Old Dominion University Research Foundation

DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCES
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529

BUILDING A GENERALIZED DISTRIBUTED SYSTEM MODEL

By

Ravi Mukkamala, Principal Investigator

Progress Report
For the period ended July 31, 1991

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Under
Research Grant NAG-1-1114
Wayne H. Bryant, Technical Monitor
ISD-Systems Architecture Branch

May 1991

DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCES
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529

BUILDING A GENERALIZED DISTRIBUTED SYSTEM MODEL

By

Ravi Mukkamala, Principal Investigator

Progress Report
For the period ended July 31, 1991

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Under
Research Grant NAG-1-1114
Wayne H. Bryant, Technical Monitor
ISD-Systems Architecture Branch

Submitted by the
Old Dominion University Research Foundation
P.O. Box 6369
Norfolk, Virginia 23508-0369

May 1991

Building a Generalized Distributed System Model

R. Mukkamala
E.C. Foudriat

Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529.

Annual Report and Renewal Request

Abstract

The key elements in the first year (1990-91) of our project were:

- Investigate the effects of modeling on distributed system performance predictions.
- Look at possible graphical interfaces to the proposed distributed prototype and simulator system.
- Conduct preliminary studies towards the design of a generalized distributed system.

In the second year of the project (1991-92), we propose to

- Develop detailed designs for the prototype.
- Implement and test the system.
- Conduct further studies on modeling distributed systems.

1 Introduction

In the 1990-91 proposal, we discussed the need for building a modeling tool for both analysis and design of distributed systems. To this end, we have been considering different design architectures for the modeling tool. Since many of the research institutions have access to networks of workstations, we have decided to build a tool running on top of the workstations to function as a prototype as well as a distributed simulator for a computing system.

In addition, we have been investigating the effects of system modeling on performance prediction in distributed systems. While some performance measures such as the average number of participating node set size of a distributed transaction is not very sensitive to the underlying model, measures such as transaction commutativity measures are quite sensitive to the evaluation models.

We have also considered the effects of static locking and deadlocks on the performance predictions of distributed transactions. While the probability of deadlock is considerably small in a typical distributed system, its effects on performance could be significant.

In this report, we summarize our progress in these three areas and describe the details of the proposed work.

2 Distributed System Model: Prototype/Simulator

The main goals of our efforts in building a general tool for simulation and prototyping of distributed systems are:

- A framework to experiment with distributed algorithms/systems.
- Implement in terms of basic primitives (e.g., RPC, reliable communication).
- A good user interface - preferably with graphic and mouse functions.
- Provisions to include user specific code for different components.
- A library of procedures representing typical options for components (e.g. two-phase locking).
- A base for distributed simulation as well as prototyping.
- Efficient mechanisms to monitor and display the activities.

- Powerful performance analysis tools.

To this end, we started looking at a transaction oriented distributed system. Since our aim is to provide a general framework rather than to provide a solution to a particular model, our goal is to provide some of the basic primitives at the bottom layer, and let the user build the needed upper level software. To make the prototype usable for a novice user, we propose to provide a graphic interface through which a user can specify the system configuration. As an example application, we considered distributed database system modeling. As shown in Figure 1, we identified seven major components. Each of these components can be further described in a detailed model. For example, the local manager can be modeled as a coordinator of local concurrency control manager and the transaction resource manager. Given a set of components, the control structure of the system can be represented through directional links. Figure 2 illustrates one such control structure.

After considering several alternates, we decided to base the graphic interface on the lines of the MIT Network simulator. The MIT simulator is developed at Massachusetts Institute of Technology with funding from DARPA. Even though it is intended for simulating communication networks, we have decided to adopt its graphic interfacing routines for our distributed simulator. Since the source code (in C) is available, we are modifying this code to suit our needs. Some of its distinguishing characteristics of the network simulator are:

- Internetwork simulator
- Components include gateways, network links, hosts, TCPs and users.
- Network configuration is displayed on the screen.
- User can control the simulation.
- Network configuration can be modified with the mouse.
- Other simulation parameters can be changed on-line using the mouse.
- Network configuration can be saved for later use.
- Several performance measures may be printed.

