "good" behaviors are reasonable to expect from a distributed system. As a result, a multitude of algorithms may exist for solving a single problem, but without more experience and analysis, their behavior cannot be well understood or compared.

This report describes an approach to providing the experience necessary for understanding the behavior of these algorithms.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

Introduction

1.1 The Problem

The basic problem to be addressed by this research project is the development of a methodology for analyzing and comparing distributed database system design alternatives. This problem is both general and specific. In general, we may ask whether there are rules or guidelines for choosing one database design alternative over another. For specific databases, we may ask which design alternative works best according to the requirements of the database. The approach taken addresses both questions, in that studies will be done to determine the general guidelines, but the tool developed for these studies will also be usable in designing specific databases. The design alternatives to be addressed by the studies in this project are the choice of the following algorithms: concurrency control, reliability, and query processing. These algorithms have been chosen because of their central importance to database processing and also because a number of alternative algorithms have already been developed for each problem.

The difficulties of studying any distributed database algorithms are numerous. First, only a few of the proposed alternatives have been implemented at any single site. Thus there is little experience with their performance in general. As a result intuition about their behavior is unreliable. This makes it very difficult even to develop reasonable hypotheses about their behavior. Second, the behavior of a distributed system is much more complex than the behavior of a centralized system. It is necessary to consider not only the behavior of a single system in isolation, but also its interactions with the other nodes of the system. For this reason, it can be exceptionally difficult to prove anything about a distributed algorithm, even that it works correctly. Third, the alternatives designed to solve a given problem make different assumptions about the system on which they are run. They may assume different topologies, different protocols, and different process structures. Even correctness criteria may vary. Finally, few analytical tools for studying distributed systems have been developed so far.

1.2 Objectives

The objectives of this project are:

- development of a software tool for analyzing and studying the design alternatives;
- application of the tool to distributed database design alternatives;
- development of new solutions to distributed database problems using the results of the above study; and
- development of experimental and analytical techniques for studying distributed algorithms in general.

The first objective of this project is the development of an experimental tool for the study of distributed systems, especially distributed database systems. The central experimental tool will be a combination testbed and simulation system. It will allow an algorithm to be coded in as a module of the system. The algorithm can then be tested in this environment. Subsequently, the behavior of the algorithm can be studied with the aid of the simulation facilities provided by the system.

The second objective is to apply the tool to a study of distributed databases. The goal of applying this experimental tool will be to determine how the structure of an system relates to its expected behavior. The assumption is that reasonable structural properties will correspond to good (or bad) behavior in a predictable way. For example, using the classification of concurrency control mechanisms into locking algorithms and timestamping algorithms, we may ask which is more efficient, more robust, or more fair. This should not be taken to imply that only this classifications will be used. In fact, one part of this objective is to determine which classifications provide the most information about behavior.

The third objective is to use the results of the above studies to develop new solutions to distributed database problems, where it is clear from the previous work that existing solutions could be improved on.

The final objective is to develop experimental and analytical techniques for studying distributed algorithms. New techniques to be developed obviously can't be predicted, but the tool itself provides one experimental technique for studying distributed algorithms. Also, experience with the tool should suggest refinements. In addition, the usefulness of various classifications of distributed database algorithms (e.g., BER80, BAD81, HSI81) will be tested. This testing will suggest connections between the classification of an algorithm and its performance that may be used in analysis.

1.3 The Approach

The approach will include the following steps:

- development of a general model of distributed database processing;
- development of the testbed/simulation model;
- validation of the correctness of the system with each design alternative to be tested;
- implementation of the design alternatives for concurrency control and reliability mechanisms as modules of the system;
- simulation experiments to collect empirical data about the behavior of the system with various design alternatives;
- development of hypotheses, on the basis of the experimental data, concerning the behavior of the distributed system with various types of designs; and
- development of analytical proofs of these hypotheses if possible.

The model of distributed database processing will be based on that of Bernstein and Goodman [BER80]. It will be more general in that reliability of the communication system will not be assumed; transaction managers and data managers will be allowed to communicate with either transaction managers or data managers; and in fact a transaction may be passed around to multiple transaction managers for processing, as described in [ROS78].

A central decision to be made in the development of the testbed/simulation model is the choice between a distributed simulation and a centralized simulation. The advantages of distributed simulation are that the testing feature will be more convincing if the simulation system is itself distributed and that it will be more efficient if the communication system is sufficiently fast. The disadvantages are increased hardware cost and overhead the problems of dealing with time; and the need to develop the software for it. Most of the software for a centralized simulation has been written and tested on an existing "ticket-sales" database.

While the number of potential algorithms to be implemented seems prohibitively large, two factors reduce the problem to manageable size:

first, the essential parts of the algorithms are relatively small programs, and second, not all algorithms need to be implemented, just those representative of important classes of algorithms. The plan of attack, in the area of concurrency control, is to build on the work of Bernstein and Goodman [BER80]; Badal [BAD81]; and Hsiao and Ozsu [HSI81]. Each of these papers contains a classification of concurrency control algorithms by their structural properties (e.g., voting or locking; centralized or decentralized). Such classifications will be used as a starting point for analyzing the behavior of the algorithms.

