PROJECT ADMINISTRATION DATA SHEET

[x] ORIGINAL [ ] REVISION NO. _____

roject No. G-36-667 (R-5491-1A0 AS of 7/1/82) DATE 6/4/82

roject Director: Nancy D. Griffeth School/Lab I & CS

ponsor: National Science Foundation; Washington, D.C.

ype Agreement: Grant No. MCS - 8200854

ward Period: From 7/1/82 To 12/31/84 (Performance) _____ (Reports)

ponsor Amount: $84,632 Contracted through:

ost Sharing: $4,455 (G-36-347 (F-5491-1A0 AS of 7/1/82)) GTRI/~~GIT~~

itle: Design of Distributed Data Base Systems

ADMINISTRATIVE DATA

OCA Contact Leamon R. Scott

1) Sponsor Technical Contact:

Thomas A. Keenan
Software Systems Science Program
Computer Science Section
Division of Mathematical & Computer Sciences
Directorate for Math & Physical Sciences
NSF
Washington, D.C. 20550 202-357-7375

2) Sponsor Admin/Contractual Matters:

Shirley P. Greene
MPS/STIA Branch
Division of Grants & Contracts
Directorate for Administration
NSF
Washington, D. C. 20550
202-357-9671

Defense Priority Rating: N/A

Security Classification: N/A

RESTRICTIONS

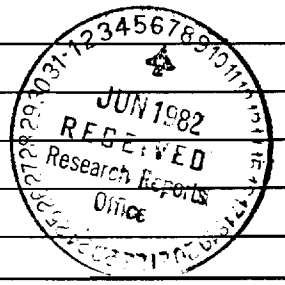
ee Attached NSF Supplemental Information Sheet for Additional Requirements.

ravel: Foreign travel must have prior approval – Contact OCA in each case. Domestic travel requires sponsor approval where total will exceed greater of $500 or 125% of approved proposal budget category.

quipment: Title vests with GIT

COMMENTS:

* Includes a 6 month unfunded flexibility period.



COPIES TO:

Administrative Coordinator
Research Property Management
Accounting
Procurement/EES Supply Services
Research Security Services
Reports Coordinator (OCA) ✓
Legal Services (OCA)
Library
EES Public Relations (2)
Computer Input
Project File
Other _____

ORM OCA 4:781

GEORGIA INSTITUTE OF TECHNOLOGY OFFICE OF CONTRACT ADMINISTRATION

SPONSORED PROJECT TERMINATION/CLOSEOUT SHEET

Date 7/22/85

Project No. G-36-667 School/~~Lab~~ ~~XXX~~ ICS

Includes Subproject No.(s)

Project Director(s) N. D. Griffeth GTRC / ~~XXX~~

Sponsor National Science Foundation

Title Design of Distributed Database Systems

Effective Completion Date: 12/31/84 (Performance) 3/31/85 (Reports)

Grant/Contract Closeout Actions Remaining:

- [X] None
- [ ] Final Invoice or Final Fiscal Report
- [ ] Closing Documents
- [ ] Final Report of Inventions
- [ ] Govt. Property Inventory & Related Certificate
- [ ] Classified Material Certificate
- [ ] Other

Continues Project No. N/A Continued by Project No. N/A

COPIES TO:

Project Director
Research Administrative Network
Research Property Management
Accounting
Procurement/GTRI Supply Services
Research Security Services
Reports Coordinator (OCA)
Legal Services
Library
GTRC
Research Communications (2)
Project File
Other A. Jones; M. Heyser

FORM OCA 69.285

G-36-667

FINAL PROJECT REPORT

NSF GRANT MSC-8200[illegible]
Software Systems Science Program
July 1, 1982 - December 31, 1984

DESIGN OF DISTRIBUTED DATABASE SYSTEMS

Final Project Report
April 1, 1985

Nancy D. Griffeth
Assistant Professor
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