Figure 1

Distributed System Model

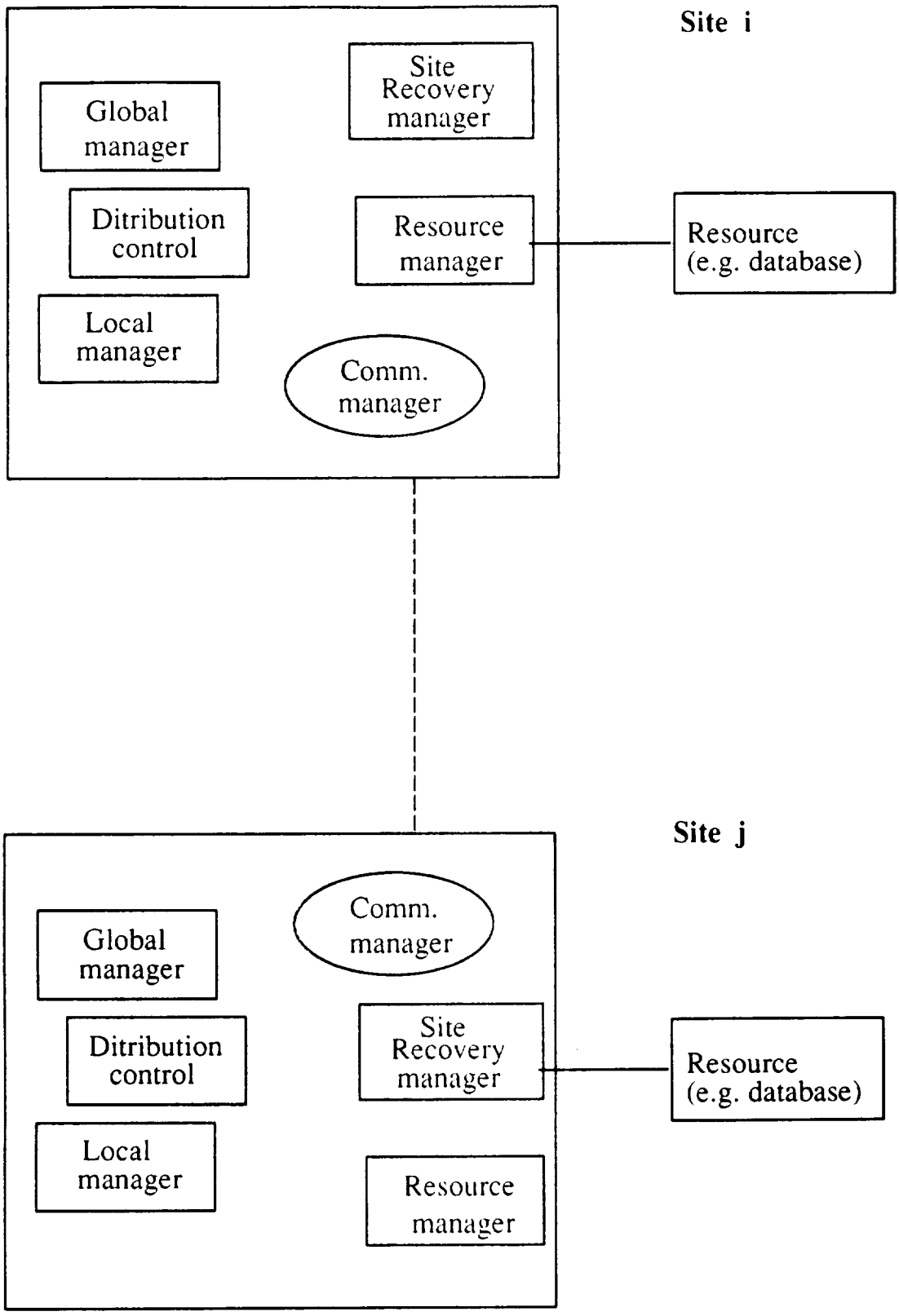