For the experimental results to be of any use, the algorithms must first be verified. Several techniques can be applied: traditional proof techniques, mutation analysis [ACR79], and traditional testing. Also, the data supplied to the system describing the data processing requirements must be realistic. Some possible sources of data for systems which are either partially distributed or reasonable candidates for distribution are banks (e.g., automated teller systems), airlines (ticketing systems); and the military (e.g., personnel and inventory systems).

Some of the measures of system performance to be used in analyzing the results are:

- Average user waiting time;
- Throughput;
- Average queue length at each node; and
- Utilization.

Other measures that need to be considered, to determine whether they are reasonable to look at in a distributed system, are fairness, avoidance of starvation, blocking, degree of concurrency, and so forth.

1.4 Significance

The work done on this project will contribute in a number of ways to the understanding of distributed database systems and to the methodology for designing them. First, the testbed and simulation tool will be usable not only for the duration of this project but will be available for additional work on distributed database systems. Furthermore, it should be sufficiently general to be used for other distributed system projects at Georgia Tech. Second, the tool will be applicable to the design of specific distributed database systems. The use of the tool to test the behaviors of various distributed database algorithms will serve as a thorough test of its correctness and performance. Third, the study of design alternatives for distributed database systems, using the tool, will provide better understanding of the range of alternatives which are reasonable for any particular case, and thus reduce the design problem. Fourth, improved understanding of the behavior of different algorithms for concurrency control, query processing, and reliability may suggest better algorithms. Finally, extensive empirical studies of a distributed database system will provide experience on which to base principles of behavior that any reasonable distributed database system ought to obey.

CHAPTER 2

Background

2.1 General Remarks

The two problems to be studied are concurrency control and reliability. Solutions to these problems will be interdependent, since reliability mechanisms are required to guarantee that concurrent transactions appear atomic to system users in spite of site failures. There are also interactions between the choice of a concurrency control algorithm and the techniques used to provide a reliable system. For example, some concurrency control algorithms are designed to continue functioning correctly in spite of site failures. Others require system reconfiguration when a site fails.

2.2 Concurrency Control

Concurrency control in a database (distributed or not) is a means of guaranteeing correct behavior while allowing maximal concurrency. As an example of the problems that can arise if uncontrolled concurrency is allowed, consider a bank automated teller system. Suppose that a customer's balance is stored redundantly at each of several locations. Then, with uncontrolled concurrency, a customer could arrange to have withdrawals of the entire balance initiated simultaneously at two remote sites; but the balance after these transactions would reflect only one of the withdrawals. This would be nice for the customer, but disastrous for the bank.

The solution to this type of problem is to use a concurrency control algorithm, which prevents this type of behavior. The standard criterion of correctness in a database was developed by Eswaran, Gray, Lorie, and Traiger in [ESW76]. Their model of a database includes _entities_, each of which has a name and a value, and _integrity constraints_, which may be expressed as predicates and restrict the set of values that may be taken on by the entities in the database. For example, in the bank database, we would require that an entity representing a balance be nonnegative and that any two entities representing the same balance (perhaps at different sites of a distributed database) be equal in value. A database state which satisfies all of the integrity constraints is a _consistent_ database state.

The unit of activity on a database is the *transaction*. A transaction consists of a set of basic database actions, usually reads and writes. A consistent transaction changes a consistent database state to another consistent database state. The database state need not be consistent while a transaction is in progress, but it must be consistent when it terminates.

A *schedule* for a set of transactions is an ordered list of the database actions specified by the transactions, preserving the order within individual transactions. If all database transactions are consistent when run alone, then clearly any *serial* schedule of transactions (i.e., a schedule in which each transaction terminates before the next begins) will be consistent. Thus in [ESW76] a database is defined to be *serializable* if it can be transformed to a serial schedule by successively interchanging database actions that cannot affect each other, and it is shown that any serializable schedule is consistent. Subsequently, Stearns, Rosenkrantz, and Lewis [ROS80] have shown that serializability is not only a sufficient but a necessary condition for consistency, if we assume "full functionality" (i.e., no restrictions placed on the interpretation of the operations in a transaction) and all entities are read before they are written.

Concurrency control algorithms are thus used to enforce serializability of schedules of transactions. Actually, one class of algorithms (the timestamp algorithms) may produce schedules which are not strictly serializable but whose effects are exactly the same as some serializable schedule. Serializability of the schedules allowed is thus the standard criterion of correctness of a concurrency control algorithm.