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

NATIONAL SCIENCE FOUNDATION
Washington, D.C. 20550

FINAL PROJECT REPORT
NSF FORM 98A

PLEASE READ INSTRUCTIONS ON REVERSE BEFORE COMPLETING

PART I–PROJECT IDENTIFICATION INFORMATION

1. Institution and Address	2. NSF Program	3. NSF Award Number
Georgia Institute of Technology School of Information and Computer Atlanta, Georgia 30332 Science	Software Systems Science	MCS - 8200854
	4. Award Period From 7/1/82 To 12/31/84	5. Cumulative Award Amount $ 84,632

6. Project Title

PART II–SUMMARY OF COMPLETED PROJECT *(FOR PUBLIC USE)*

The focus of this research project is the behavior of distributed database systems in the presence of failures and concurrent access to the database. Several algorithms for reliable distributed database systems have been developed as part of the work. The techniques employed to study the behavior of the algorithms have included formal analysis of their properties; simulation studies of their performance; and a novel method of queuing network analysis, which can be used even when transactions are allowed to hold multiple resources simultaneously. The algorithms studied are : (1) five transaction scheduling algorithms (three are novel, to the best of our knowledge) which quarantee that each transaction either completes or has no effect on the database; (2) novel logging and roll-back algorithms which allow localization of the recovery effort to individual levels of abstraction; (3) a novel file migration algorithm for a distributed system which quarantees that at least one copy of the file is always available; and (4) a novel resource allocation algorithm (which could be applied to redundant copies of a file) which is amenable to expect cost analysis and which provides a very reasonable allocation of resources to requests. Also as part of this work, my colleagues and I have completed the study of some algorithms for determining an optimal placement of identical resources in a network structured as a tree and for allocating the resources to requests arriving in the tree.

PART III–TECHNICAL INFORMATION *(FOR PROGRAM MANAGEMENT USES)*

1. ITEM *(Check appropriate blocks)*	NONE	ATTACHED	PREVIOUSLY FURNISHED	TO BE FURNISHED SEPARATELY TO PROGRAM	
				Check (✓)	Approx. Date
a. Abstracts of Theses		X			
b. Publication Citations		X			
c. Data on Scientific Collaborators		X			
d. Information on Inventions					
e. Technical Description of Project and Results		X			
f. Other *(specify)*					

2. Principal Investigator/Project Director Name *(Typed)*	3. Principal Investigator/Project Director Signature	4. Date
Dr. Nancy D. Griffeth		5/27/85

JAN 2 8 1985

GEORGIA INSTITUTE OF TECHNOLOGY

Office of
Graduate Studies and Research

Date January 14, 1985

REQUEST FOR APPROVAL OF THESIS TOPIC

NAME Martin (First) Froilan (Middle) Maldonado (Last)

requests approval to prepare and present a thesis in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Information and Computer Science

Thesis Title: A Comparison of File Allocation Decision-Making Schemes

Brief description:

This thesis addresses the decision-making process for file allocation. Decentralized and centralized decision-making will be compared. In the first case, each node in the network decides independently the allocation of copies of a file, while in the second case, one node performs the decision. We expect to have tradeoffs between the cost of using the files and the cost of collection and/or disseminating information about file usage. It is suspected that decentralized decision-making might produce more costly allocations than the centralized procedure. However, the centralized decision-making process incurs additional costs in order to collect the required usage information and to divulge the decision. In general, this corresponds to the situation where we are forced to make tradeoffs between the optimality of a solution and the cost of obtaining that solution.

Approved:

Director of School

Signature of Graduate Student

Thesis Advisor

33124
Campus P. O. Box

Member Reading Committee or Thesis Advisory Committee

Dean, Office of Graduate Studies and Research

Prepare original only. Graduate Office will distribute copies.