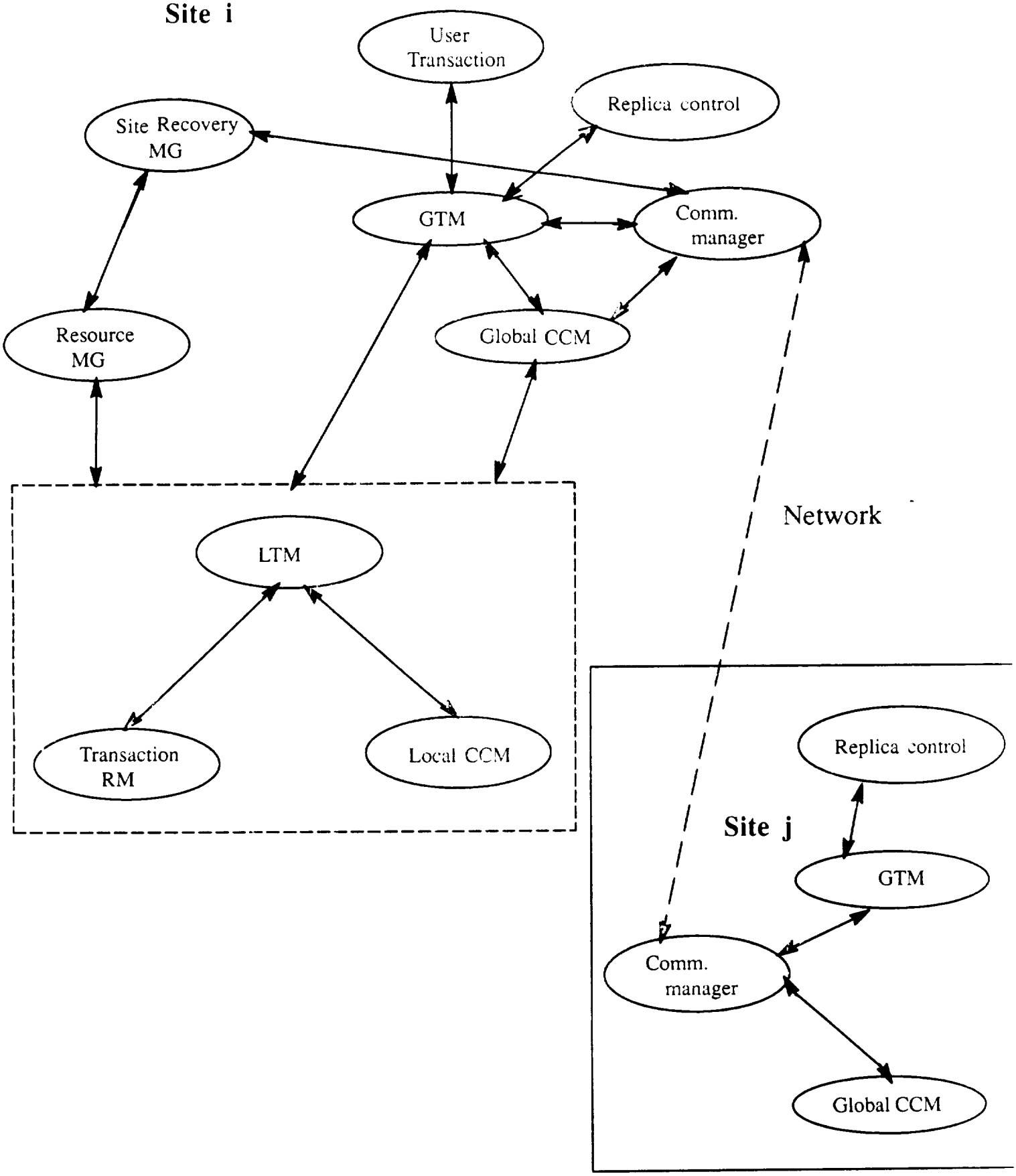