Several authors [LYN81,RIE81,GAR81] have proposed various generalizations of serializability as an alternative criterion for correctness of concurrency control algorithms. Lynch's generalization provides for the user (or application system) to specify a set of interleavings of actions which are correct. The set may include nonserializable as well as serializable interleavings. Garcia-Molina proposes two levels of locking, local and global. Local locking is used to guarantee that a sequence of actions is atomic at a single site. Global locking is used in the usual way for detection of concurrency conflicts. The advantage of his method is that knowledge of the database semantics may be used to allow a non-local transaction to release local locks as soon as its local activity is complete. For example, a transfer of money from one bank branch to another may be considered completed as it has been determined that there is enough money in the source branch to perform the transfer. Ries and Smith discuss "nested transactions", in which one transaction system uses transactions provided by a second transaction system. The nested transactions may be serialized with each other in any order, not necessarily in the same order as the calling transactions. For example, if two database transactions request the file system to allocate space, it is not necessary to serialize the space allocations in the same order as the database transactions.

2.2.1 Concurrency Control Algorithms

Bernstein and Goodman categorize concurrency control algorithms as either two-phase locking algorithms or as timestamping algorithms [BER80]. Two-phase locking algorithms ensure consistency by prohibiting a transaction from requesting more locks if it has released any locks. Each transaction has a "growing" phase during which it requests locks and a "shrinking" phase during which it releases the locks it has set. Between these two phases is a "lockpoint"; the execution behaves as if all entities were updated at the lockpoint. Locking schemes are prone to deadlocks and require a policy for avoiding or breaking them.

Timestamp ordering algorithms depend on assigning a unique time to each transaction as it arrives, and guaranteeing that the effect of running a group of transactions is the same as if they had been run serially in arrival order. A transaction must not perform updates on the basis of data which is out-of-date. That is, it must not overwrite an update created by a later transaction. Also, it must not read data written by a later transaction.

Centralized concurrency control algorithms are all locking schemes, in which locks are controlled centrally and must be requested from a designated site. One variant of this is Stonebraker's "primary copy" scheme for INGRES [STO79], in which the site may vary from one data entity to another. A decentralized algorithm which utilizes locking is called "basic 2PL" by Bernstein and Goodman [BER80]. In this technique, the lock on an entity is granted by the site at which it is stored. They also describe a technique called "voting 2PL", which requires only that a transaction obtain a majority of the locks for each data item it requires. Since only one transaction at a time can have a majority, this is sufficient to prevent consistency violations.

Timestamping approaches to concurrency control including voting schemes, a multi-version database algorithms, and the SDD-1 protocols. The best-known voting scheme is probably Thomas' majority voting algorithm [THO79] (also called the distributed voting algorithm by Garcia-Molina [GAR78]), in which a majority of the sites must approve any transaction. This idea has been generalized by Gifford [GIF79] to allow assignment of any number of votes to each site, and require only that a majority of votes be collected by a transaction. This reduces to a centralized algorithm if one site has all the votes. As Thomas noted in [THO79], any rule will work which requires that two conflicting transactions both get permission to proceed from some single site.

Reed's multi-version algorithm [REE78] requires that multiple versions of each entity be maintained in the database, with each version including the range of times for which the value is known to have applied. Each action on the database has a time associated with it. If it is a read and the entity has a value for some range of times including the read, then the value is returned; if no such value exists, the range of times for some value is increased to include the time of the read. If the action is a write, it must not change a value which already holds for the time of the write; if it tries to, the transaction is aborted.

The SDD-1 protocols [BER77] also utilize timestamps to guarantee different levels of synchronization of transactions. The idea is that many groups of transactions will require only limited synchronization with respect to each other. To take advantage of this fact, the transactions must be analyzed beforehand to determine the types of synchronization required. Of course, this requires that the transactions to be used are known beforehand. Four types of synchronization are identified. P1 synchronization is purely local; no global synchronization is attempted. P2 synchronization can be used to guarantee that reads are consistent, although they may be out-of-date. The largest local entity timestamp at the site initiating the transaction is used as the time of the read. P3 synchronization guarantees that reads are up-to-date as of the current time of the transaction; this is used for potentially conflicting updates. P4 synchronization is used for unanticipated transactions and for P2 or P3 transactions requiring so many entities they might be subject to starvation.

The classification into locking and timestamping algorithms refers to the method used to prevent consistency violations. The locking algorithms require that a transaction must reach a lockpoint, when it has exclusive control of all data-items, before it may complete. The timestamping algorithms require that actions on data-items be performed in timestamp order of the transactions requesting the actions. But it is also necessary to decide what to do with transactions that never reach their lockpoints (due to deadlocks) and with transactions that discover a timestamp conflict with other active transactions.

2.2.2 Deadlock Management

Deadlocks may be handled either by deadlock detection or deadlock prevention. Deadlock detection requires maintaining a graph of active transactions. The nodes of the graph represent transactions and the arcs represent the "waits-for" relation. Deadlock prevention requires guaranteeing that no deadlocks ever occur.

Centralized deadlock detection could be used with a centralized locking algorithm. However, it would be extremely expensive with a decentralized algorithm. Two methods for decentralized deadlock detection are described in [MEN78]. One method imposes a hierarchy on the network and detects deadlocks at the lowest possible node of the tree. This method was designed to reduce the communications cost incurred with centralized deadlock detection. The second method requires recursively sending notification of new "blocking transactions" to the originating site of each transaction thus blocked. This method was designed to continue functioning in a system prone to failures.