Member Reading Committee or Thesis Advisory Committee

GEORGIA INSTITUTE OF TECHNOLOGY

Office of
Graduate Studies and Research

Date 11-19-84

REQUEST FOR APPROVAL OF THESIS TOPIC

NAME John (First) Alan (Middle) Miller (Last)

requests approval to prepare and present a thesis in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Thesis Title: Approximate Queueing Network Analysis of Database Systems

Brief description:

In this work we view a database system as a queueing network consisting of transactions operatings on data granules. Because the network turns out to be nonseparable (Lazo 84) (which usually implies intractability), we apply a mean substitution approximation. A nice property of this approximation is that it produces simple analytic models, both conceptually and computationally. We then apply this technique to analyze the performance of database systems. In particular we study the effect that database protocols have on performance measures such as throughput and response time. We compare the standard protocols found in the literature on the basis of these performance measures and consider the design of optimal protocols.

Approved:

Director of School

Thesis Advisor

Member Reading Committee or Thesis Advisory Committee

Member Reading Committee or Thesis Advisory Committee

Signature of Graduate Student

37000
Campus P. O. Box

Dean, Office of Graduate Studies and Research

Prepare original only. Graduate Office will distribute copies.

Design of Distributed Database Systems

Final Project Report
April 1, 1985

Nancy D. Griffeth
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

Publication Citations

Refereed Journals

Reliable Scheduling of Transactions on Unreliable Systems, ACM SIGACT/SIGMOD Symposium on Principles of Database Systems, April 2-4 1984, by Marc H. Graham, Nancy D. Griffeth, and Barbara Smith-Thomas.

Simulation of Concurrency Control and Recovery Protocols for Distributed Database Systems, IEEE Infocomm 84, Apr 9-12 1984, by Nancy D. Griffeth and Magdi Morsi.

Performance Modeling of Database Recovery Protocols, IEEE Symposium on Reliability in Distributed Software and Database Systems, Oct 15-17 1984, by Nancy D. Griffeth and John A. Miller; also, to appear in IEEE Transactions on Software Engineering, June 1985.

Optimal Placement of Identical Resources in a Tree, to appear in *Information and Control*, by Michael J. Fischer, Nancy D. Griffeth, and Nancy A. Lynch.

Probabilistic Analysis of a Network Resource Allocation Algorithm, to appear in *Information and Control*, by Nancy A. Lynch, Nancy D. Griffeth, Michael J. Fischer, and Leo J. Guibas.

Non-refereed Reports

A Simulation Tool for Distributed Databases, Georgia Tech Technical Report GIT-ICS-83/14, by Nancy D. Griffeth and Magdi Morsi.

Simulation of Concurrency Control and Recovery Protocols for Distributed Systems, Georgia Tech Technical Report GIT-ICS 83/16, by Nancy D. Griffeth and Magdi Morsi.

Recovery of Actions and Subactions in a Nested Transaction System, Georgia Tech Technical Report GIT-ICS-84/12, March 1984, by Marc H. Graham, Nancy D. Griffeth, and J. Eliot B. Moss.

Algorithms for Reliable Scheduling of Database Transactions, Georgia Tech Technical Report GIT-ICS-84/15, by Marc H. Graham, Nancy D. Griffeth, and Barbara Smith-Thomas.

File Migration Algorithms for Decentralized File Allocation, Georgia Tech Technical Report GIT-ICS-85/10, April 1985, by Nancy D. Griffeth and Martin A. Maldonado.

Two Approaches to Reducing the Overhead of Recovery from Transaction Failures, to appear in *Database Engineering*, by Nancy D. Griffeth.