Figure 2


Site i
User Transaction
Replica control
Site Recovery MG
GTM
Comm. manager
Resource MG
Global CCM
LTM
Transaction RM
Local CCM
Network
Site j
Replica control
GTM
Comm. manager
Global CCM

Since process communication is a basic primitive needed in distributed systems, we have decided to provide this as a basic mechanism in our system. Currently, we are experimenting with the Sun RPC system calls to design a high-level primitive. RPC has several advantages including:

- Hiding details of network programming
- Availability of library routines
- Hiding the operating system dependencies
- Availability of the standard data representation using XDR format which allows a simple way of transferring data.

3 Effects of Modeling on Performance Predictions

As a second part of our study, we have conducted investigations to determine the impact of modeling on distributed system performance. Here, we summarize the results of two such studies:

Study 1: Effect of Data Distribution Models on Transaction Commutativity [2]. Recognizing commutativity among transactions appears to reduce the number of rollbacks (at the time of merge) in a partitioned distributed database system [1]. The main objective of this study is to determine the impact of data distribution modeling on the evaluation of the benefits due to commutativity. We studied the effects of six distinct data distribution models on the evaluation of the number of rollbacks. We derived closed form expressions for five of the six models, and used simulation for the sixth model. The conclusions from this study are summarized as follows.

- Random data models that assume only average information about the system result in conservative estimates of system throughput.
- Adding more system information does not necessarily lead to better approximations. In this paper, the system information is increased from model 6 to model 2. Even though this increases the computational complexity, it does not result in any significant improvement in the estimation of the number of rollbacks.

- Transaction commutativity appears to significantly reduce transaction rollbacks in a partitioned distributed database system. *This fact is only evident from the analysis of model 1.* On the other hand, when we look at models 2-6, it is possible to conclude that commutativity is not helpful unless it is extremely high. Thus, conclusions from model 1 and models 2-6 are *contradictory.*

- The replication distribution (i.e., the actual number of copies for each object) seems to effect the evaluations significantly. Thus, accurate modeling of this distribution is vital to evaluation of rollbacks.

Study 2: Effect of Data Distribution and Reliability Models on Transaction Availability [3]. In this study, we selected three abstractions for data distribution modeling and three for node reliability modeling, and constructed six system models. Here, transaction availability is defined as the probability with which all data copies required by a transaction are available at the beginning of its execution. As before, we could derive closed form expressions with five of the six models (using probabilistic analysis), and used simulation for the other model. A transaction was characterized by the number of data objects that it accesses, s. The conclusions derived from this study are summarized as follows.

- By choosing a proper distributed database model, the computational complexity of transaction availability evaluations can be significantly reduced.

- For values of $s \leq 10$, all models result in almost the same transaction evaluation.

- The degree of replication of individual (or group) data objects seems to have a significant effect on transaction availabilities. Thus, when different data objects have different copies, adopting average degree of replication at the system level may not result in accurate availability evaluations.

- The actual distribution of data object copies has some, if not significant, impact on availability evaluation.

- In a heterogeneous environment where different nodes may have different reliabilities, it is sufficient to represent each node by the average node reliability, without affecting the availability evaluations.

Having conducted these studies, we conclude that

- Adopting simple models may drastically reduce the complexity of metric evaluations.
- Choosing analytically tractable models enables easy interpretation of functional dependencies.
- By choosing inappropriate models, for either analytical tractability or conceptual simplicity, it is possible to arrive at incorrect conclusions.
- Model choice is highly dependent on the metric. While a simple model serves well for one metric, it may be insufficient for another metric.

4 Determining the Effects of Locking on Distributed Transactions

Deadlocks are known to deteriorate performance in both centralized and distributed database systems [4,5]. In spite of this, several performance studies have ignored the deadlock problem in their analyses [6]. In [4], Shyu and Li proposed an elegant technique to evaluate the response time and throughput of transactions in a non-replicated DDS. Assuming *exclusive* locking (i.e., only write operations), they model the queue of lock requests at an object as a M/M/1 queue. This results in a closed-form for the waiting time distribution at a node, expressed in terms of the average rates of arrivals of requests and the average lock-holding time.

In general, a database transaction reads from a set of data objects (the read-set) and writes on to a set of data objects (the write-set). In this paper, we consider both the the read and the write operations of database transactions, and propose a technique for performance evaluation.