If a deadlock prevention method is to be used, one way of guaranteeing that no deadlocks occur is to guarantee that locks are assigned in the same order to all transactions for all entities referenced at all sites. This can be done by assigning sequence numbers to transactions and granting lock requests to the lowest pending sequence number. This technique is used in Garcia-Molina's "hole list" (MCLA-h) scheme [GAR78, GAR79]. In this scheme, instead of requiring each action on a database entity to wait at the central site for a lock, a sequence number is assigned to the transaction, and the action proceeds immediately to the distributed sites. The "hole list" refers to a list of sequence numbers of transactions the sites need not wait for. Another technique using sequence numbers is Lelann's token-passing scheme [LEL78], in which the site with the "right" to grant sequence numbers is the site having possession of a token, which is passed around a ring. Two other locking algorithms using sequence numbers are the centralized WAIT-DIE and WOUND-WAIT algorithms of Rosenkrantz, Stearns, and Lewis [ROS78].

2.2.3 Conflict Resolution

Finally, with either locking or timestamping algorithms, it is necessary to decide how to resolve conflicts (i.e., deadlocks, potential deadlocks, and timestamp conflicts). This can be done by using a sequence numbering scheme such as the "valid numbering schemes" described in [ROS78] or by voting, as in [THO79] and [GIF79]. Timestamps qualify as a valid numbering scheme. Algorithms which avoid conflicts by assigning a number or timestamp to each transaction and then forcing each transaction to wait until all previous transactions have executed will be classified as resolving conflict using a numbering scheme.

2.3 Reliability in a Distributed Database

The goals of a reliability mechanism in a distributed database system are to guarantee that:

- the active sites can continue to function in the presence of failure; and
- a failed site can be restored to the system when the cause of the failure is corrected.

The first goal, to permit the system to continue to function in the presence of failure, requires (1) detection of the failure; (3) possible reconfiguration of the system after the failure is detected; and (2) preserving the atomicity of transactions that may be active both before and after the failure. The second goal, restoring the failed site after the cause of the failure is corrected, requires (1) sufficient information to determine what the current state of the site should be; (2) a protocol for reintroducing it into the system; and (3) possible reconfiguration of the system after the failed site has recovered.

In this project, it will be assumed that failures are detected by some means (e.g., as in the "local status layer" of RELNET [HAM81]). The remaining problems are reconfiguration; atomicity; information requirements; and the post-failure protocol.

2.3.1 Reconfiguration

Reconfiguration may not be required if the site algorithms are written so that the system continues to function in the same way in spite of failures. In many cases, however, there are "special-purpose" sites (primary sites, in [ALS76] and INGRES [STO77,STO79]; spoolers and commit backup processes, in SDD-1 [HAM81]) whose functions must be reassigned to other sites when the special-purpose site fails. The reassignment may be fixed before the site failure, as in [ALS76] and [HAM81] or it may be determined after the site failure (e.g., by a vote of the live sites [GAR81]). Reconfiguration following a site recovery would then involve reassigning a special function to the recovered site or possibly assigning it as a backup for such a function.

2.3.2 Atomicity

A transaction is defined as a set of primitive database operations. It is required to be an atomic unit of action, that is, either all operations of the transaction are performed or none are. In a distributed system, this means that if any site decides to "commit" itself to the transaction, then all sites must. Also, if any site decides not to perform the transaction, then the remaining sites must agree.

Site failure raises the possibility that the failed site may never know what decision the other sites came to. Conversely, the other sites involved may not know what the failed site decided to do. But the atomicity requirement means that all sites must agree on the decision, in spite of failures.

The standard solution to this problem is the "two-phase commit" protocol [GRA78]. In the first phase, changes to the database are made in a reversible way. In the second phase (the "commit" phase), when it is known that all sites making changes are agreed to make them, then changes are made permanent. If any site decides not to make the changes, then the transaction is aborted. The basic choices are how to make the changes reversible and how to decide to make the changes permanent.

Changes may be made reversible in two ways: by writing an UNDO log entry before making the changes or by changing copies only until the decision to make the changes permanent has be made. The logging technique is discussed in [GRA78]. Updating of copies only is used in DELTA [LEL81]. Reed's multiversion system [REE78] may also be viewed as updating only copies until the commit is made.

The decision to make the changes permanent may be made by a vote of all involved sites [GRA78,LEL81] or by reaching the "normal" or "abnormal" end of a transaction [REE78,ROS78]. Which technique is used is related to the underlying model of transaction execution. If a transaction is executed by a "transaction manager" (SDD-1) or a "producer" (DELTA), which sends a sequence of read, write, and commit commands to the other sites, then there is a natural choice of site to initiate and count the vote. If, however, the transaction is viewed as a process which migrates from site to site, then it is more natural to let the site at which the transaction terminates (either normally or abnormally) make the decision to commit or abort it, depending on the type of termination.