Scientific Collaborators

Co-investigators

Michael J. Fischer
Professor
Yale University

Nancy A. Lynch
Associate Professor
Massachusetts Institute of Technology

Leo J. Guibas
Member of Technical Staff
Xerox Palo Alto Research Center

J. Eliot B. Moss
formerly Captain
U.S. Army
currently Assistant Professor
University of Massachusetts

Marc H. Graham
Assistant Professor
Georgia Institute of Technology

Barbara Smith-Thomas
Assistant Professor
University of North Carolina at Greensboro

Research Assistants and Students

John A. Miller
Research Assistant (funded by this grant)
Georgia Institute of Technology

Martin Maldonado
Graduate Student
Georgia Institute of Technology

Magdi Morsi
Research Assistant (funded by this grant)
Georgia Institute of Technology

Technical Description of Project and Results

Introduction

The focus of this research project is the behavior of distributed database systems in the presence of failures and concurrent access to the database. Several algorithms for reliable distributed database systems have been developed as part of the work. The techniques employed to study the behavior of the algorithms have included formal analysis of their properties; simulation studies of their performance; and a novel method of queuing network analysis, which can be used even when transactions are allowed to hold multiple resources simultaneously. The algorithms studied are: (1) five transaction scheduling algorithms (three are novel, to the best of our knowledge) which guarantee that each transaction either completes or has no effect on the database; (2) novel logging and roll-back algorithms which allow localization of the recovery effort to individual levels of abstraction; (3) a novel file migration algorithm for a distributed system which guarantees that at least one copy of the file is always available; and (4) a novel resource allocation algorithm (which could be applied to redundant copies of a file) which is amenable to expected cost analysis and which provides a very reasonable allocation of resources to requests. Also as part of this work, my colleagues and I have completed the study of some algorithms for determining an optimal placement of identical resources in a network structured as a tree and for allocating the resources to requests arriving in the tree.

The Algorithms and their Properties

Reliable Transaction Scheduling

Several scheduling protocols have been developed which guarantee "recoverable" execution of transactions in a distributed database system. These protocols are described in detail in GGST . Recoverable execution means that a transaction either runs to completion or has no effect on the database; in H it is shown that a necessary and sufficient condition for recoverability is that no transaction commit before any transaction on which it depends. Transaction operations are usually taken to be "read" and "write". A transaction which reads a data value then depends on the last transaction that wrote it. We call a data value dirty if it was written by a transaction which has not yet committed; any transaction which reads such a value is reading "dirty" data. (There are some authors who work with a more general class of operations, with dependency appropriately defined for such classes K , A , SS). The scheduling protocols described here apply to transactions which request read, write, commit, and abort operations; the scheduler itself may also introduce abort operations. These protocols are:

(1) the **optimistic** protocol, which blocks a transaction from committing until all transactions on which it depends have been committed (reads and writes are never blocked);
(2) the **pessimistic** protocol, which blocks reads and over-writes of dirty data;
(3) the **realistic** protocol, which blocks only reads of dirty data (multiple versions of written items must be maintained when dirty data is over-written, since blocked reads may be waiting on early versions);
(4) the **paranoid** protocol, which aborts any transaction which tries to read or over-write dirty data; and
(5) the **deferred write** protocol, similar to that used in optimistic concurrency control KR , which postpones all writes until a transaction requests a commit (if this protocol is used, serializability can only be enforced by checking for it at commit point).

Several questions were raised about the formal properties of these protocols. First, does the recovery protocol introduce aborts of transactions that would otherwise have committed? Second, what is the effect of a recovery protocol on the meaning of the schedule? In other words, suppose that a schedule is semantically correct, in that the interleaving of reads and writes gives a meaningful result (this would usually mean serializable). Will the recovery protocol change the meaning of the schedule so that we can no longer be sure that it is semantically correct? Third, if we know that serializability is the condition for a schedule to be semantically correct, then what is the effect of the recovery protocol on the serializability of

the schedule (whether it changes the meaning or not)?

Introduction of Aborts. The paranoid protocol introduces aborts to enforce recoverability. We would expect the largest number of transaction aborts from this protocol. The pessimistic and realistic protocols introduce them only to break deadlocks. It is interesting to note that if all reads are known to precede all writes in transactions, then no deadlock can occur using the realistic protocol and no aborts will be introduced. Finally, the optimistic protocol cascades aborts. One result of the performance studies described below was that aborts rarely cascaded. Crude analysis indicated that this was at least in part because all reads did precede all writes in the transactions used in the simulations. In this case, the probability of the first cascaded abort is small and the probability of the second is infinitesimal.