We make the following observations from evaluations made with our technique.

- As expected, the presence of shared locks has a substantial impact on the probability of deadlock occurrence. When only 1/3 of the accessed data objects are updated, the probability of deadlock is considerably small as compared to when all objects are updated.
- The observations about the deadlock probabilities are also valid for restart probabilities.

- Transaction response times are also quite sensitive to the ratio of shared locks. Here, we compare the response times when deadlocks are ignored with those obtained when deadlocks are considered. The effect of deadlocks is more predominant at higher transaction loads and with smaller values of read ratio. When 1/3 of the accessed objects are updated, the effect of deadlocks is not significant on response time.

- The effect of deadlocks on response time is decreased with the increase in the number of data items. Obviously, this is due to the decrease in probability of conflicts and hence a decrease in deadlock occurrence. When only 1/3 of the accessed data are updated, this effect is almost insignificant. When 2/3 of the accessed data are updated, deadlocks seems to have a noticeable effect on response time.

- When a small number of data objects are accessed, the probability of deadlock is negligible, and hence its effect on response time is small. When more data objects are accessed, the effect of deadlocks on response times is significant.

5 Summary of Accomplishments in 1990-91

We have published the results of our research (since August 1990) in two conferences. In addition, two papers are submitted for publication in international journals. These are:

1. Y. Kuang and R. Mukkamala, "Performance Analysis of Static Locking in Replicated Distributed Database Systems," *Proc. Southeastcon 1991*, pp. 698-701.

2. Y. Kuang and R. Mukkamala, "A Note on the Performance Analysis of Static Locking in Distributed Database Systems", Submitted to IEEE Trans. Computers, December 1990.

3. R. Mukkamala, "Effects of Distributed Database Modeling on Evaluation of Transaction Rollbacks," *Proc. WSC'91*, December 1990, pp. 839-845.

4. R. Mukkamala, "Measuring the Effects of Distributed Database Models On Transaction Availability Measures," Submitted to Performance Evaluation Journal, March 1991.

In addition, our current work on building the prototype for a distributed system should result in several conference and journal papers in 1991-92.

6 Proposed Research Efforts in 1991-92

During the next grant period (August 1991 to July 1992), we propose to continue the study and development of the distributed prototyping and simulator system. The main main problems that we need to solve in this period are:

- Complete the graphic interface design and implement it on Sun workstations.
- Investigate efficient means of offering flexible as well as efficient means of specifying interfacing between system components. We expect this phase to consume considerable time.
- Design, build, and test a specific system using the primitives offered by the system. Experiences from building a specific system should aid us in developing a generalized prototyping tool.
- We propose to use the prototype to evaluate the performance of several distributed mutual exclusion policies. Such a study may result in the development of new policies.
- We propose to do further investigations in modeling of distributed systems and determine their impact on predictive analysis tools.

References

[1] S. Jajodia and P. Speckman, "Reduction of conflicts in partitioned databases," Proc. 19th Annual Con. on Information Sciences and Systems, 1985, pp. 349-335.

[2] R. Mukkamala, "Effects of Distributed Database Modeling on Evaluation of Transaction Rollbacks," *Proc. WSC'91*, December 1990, pp. 839-845.

[3] R. Mukkamala, "Measuring the Effects of Distributed Database Models On Transaction Availability Measures," Submitted for publication, March 1991.

[4] S.-C. Shyu and V. O. K. Li, "Performance analysis of static locking in distributed database systems," *IEEE Trans. Computers*, vol. 39, no. 6, pp. 741-751, June 1990.

[5] Y. C. Tay, R. Suri, and N. Goodman, "A mean value performance model for locking in databases: The no-waiting case," *J. ACM*, vol. 32, no. 3, pp. 618-651, July 1985.

[6] M. Singhal and A. K. Agrawala, "Performance analysis of an algorithm for concurrency control in replicated database systems," *Proc. ACM SIGMETRICS Conf. Measurement Modeling Comput. Syst.*, 1986, pp. 159-169.