A problem with the two-phase commit protocol is that the final decision to commit or abort a transaction may be delayed until after a failed site has been recovered. For example, if the failed site is the "transaction manager" in SDD-1 or the "producer" in DELTA, then the count of the vote would be delayed. The system can correctly wait for the site to recover, but the delay may be intolerable.

The alternatives are to abort the transaction immediately when a component fails; to tolerate the delay; or to introduce a new protocol for committing transactions. The third approach is taken in [SKE81], in which sites seek a concensus on committing or aborting; and in SDD-1 [BER80], in which only a transaction may be aborted only on a read, so that once all update messages have been passed to the guaranteed delivery layer of the message system, the transaction may be committed in spite of site failures.

2.3.3 Information Requirements

A useful classification of the information used in restoring a failed site to the system is given in [GAR81]. He identifies three possibilities: no information is used, a log is used, or "persistent messages" are used. If no information is used, then the current state of the failed site must be determined from the states of the active sites in the system. Thus there must be enough redundancy in the system to allow determination of the state of one site from some subset of the other sites. If a log is used, as in [GRA78], the failed site recovers by performing all of the missed actions. If "persistent messages" are used, then the communications system must remember all the missed actions and guarantee that the failed site receives them [HAM81].

2.3.4 Recovery Protocol

When a failed site recovers, it is first brought up-to-date, as discussed in the preceding paragraph. Subsequently, it rejoins the system, possibly with reconfiguration of the system. For example, in [ALS76], the site must first request to rejoin the system. This request must be transmitted, either directly or through or sites, to the primary site, which informs all sites to add the "new" site to their tables. In [LEL81], when a node wants to rejoin the system, it must get a checkpoint and all subsequent actions to bring it up-to-date. Then it may rejoin.

2.4 Performance Studies

To date, there has been little published work on the performance of concurrency control algorithms. Three major exceptions are the work of Garcia-Molina [GAR78,GAR79], Gelenbe and Sevcik [GEL78], and Bernstein and Goodman [BER80]. Garcia-Molina's work has focussed primarily on simulating certain algorithms. Gelenbe and Sevcik have suggested a queuing network approach to determining two measures of internal database performance (as opposed to external measures such as response time and throughput). Bernstein and Goodman have also analyzed many concurrency control algorithms, comparing them according to several internal measures. The underlying thesis of the proposed work is that the above-mentioned work can be significantly extended by combining the approaches. The simulation experiments can suggest theorems to prove and provide examples of system behavior to explain by analytic methods. When analytic methods fail, simulation can be used to clarify the behavior of the algorithms. This technique was used with success in the work on "ticket systems" discussed in section IV.

Garcia-Molina has done extensive simulation of 3 algorithms (and some variants): centralized locking, distributed voting, and Ellis' ring algorithm. Other, significantly different algorithms not covered in his work include Reed's multiversion algorithm, the WAIT-DIE and WOUND-WAIT algorithms, and the SDD-1 algorithms. His simulations were based on a model that allowed only updates on a fully redundant distributed database system. His results show that the centralized version has lower response time for all but very heavy loads; lower I/O utilization, probably because of the redundant I/O required in a fully redundant system; and slightly fewer messages per update. The crucial parameter in determining the difference seemed to be I/O utilization, suggesting that less redundancy in the database might produce more favorable results for decentralized algorithms. An important factor of the response time in either case is the load, as reflected in the transaction interarrival time. The simulation model used in the work proposed here would have to include significantly more detail than Garcia-Molina's, in order to determine why a concurrency control algorithm caused observed behavior patterns. For the same reason, some additional performance measures would be of interest, such as congestion at a node, blocking, restarts, etc.

Gelenbe and Sevcik have developed a queuing analysis technique for evaluating distributed database systems. Their measures of database performance are the <u>coherence</u> (i.e., the degree of agreement of the sites on the value of an entity) and the <u>promptness</u> (i.e., the average time required to update an entity at a site). Their techniques are illustrated in [GEL78] on two rather special-purpose database systems but would also apply to more general databases. The analytical technique can be used to help validate the simulation results. The measures defined by Gelenbe and Sevcik apply to the internal database behavior rather than to a database user's external view of its performance. The intention of the proposed work is to relate the performance of the database, as seen by a user, to its internal behavior, and to relate the internal behavior to the operation of the concurrency control algorithm in the particular distributed database system. This will relate the design of the distributed database to the output of the database and not just to its internal appearance.

Bernstein and Goodman have discussed the performance of a huge variety of concurrency control algorithms in [BER80]. They use four measures which they regard to be important in the total cost of concurrency control and which can be determined analytically from the algorithms themselves. These measures are: communication overhead (represented by number of messages), local processing overhead, blocking, and restarts. The relationship of these measures to response time and throughput depends on assumptions about how a distributed system behaves.

At present, such assumptions must necessarily be generalizations of experience with single-computer systems, since there has been so little experience with distributed systems. Unfortunately, it is hard to reason about systems with which we have had little experience. As a result, many seemingly obvious assumptions and hypotheses about distributed systems may prove wrong. The work to be described in section IV mentions two examples of this problem. The simulation experiments that I am proposing provide one way to gain experience with distributed database systems.