Meaning-preservation. The most obvious change to the meaning of operations comes with the deferred write protocol. Since this protocol postpones writes while letting reads proceed, the value which would have been read may not be available when the read is executed. The pessimistic and realistic protocols may also change the meaning of a read operation. This happens when the write on which a blocked read waits is rolled back because the transaction requesting the write has been aborted. The read must then access the previous value of the data rather than the one it was waiting on. The paranoid protocol changes the meaning of a read as a result of a different sequence of events. Suppose that transaction A writes a value in record 1 which will eventually be read by transaction B. Suppose also that A must be aborted by the paranoid protocol because, after writing record 1, it tries to read dirty data. Then the the meaning of B's read has been changed. In this case also, the read will access the previously written data value. The optimistic protocol does not rearrange operations in any way and will abort a transaction rather than change the meaning of any of its operations. Thus the meanings of the operations will be preserved.

Preservation of Serializability. Although serializability is preserved only by the optimistic protocol, the class of DSR schedules P is preserved by the optimistic, pessimistic, paranoid, and realistic protocols. Only deferred writes fails to preserve this class. Since this is the largest known class of schedules which is recognizable in polynomial time, and since all practical schedulers recognize only schedules in this class, we view DSR as the most important class to be preserved. It would seem to be a serious failing of the deferred write protocol that it does not preserve DSR.

Logging and Rollback

Most work on transaction systems has viewed the transactions as operating on a single level of abstraction. Additional knowledge of transaction semantics is savailable in multi-level systems. This knowledge can be used to reduce the scope of logging and procedures for rolling back (undoing) actions. Some examples and formal proofs can be found in GGM .

The central idea is suggested by a trick which can be applied to dynamic structures such as B-trees. Suppose that a transaction adds a record to an indexed relation, by adding the record to the relation and then adding the record key to the index. Suppose also that the transaction continues operating on the database and that other transactions which are executing concurrently cause changes to the index structure (for example, by splitting or coalescing nodes). If it eventually becomes necessary to abort the initial transaction, it might appear that it will also be necessary to roll back every action that changed the B-tree structure after the record key was added to the index. In this way, the B-tree structure is restored to its state at the time the key was added. As a result, concurrent operations on the B-tree would usually be prohibited. Of course, it is not really necessary to be this conservative. We can simply delete the key from the B-tree to rollback the operation that added the key. This is enough because we do not really care about the structure of the B-tree; we only care about the set of keys it contains.

In GGM , the ideas used in the B-tree trick have been formalized and can be applied in the general case. We assume multiple levels of abstraction. At each level of abstraction we define abstract actions which are implemented by state-dependent sequences of concrete actions. The concrete actions at one level are the abstract actions at the next lower level (except of course at the lowest level). We assume that with every action, we are supplied a collection of state-dependent undo actions. (The addition of a key to an index illustrates why undo actions must be state-dependent. If the key was already in the index, then the undo is the identity. If the key was not in the index, then the undo is a delete.)

If actions A and B, both at the same level of abstraction, do not conflict with each other, then it is possible to roll back action A without first rolling back a later action B. For recovery purposes, actions A and B conflict if undo(A), for the prior state of A, does not commute with B. (This definition is slightly different from the definition of conflict for the purpose of serializability, where actions A and B conflict if they do not commute.) To undo an action, we must first undo all later actions which conflict with it.

The situation is complicated somewhat if we must consider the concrete level as well. When an abstract action must be

rolled back before it has finished executing, that is, before completion of the sequence of concrete actions which implement it, then we cannot undo it at the abstract level but we can abort it. To abort it, we recursively abort incomplete concrete actions and roll back complete concrete actions in reverse order to their order of execution. (The concrete actions at the lowest level must be atomic to halt this recursion.) An abstract action depends on an earlier abstract action if it has a child (i.e., a concrete action implementing it) which conflicts with a child of the earlier abstract action. We must not abort an action before we abort all actions which depend on it, to guarantee that no action is undone before any later conflicting action has been undone. We formalize this intuition about the correctness of an abort in the following definition. A schedule is bf revokable if every undo action is the child of an abort action and no action is aborted before every action which depends on it has been aborted. This definition is symmetric to the definition of recoverability, that no action can commit until every action on which it depends has committed.