APPENDIX

Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

EFFECTS OF DISTRIBUTED DATABASE MODELING ON EVALUATION OF TRANSACTION ROLLBACKS

Ravi Mukkamala

Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529-0162.

ABSTRACT

Data distribution, degree of data replication, and transaction access patterns are key factors in determining the performance of distributed database systems. In order to simplify the evaluation of performance measures, database designers and researchers tend to make simplistic assumptions about the system. In this paper, we investigate the effect of modeling assumptions on the evaluation of one such measure, the number of transaction rollbacks, in a partitioned distributed database system. We develop six probabilistic models and develop expressions for the number of rollbacks under each of these models. Essentially, the models differ in terms of the available system information. The analytical results so obtained are compared to results from simulation. From here, we conclude that most of the probabilistic models yield overly conservative estimates of the number of rollbacks. The effect of transaction commutativity on system throughput is also grossly undermined when such models are employed.

1. INTRODUCTION

A distributed database system is a collection of cooperating nodes each containing a set of data items (In this paper, the basic unit of access in a database is referred to as a data item.). A user transaction can enter such a system at any of these nodes. The receiving node, sometimes referred to as the *coordinating* or *initiating* node, undertakes the task of locating the nodes that contain the data items required by a transaction.

A partitioning of a distributed database (DDB) occurs when the nodes in the network split into groups of communicating nodes due to node or communication link failures. The nodes in each group can communicate with each other, but no node in one group is able to communicate with nodes in other groups. We refer to each such group as a *partition*. The algorithms which allow a partitioned DDB to continue functioning generally fall into one of two classes [Davidson et al. 1985]. Those in the first class take a *pessimistic* approach and process only those transactions in a partition which do not conflict with transactions in other partitions, assuring mutual consistency of data when partitions are reunited. The algorithms in the second class allow every group of nodes in a partitioned DDB to perform new updates. Since this may result in independent updates to items in different partitions, conflicts among transactions are bound to occur, and the databases of the partitions will clearly diverge. Therefore, they require a strategy for conflict detection and resolution. Usually, rollbacks are used as a means for preserving consistency; conflicting transactions are rolled back when partitions are reunited. Since coordinating the undoing of transactions is a very difficult task, these methods are called *optimistic* since they are useful primarily in a situation where the number of items in a particular database is large and the probability of conflicts among transactions is small.

In general, determining if a transaction that successfully executed in a partition is rolled back at the time the database is merged depends on a number of factors. Data items in the read-set and the write-set of the transaction, the distribution of these data items among the other partitions, access patterns of transactions in other partitions, data dependencies among the transactions, and semantic relation (if any) between these transactions are some examples of these factors. Exact evaluation of rollback probability for all transactions in a database (and hence the evaluation of the number of rolled back transactions) generally involves both analysis and simulation, and requires large execution times [Davidson 1982; Davidson 1984]. To overcome the computational complexities of evaluation, designers and researchers generally resort to approximation techniques [Davidson 1982; Davidson 1986; Wright 1983a; Wright 1983b]. These techniques reduce the computation time by making simplifying assumptions to represent the underlying distributed system. The time complexity of the resulting techniques greatly depends on the assumed model as well as evaluation techniques.

In this paper we are interested in determining the effect of the distributed database models on the computational complexity and accuracy of the rollback statistics in a partitioned database.

The balance of this paper is outlined as follows. Section 2 formally defines the problem under consideration. In Section 3, we discuss the data distribution, replication, and transaction modeling. Section 4 derives the rollback statistics for one distribution model. In Section 5, we compare the analysis methods for six models and simulation method for one model based on computational complexity, space complexity, and accuracy of the measure. Finally, in Section 6, we summarize the obtained results.

2. PROBLEM DESCRIPTION

Even though a transaction T_1 in partition P_1 may be rolled back (at merging time) by another transaction T_2 in partition P_2 due to a number of reasons, the following two cases are found to be the major contributors [Davidson 1982].

i. $P_1 \neq P_2$, and there is at least one data item which is updated by both T_1 and T_2. This is referred to as a *write-write* conflict.

ii. $P_1 = P_2$, T_2 is rolled back, and it is a *dependency parent* of T_1 (i.e., T_1 has read at least one data item updated by T_2, and T_2 occurs prior to T_1 in the serialization sequence).