2.5 Simulation Techniques

The simulation of a distributed database system can be done using conventional simulation techniques. Such an approach was taken for the "ticket system" work discussed below. However, primarily for reasons of performance, the use of distributed simulation may be preferable. A number of papers have appeared recently on this topic [BRY79,CHA79,PEA79]. The primary problem with using a distributed system for simulation is the management of simulation time when no shared variable "clock" is available. Chandy [CHA79] has proposed a "time-exchange" system which requires each process to maintain a time on each of its output lines, and to take the next event from the input line with the lowest time. Peacock, Wong, and Manning [PEA79] have extended the method of Chandy and devised other methods as well, including a "scaled real-time" method in which simulation time is simply scaled real time. In the terminology of Peacock, Wong, and Manning, the simulation methods most likely to be of use in this project are the "loose event-driven" methods (because they should provide the best performance) and the "scaled real-time" method (to assist in developing intuition).

CHAPTER 3

The Simulation Tool

3.1 Introduction

The objectives of this project are (1) to develop an experimental software tool for testing and simulating distributed systems; (2) to apply the tool to distributed database systems; (3) to develop new solutions to distributed database problems using the results of the experiments; and (4) to develop both experimental and analytical techniques for studying distributed algorithms in general.

The first two objectives require development of a model of distributed database systems. This model will of necessity include a sub-model of a distributed system. The first objective -- that the tool be applicable to distributed systems in general -- requires that the submodel be separable from the model and that it be sufficiently general to allow study of a wide range of distributed system problems. Problems likely to be addressed at Georgia Tech (in addition to database problems) are distributed compilation and distributed resource allocation.

3.2 The Distributed Database Model

There are four parts to the distributed database model: the communication system submodel the distributed system submodel, the data system submodel, and the user interface submodel.

3.2.1 The Communication System Submodel

In the communication system submodel, it will be assumed that point-to-point communication can be described by the following parameters:

- the delay time distribution function;
- the mean delay time;
- the variance in delay time (if applicable); and
- the probability that a message is lost.

These parameters may change dynamically, to simulate line failures while the system is running. The communication system submodel simulates the data link and physical layers of the ISO reference model of open systems interconnection [ISO81].

3.2.2 The Distributed System Submodel

The distributed system submodel will contain any required routing and error recovery techniques. It simulates the transport and network layers of the ISO reference model. For many topologies to be tested (e.g., star tree, and loop), the routing algorithms should be trivial. Several standard ones can be supplied as part of the software tool. The distributed system submodel will also contain the characteristics of the system nodes. These will include the following parameters:

- the node step time;
- the access time to secondary memory;
- the node memory size; and
- the secondary memory size.

3.2.3 The Data System Submodel

The data system submodel will contain "data managers" and the user interface submodel will contain "transaction managers", as in the Bernstein and Goodman model of distributed database systems [BER80]. Operations performed by the data system submodel are:

- read a data granule (item, record, page, etc.);
- write a data granule;
- lock a data granule;
- unlock a data granule;
- read a timestamp for a data granule;
- set a timestamp for a data granule;
- commit a data granule.

The definition of data granule is similar to the definition of Ries and Stonebraker [RIE77]. It specifies the smallest unit of data that can be locked and unlocked (for concurrency control), read and written (for query processing), or written to secure storage (for reliability). To permit study of algorithms in which the transaction managers do not know where data may be stored -- only the data managers know where it is -- the data managers will be allowed to communicate with each other. To permit study of algorithms assuming that transactions may be passed from site to site, the transaction managers will also be allowed to communicate.

3.2.4 The User Interface Submodel

The user interface submodel will process the transactions. Transactions are identified by special delimiting statements at the beginning and end. The statements inside a transaction may be any sequence of data manager operations.

```
-----------------------------------------------------------------------

                Node A                Node B


User <--------->User                  User     <--------->User
                Interface             Interface
                   ^                     ^
                   |                     |
                   V                     V
Database <----->Data                  Data     <--------->Database
                Subsystem             Subsystem
                   ^                     ^
                   |                     |
                   V                     V
                Distributed           Distributed
                Subsystem             Subsystem
                   ^                     ^
                   |                     |
                   V                     V
                Communication<----->Communication
                Subsystem             Subsystem
-----------------------------------------------------------------------
```

Figure 1. A Schematic of the Distributed Database System Model

3.3 System Architecture and Specifications

The proposed experimental tool will contain a module corresponding to each of the submodels discussed in the preceding section. Parameters may be specified independently for each module and algorithms may be plugged into the appropriate module.

3.3.1 Output Analysis

In addition, the results of the simulation must be tabulated. To accomplish this purpose, each system action (i.e., message or access to a database) will be logged. The log will be used to compute the following basic measures:

- expected response time;
- throughput;
- utilization; and
- queue length at each node.

Expected response time, throughput, and queue lengths can be computed on the basis of the user interface module output. Utilization must be computed from information recorded by the communication system, distributed system, and data system modules.