Using the above algorithm for rolling back has very nice implications for the size of the log. Once an abstract action has been completed, we can record its state-dependent undo and throw away the undo actions for the concrete actions implementing it. Thus the size of the log could be reduced considerably.

The above definitions assume that conflicting actions and dependent actions are ordered. It is possible to interleave the concrete actions implementing a collection of abstract actions in such a way that no reasonable order can be defined on the abstract actions. Such interleavings would be undesirable. The existence of the required order on the abstract actions can be guaranteed by a modified form of serializability. First, let us say that actions A and B conflict if either A or undo(A) does not commute with B. Next, let us say that a schedule of concrete actions implementing a schedule of abstract actions is serializable if the schedule can be transformed to a serial schedule by swapping non-conflicting actions. (This is a version of conflict-preserving serializability.) We require serializability at each level of abstraction. That is, if we treat the sequence of actions executed at a level as transactions and consider the schedule of concrete actions implementing them, the schedule is serializable. It follows from a result in BBGLS that this level-by-level form of serializability guarantees that the top-level state resulting from an execution is the same as the top-level state resulting from some serial execution. It also follows from this level-by level serializability that a partial order can be defined on the abstract actions at each level. We do this inductively: the partial order on the abstract actions at the lowest level is given by the depends on relation. It can be shown that if abstract actions A and B conflict, then so must at least one of their children, so that conflicting abstract actions will be ordered. Thus we can define the depends on relation at the next higher level.

A result in GGM establishes that the abstract state resulting from a history of operations which is level-by-level serializable, recoverable, and revokable will be the same as the abstract state resulting from the history after all aborted operations have been deleted. And from the above-cited result in BBGLS , it follows that the abstract state will be the same as that resulting from a serial execution of the actions which were not aborted.

The File Migration Algorithm

An algorithm described in GM3 guarantees reliable migration of files, by use of a token scheme in which a site having a token is prohibited from deleting its copy of the file. We envision a distributed system in which each site having a copy of the file may either send a new copy to another site or delete its own copy. The process of sending a new copy is controlled by a commit protocol, which guarantees that if both sites are "up" throughout the execution of the protocol then both sites will have a copy on completion of the protocol. Otherwise, the site originally having a copy will still have a copy, but the other site may or may not have a copy depending on when the failure occurred.

We assume that the file must be protected from any number of site failures less than a given number k. To guarantee this, we guarantee that there are always k copies of the file in the system (some of which may be at failed sites, but at least one of which is necessarily at a running site). This is achieved by giving tokens to k of the sites which have a copy of the file. No site having a token may delete its own copy of the file without first sending the token to a site having a copy of the file but no token. (If all sites having copies also have tokens, it will be necessary to make a new copy at a site not having one.) A commit protocol similar to the protocol for copying the file is used to send the token. If both sites are still up when the protocol is complete, then the token has been copied successfully and the sending site can now delete the token and the file.

There may eventually be more than k tokens in the system due to the indeterminate nature of the completion of the commit protocol if there is a failure while it is running. To eliminate them, each of the k tokens is given a unique ID and each recovering site is required to eliminate its token if a token of the same ID exists at a running site while it is recovering. No commit is required to accomplish this. Instead, if any such token is detected, it simply eliminates its own.

Two additional rules are required to guarantee that all running copies are up-to-date:

(1) A recovering site which has a token gets a copy from a running site and a recovering site which does not have a token throws away its copy;

(2) Every write operation writes available all copies of the file at sites having tokens.