The above discussion on reasons for rollback only considers the syntax of transactions (i.e. read- and write-sets) and does not recognize any semantic relation between them. To be more specific, let us consider transactions T_1 and T_2 executed in two different partitions P_1 and P_2 respectively. Let us also assume that the intersection between the write-sets of T_1 and T_2 is non-empty. Clearly, by the above definition, there is a write-write conflict and one of the two transactions has to be rolled back. However, if T_1 and T_2 commute with each other, then there is no need to rollback either of the transactions at the time of partition merge [Garcia-Molina 1983; Jajodia and Speckman 1985; Jajodia and Mukkamala 1990]. Instead, T_1 needs to be executed in P_2 and T_2 needs to be executed in P_1. The analysis in this paper take this property into account.

In order to compute the number of rollbacks, it is also necessary to define some ordering ($O(P)$) on the partitions. For example, if T_1 and T_2 correspond to case (i) above, and do not commute, it is necessary to determine which of these two are rolled back at the time of merging. Partition ordering resolves this ambiguity by the following rule: Whenever two conflicting but non-commuting transactions are executed in two different partitions, then the transaction executed in the lower order partition is rolled back.

Since a transaction may be rolled back due to either (i) or (ii), we classify the rollbacks into two classes: Class 1 and Class 2 respectively. The problem of estimating the number of rollbacks at the time of partition merging in a partially replicated distributed database system may be formulated as follows.

Given the following parameters, determine the number of rolled back transactions in class 1 (R_1) and class 2 (R_2).

- n, the number of nodes in the database;
- d, the number of data items in the database;
- p, the number of partitions in the distributed system (prior to merge);
- t, the number of transaction types;
- GD, the global data directory that contains the location of each of the d data items; the GD matrix has d rows and n columns, each of which is either 0 or 1;
- NS_k, the set of nodes in partition k, $\forall k = 1, 2, \ldots, p$;
- RS_j, the read-set of transaction type j, $j = 1, 2, \ldots, t$;
- WS_j, the write-set of transaction type j, $j = 1, 2, \ldots, t$;
- N_{jk}, the number of transactions of type j received in partition k (prior to merge), $j = 1, 2, \ldots, t$, $k = 1, 2, \ldots, p$.
- CM, the commutativity matrix that defines transaction commutativity. If $CM_{j_1 j_2} = true$ then transaction types j_1 and j_2 commute. Otherwise they do not commute.

The average number of total rollbacks is now expressed as $R = R_1 + R_2$.

3. MODEL DESCRIPTION

As stated in the introduction, the primary objective of this paper is to investigate the effect of data distribution, replication, and transaction models on estimation of the number of rollbacks in a distributed database system.

To describe a data distribution-transaction model, we characterize it with three orthogonal parameters:

1. Degree of data item replication (or the number of copies).
2. Distribution of data item copies.
3. Transaction characterization

We now discuss each of these parameters in detail.

For simplicity, several analysis techniques assume that each data item has the same number of copies (or degree of replication) in the database system [Coffman et al. 1981]. Some other techniques characterize the degree of replication of a database by the average degree of replication of data items in that database [Davidson 1986]. Others treat the degree of replication of each data item independently.

Some designers and analysts assume some specific allocation schemes for data item (or group) copies (e.g., [Mukkamala 1987]). Assuming complete knowledge of data copy distribution (GD) is one such assumption. Depending on the type of allocation, such assumptions may simplify the performance analysis. Others assume that each data item copy is randomly distributed among the nodes in the distributed system [Davidson 1986].

Many database analysts characterize a transaction by the size of its read-set and its write-set. Since different transactions may have different sizes, these are either classified based on the sizes, or an average read-set size and average write-set size are used to represent a transaction. Others, however, classify transactions based on the data items that they access (and not necessarily on their size). In this case, transaction types are identified with their expected sizes and the group of data items from which these are accessed. An extreme example is a case where each transaction in the system is identified completely by its read-set and its write-set.