Secondary measures whose relationship to the primary measures will be of interest are:

- number of messages;
- number of bits sents
- number of errors in transmission;
- number of nodes (dispersion) required by a transaction or query;
- number of nodes actually used in responding to a transaction or query;
- local processing overhead; and
- I/O time.

BIBLIOGRAPHY

[ACR79] Acree, A. T., T. A. Budd, R. A. DeMillo, R. J. Lipton, F. G. Sayward. "Mutation analysis". Technical Report GIT-ICS-79/08.

[ALS76] Alsberg, P. A., and J. D. Day. "A principle for resilient sharing of distributed resources." Proceedings of the 2nd International Conference on Software Engineering, San Francisco (1976), 562-570.

[BAD79] Badal, D. Z. "On efficient monitoring of database assertions in distributed databases". Proceedings of the 4th Berkeley Conference on Distributed Data Management and Computer Networks (August, 1979). 125-135.

[BAD78] Badal, D. Z., and Popek, G. J. "A proposal for distributed concurrency control for partially redundant distributed data base systems". Proceedings of the 3rd Berkeley Workshop on Distributed Data Management and Computer Networks (1978). 273-285.

[BAD81] Badal, D. Z. "Concurrency control overhead or a closer look at blocking vs. nonblocking concurrency control mechanisms". Proceedings of the 5th Berkeley Conference on Distributed Data Management and Computer Networks (June 1981).

[BER77] Bernstein, P. A., Shipman, D. W., Rothnie, J. B., and Goodman, N. "Concurrency control in a system for distributed databases (SDD-1)". ACM Transactions on Database Systems 5, 1 (March 1980), 18-51.

[BER80] Bernstein, P. A., and Goodman, N. "Fundamental algorithms for concurrency control in distributed database systems". CCA Technical Report (Feb. 1980).

[BRY79] Bryant, R. E. "Simulation on a distributed system". Proceedings of the First International Conference on Distributed Systems, (Oct. 1979), 544-552.

[CHA79] Chandy, K. M., Holmes, V., and Misra J. "Distributed simulation of networks". Computer Networks 3, 2 (1979). 105-83.

[ELL77] Ellis, C. A. "A robust algorithm for updating duplicate databases". Proceedings of the 2nd Berkeley Workshop on Distributed Data Management and Computer Networks (1977). 146-158.

[EPS78] Epstein, R., M. Stonebraker, and E. Wong. "Distributed query processing in a relational data base system". Proceedings of the ACM-SIGMOD International Conference on the Management of Data (June 1978). 169-180.

[EPS80] Epstein, R., and M. Stonebraker. "Analysis of distributed data base processing strategies". Proceedings of the Sixth International Conference on Very Large Data Bases (1980). 92-101.

[ESW76] Eswaran, K. P., Gray, J. N., Lorie, R. A., and Traiger, I. L. "On the notions of consistency and predicate locks in a database system". Comm. ACM 19, 11 (Nov. 1976), 624-633.

[FIS80] Fischer, M. J., Griffeth, N. D., and Lynch, N. A. "Optimal placement of resources in a distributed network". To appear in Proceedings of the Second International Symposium on Distributed Computer Systems (April, 1980).

[GAR78] Garcia-Molina, H. "Performance comparison of two update algorithms for distributed databases". Proceedings of the 3rd Berkeley Conference on Distributed Data Management and Computer Networks (1978). 108-89.

[GAR80A] Garcia-Molina, H. "Reliability issues for completely replicated distributed databases". Technical Report 266, Department of EECS, Princeton University (April, 1980).

[GAR80B] Garcia-Molina, H., and Wiederhold, G. "Read-only transactions in a distributed database". Technical Report 267, Department of EECS, Princeton University (April, 1980).

[GAR79] Garcia-Molina, H. "Performance comparison of update algorithms for distributed databases". Progress Report 8 (February, 1979).

[GAR81] Garcia-Molina, H. "Elections in a distributed computing system". Panel discussion, IEEE Symposium on Reliability in Distributed Software and Database Systems (July 1981).

[GRA78] Gray, J. N. "Operating systems: an advanced course", ed. by R. Bayer, R. M. Graham, and G. Seegmuller. Springer-Verlag, New York, 1978, 393-481.

[GRD80] Gardarin, G, and Chu, W. W. "A distributed control algorithm for reliably and consistently updating replicated databases". IEEE Transactions on Computers C-29, 12 (Dec. 1980). 1060-1067.

[GEL78] Gelenbe, E., and Sevcik, K. "Analysis of update synchronization for multiple copy data-bases". Proceedings of the 3rd Berkeley Conference on Distributed Data Management and Computer Networks (1978). 69-90.

[GIF79] Gifford, D. K. "Weighted voting for replicated data". Xerox Palo Alto Research Center Technical Report CSL-79-14 (Sept. 1979).

[HAM80] Hammer, M. and D. Shipman. "Reliability mechanisms for SDD-1: a system for distributed databases". ACM Transactions on Database Systems 5, 4 (Dec. 1980), 431-466.