The performance study of file allocation methods, which is discussed in the next section, assumes that the above algorithm is used to migrate files while the system is running. Use of such an algorithm makes decentralized control of file allocation,

using heuristics based only on the usage at individual sites, feasible. We hypothesize that under some circumstances, decentralized control of file allocation will be more effective than centralized.

Placement and Allocation of Resources

The placement algorithms described in FGL address the problem of locating a number t of identical resources (one example would be redundant copies of a file) at nodes of a tree so as to minimize the total expected cost of servicing a set of requests arriving randomly at nodes. The cost of servicing a single request is defined to be the tree distance from the requesting node to the node containing the resource satisfying the request. We bound the cost of optimal placements by finding simple placements whose total cost differs from optimality by at most the number of edges in the tree. For any fixed tree T, the cost of these placements grows as O(sqrt(t)), where the constant implicit in the "O" notation dpends on the size and shape of T. In the case of balanced trees with k leaves, that constant is at most sqrt(2k/pi).

Even though characterizing the exact cost of optimal placements appears to be difficult, we show that they can be found in time O(mt), where m is the number of edges in the tree. In the case of a complete (rooted) tree of degree d with a symmetric probability of request arrivals, an optimal placement can be found in time O(min l,log t +t) where l is the height of the tree. Moreover, the placement itself is symmetric. A whole placement (one in which an integral number of resources are placed at each node), can also be found in the same time.

In LGFG , an algorithm is described which allocates resources to requests as the requests arrive at random nodes of the tree. The central feature of the algorithm is its locality. The algorithm searches certain nearby nodes of the tree first before proceeding to search for resources at greater distances. In analyzing this algorithm, we were able to prove a difficult and surprising result about the expected cost of the allocations it produces. The allocation of resources occurs as requests actually arrive in the network and thus may not be optimal since information about future arrivals is not available. Surprisingly, however, the expected response time is a non-decreasing function of request interarrival time. Therefore, the worst case occurs when requests come in so far apart that they are processed sequentially. Analysis shows that the expected response time in this case is bounded by a constant, independent of the size of the network.

Work is ongoing to extend the methods of analysis used in the expected cost analysis to concurrency control and recovery algorithms for distributed systems.

Performance Studies

A system (SORCERER) for simulating distributed database algorithms has been designed, as described in GM2 . The novelty of SORCERER is that any component of the system may be simulated at varying levels of detail without altering other components of the system. For example, the validity of using probability distributions to model the resources utilized by a commit protocol, rather than encoding the protocol, may be tested directly. Database protocols may be encoded in SORCERER as *scripts*, which are a restricted form of procedure. This SORCERER combines flexibility both simplicity.

The simulation study of the scheduling protocols proposed in GGST has been carried out, as described in GM1 , with somewhat surprising results. In comparing the realistic and optimistic protocols we found that although in many cases the throughput for the optimistic protocol is slightly higher, it suffers more performance degradation on an unreliable, low capacity system. The pessimistic protocol had surprisingly poor performance. Although its throughput was quite good when there are a small number of writes compared to the number of reads, it was normally in a dead heat with the paranoid protocol. When there are many write operations, the pessimistic protocol is nearly an order of magnitude worse than the realistic protocol. Hence if we were to rank protocols in order of throughput, we would have to say that the realistic protocol edges out the optimistic protocol for first place, while the pessimistic and paranoid protocols are in a dead heat for a distant last place. We might expect that we would pay for the throughput of the realistic protocol with the extra space for multiple versions of written data values. In comparison with the pessimistic protocol, this is not so: the queues of blocked writes, in the pessimistic protocol, will require about the same amount of space as the multiple versions in the realistic protocol.

A queuing network model of the execution of database transactions has been studied and validated against the simulation. The most interesting feature of this model is that it can be used even when transactions (i.e., the customers) must hold multiple resources simultaneously. It is shown in GM1 that the only assumptions necessary to use this analysis are (1) that the number of locks held and the waiting time of a blocked transaction can be closely approximated by their means and (2) that the service rate increases linearly with the number of transactions. The result gives a close approximation to the steady state of the system, where the state is a vector $n1,\ldots,nk,b1,\ldots,bk$, ni is the number of transactions executing their ith operation, and bi is the number of transactions blocked before their ith operation. From the steady state, we can compute all of the interesting measures such as throughput, number of blocked transactions, number of locks held, space required for queues or versions, and so forth.

Finally, a simulation study has been designed to compare decentralized to centralized control of file allocation. A single central site which has full knowledge of usage of the copies of a file can often come close to an optimum allocation, but the heuristics for doing so are relatively expensive. Furthermore, the central site incurs extra communication costs when it gathers usage information and controls file movement. while the system is running. The latter method is made possible by the algorithms described above for dynamic migration of files in a distributed system.

Bibliography

A Allchin, James E. *An Architecture for Reliable Distributed Systems*, Ph. D. dissertation, Georgia Tech Technical Report GIT-ICS-82-23, 1983.

BBGLS Beeri, C., P.A. Bernstein, N. Goodman, M. Y. Lai, and D. Shasha. "A Concurrency Control Theory for Nested Transactions", *Proceedings of the 1983 ACM SIGACT/SIGOPS Symposium on Principles of Distributed Computing*, August 1983.

FGL Fischer, Michael J., Nancy D. Griffeth, and Nancy A. Lynch. "Optimal Placement of Identical Resources in a Tree", to appear in *Information and Control*.

GGM Graham, Marc H., Nancy D. Griffeth, and J. Eliot B. Moss. "Recovery of Actions and Subactions in a Nested Transaction System", Georgia Tech Technical Report GIT-ICS-84/12, March 1984.

GGST Graham, Marc H., Nancy D. Griffeth, and Barbara Smith Thomas. Reliable Scheduling of Transactions on Unreliable Systems, *Proceedings of the ACM-SIGACT/SIGMOD Symposium on Principles of Database Systems*, April 1984.

G Griffeth, Nancy D. "Two Approaches to Reducing the Overhead of Recovery from Transaction Failures", to appear in *Database Engineering*.

GM3 Griffeth, Nancy D., and Martin Maldonado. "File Migration Algorithms for Decentralized File Allocation", Georgia Tech Technical Report GIT-ICS-85/10, April 1985.

GM1 Griffeth, Nancy D. and John A. Miller. "Performance Modeling of Database Recovery Protocols", *Proceedings of the IEEE Symposium on Reliability in Distributed Software and Database Systems*, Oct 15-17 1984; also, to appear in *IEEE Transactions on Software Engineering*, June 1985.

GM2 Griffeth, Nancy D., and Magdi Morsi. "Simulation of Concurrency Control and Recovery Protocols for Distributed Database Systems", *Proceedings of IEEE Infocomm 84*, Apr 9-12 1984. H Hadzilacos, Vassos. "An Operational Model of Database System Reliability", *Proceedings of the 1983 SIGACT/SIGOPS Symposium on Principles of Distributed Computing*, August 1983.

K Korth, H. F. "Locking primitives in a Database System", *Journal of the ACM*, vol. 30 no. 1, January 1983.

KR Kung, H. T., and J. T. Robinson. "On Optimistic Methods for Concurrency Control", *ACM Transactions on Database Systems*, Vol. 6 No. 2, June 1981.

LGFG Lynch, Nancy A., Nancy D. Griffeth, Michael J. Fischer,

and Leo J. Guibas. "Probabilistic Analysis of a Network Resource Allocation Algorithm", to appear in *Information and Control*.

P Papadimitriou, Cristos H. "The Serializability of Concurrent Database Updates", *Journal of the* ACM, vol. 26 no. 4, 1979.

SS Schwartz, Peter M., and Alfred Z. Spector. "Synchronizing Shared Abstract Types", ACM *Transactions on Computer Systems*, vol. 2 no. 3, August 1984.