[HEV78] Hevner, A. R., and S. B. Yao. "Query processing on a distributed database". Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks (August 1978). 91-107.

[HSI81] Hsiao, D. K., and T. M. Ozsu. "A survey of concurrency control mechanisms for centralized and distributed databases." Technical Report OSU-CISRC-TR-81-1 (February 1981).

[ISO81] ISO/TC97/[illegible] "Data processing -- open systems interconnection -- basic ref[illegible] model". Computer Networks 5, 2 (April 1981). 81-88.

[KIM76] Kimbleton, S. R., and Schneider, G. M. "Computer Communication Networks: Approaches, Objectives, and Performance Considerations". ACM Computing Surveys 7,3 (1976). 129-173.

[LAM76] Lamport, L. "Towards a theory of correctness for multi-user data base systems". MCA Technical Report CA-7610-0712 ([illegible]976).

[LAM78] Lamport, L. "Time, clocks, and the ordering of events in a distributed system". Comm. ACM 21, 7 (July 1978). 558-565.

[LAM79] Lampson, B. W., and H. E. Sturgis. "Crash recovery in a distributed data storage system." To appear in CACM.

[LEL78] Lelann, G. "Algorithms for distributed data-sharing systems which use tickets". Proceedings of the 3rd Berkeley Workshop on Distributed Data Management and Computer Networks (1978). 259-272.

[LEL81] Le Lann, G. "A distributed system for real-time transaction processing." Proceedings of the 14th HICSS Conference, Hawaii, USA (Jan., 1981).

[LIN79] Lindsay, B. G., et. al. "Notes on distributed databases". IBM Research Report RJ2571 (July 1979).

[MEN78] Menasce, D. A., and Muntz, R. R. "Locking and deadlock detection in distributed databases". Proceedings of the 3rd Berkeley Workshop on Distributed Data Management and Computer Networks (1978). 215-232.

[MEN80] Menasce, D. A., Popek, G. J., and Muntz, R. R. "A locking protocol for resource coordination in distributed databases". ACM Transactions on Database Systems 5, 2 (June 1980). 103-138.

[PAI79] Paik, I.-S., and C. Delobel. "A strategy for optimizing the distributed query processing". Proceedings of the Fifth International Conference on Very Large Data Bases, 1979. 686-698.

[PWM79] Peacock, J. K., Wong, J. w., and Manning, E. G. "Distributed simulation using a network of processors". Computer Networks 3, 1 (1979). 44-56.

[REE78] Reed, D. P. "Naming and synchronization in a decentralized computer system". Ph.D. Thesis, MIT, Department of EECS (Sept. 1978).

[RIE81] Ries, D. R. and G. C. Smith. "Nested transactions in distributed systems". Proceedings of the IEEE Symposium on Reliability in Distributed Software and Database Systems (July, 1981), 117-123.

[ROS78] Rosenkrantz, D. M., Stearns, R. E., and Lewis, P. M. "System level concurrency control for distributed database systems". ACM Transactions on Database Systems 3, 2 (1978). 178-198.

[ROS80] Rosenkrantz, D. M., Stearns, R. E., and Lewis, P. M. "Consistency and serializability in concurrent database systems". SUNY at Albany Dept. of Computer Science Technical Report 80-12 (August 1980).

[ROT80] Rothnie, J. B. et. al. "Introduction to a system for distributed databases (SDD-1). ACM Transactions on Database Systems 5,1 (1980). 1-17.

[ROT77] Rothnie, J. B., and Goodman, N. "A survey of research and development in distributed database management". Proceedings of the 3rd International Conference on Very Large Databases (1977).

[SCH78] Schapiro, R. M., and R. E. Millstein. "Failure recovery in a distributed data base system." Proceedings of the 4th International Conference on Very Large Databases (1978), 66-70.

[SKE81] Skeen, D. "A decentralized termination protocol". Proceedings of the IEEE Symposium on Reliability in Distributed Software and Database Systems (July, 1981), 27-32.

[STE81] Stearns, R. E., and D. J. Rosenkrantz. "Distributed database concurrency controls using before-values." Technical Report 81-1 (February 1981).

[STO77] Stonebraker, M., and Neuhold, E. "A distributed data base version of INGRES". Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks. May 1977, 19-36.

[STO79] Stonebraker, M. "Concurrency control and consistency of multiple copies of data in distributed INGRES". IEEE Trans. Software Engineering SE-5, 3 (May 1979). 188-194.

[THO79] Thomas, R. H. "A majority consensus approach to concurrency control for multiple copy databases". ACM Transactions on Databases 4, 2 (June 1979).

[TOA79] Toan, N. G. "A unified method for query decomposition and shared information updating in distributed systems". Proceedings of the Fifth International Conference on Very Large Data Bases. 1979, 679-685.

[WON77] Wong, E. "Retrieving dispersed data from SDD-1: a system for distributed databases". Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks (May 1977). 271-235.

[YOU79] Youssefi, K. and E. Wong. "Query processing in a relational database management system." Proceedings of the Fifth International Conference on Very Large Data Bases. 1979, 409-417.