With these three parameters, we can describe a number of models. Due to the limited space, we chose to present the results for six of these models in this paper.

We chose the following six models based on their applicability in the current literature, and their close resemblance to practical systems. In all these models, the rate of arrival of transactions at each of the nodes is assumed to be completely known a priori. We also assume complete knowledge of the partitions (i.e. which nodes are in which partitions) in all the models.

Model 1: Among the six chosen models, this has the maximum information about data distribution, replication, and transactions in the system. It captures the following information.

- *Replication:* Data replication is specified for each data item.
- *Data distribution:* The distribution of data items among the nodes in the system is represented as a distribution matrix (as described in Section 2).
- *Transactions:* All distinct transactions executed in a system are represented by their read-sets and write-sets. Thus, for a given transaction, the model knows which data items are read, and which data items are updated. The commutativity information is also completely known and is expressed as a matrix (as described in Section 2).

Model 2: This model reduces the number of transactions by combining them into a set of transaction types based on commutativity, commonalities in data access patterns, etc. Since the transactions are now grouped, some of the individual characteristics of transactions (e.g. the exact read-set and writes-set) are lost. This model has the following information.

- *Replication:* Average degree of replication is specified at the system level.
- *Data distribution:* Since the read- and write-set information is not retained for each transaction type, the data distribution information is also summarized in terms of average data items. It is assumed that the data copies are allocated randomly to the nodes in the system.
- *Transactions:* A transaction type is represented by its read-set size, write-set size, and the number of data items from which selection for read and write is made. Since two transaction types might access the same data item, it also stores this overlap information for every pair of transaction types. The commutativity information is stored for each pair of transaction types.

Model 3: This model further reduce the transaction types by grouping them based only on commutativity characteristics. No consideration is given to commonalities in data access pattern or differing read-set and write-set sizes. It has the following information.

- *Replication:* Average degree of replication is specified at the system level.
- *Data distribution:* As in model 2, it is assumed that the data copies are allocated randomly to the nodes in the system.
- *Transactions:* A transaction type is represented by the average read-set size and average write-set size. The commutativity information is stored for all pairs of transaction types.

Model 4: This model classifies transactions into three types: read-only, read-write, and others. Read-only trans-

actions commute among themselves. Read-write transactions neither commute among themselves nor commute with others. The others class corresponds to update transactions that may or may not commute with transactions in their own class. This fact is represented by a commute probability assigned to it.

- *Replication:* Average degree of replication is specified at the system level.
- *Data distribution:* As in model 2, it is assumed that the data copies are allocated randomly to the nodes in the system.
- *Transactions:* Read-only class is represented by average read-set size. The read-write class is represented by average read-set and write-set sizes. The others class is represented by the average read-set size, average write-set size and the probability of commutation.

Model 5: This model reduces the transactions to two classes: read-only and read-write. Read-only transactions commute among themselves. The read-write transactions corresponds to update transactions that may or may not commute with transactions in their own class. This fact is represented by a commute probability assigned to it.

- *Replication:* Average degree of replication is specified at the system level.
- *Data distribution:* As in model 2, it is assumed that the data copies are allocated randomly to the nodes in the system.
- *Transactions:* Read-only class is represented by average read-set size. The read-write class is represented by average read-set and write-set sizes, and the probability of commutation.

Model 6: This model identifies read-only transactions and other update transactions. But these two types have the same average read-set size. Update transactions may or may not commute with other update transactions.

- *Replication:* Average degree of replication is specified at the system level.
- *Data distribution:* As in model 2, it is assumed that the data copies are allocated randomly to the nodes in the system.
- *Transactions:* The read-set size of a transaction is denoted by its average. For update transactions, we also associate an average write-set size and the probability of commutation.

Among these, model 1 is very general, and assumes complete information of data distribution (GD), replication, and transactions. Other models assume only partial (or average) information about data distribution and replication. Model 1 has the most information and model 6 has the least.

4. COMPUTATION OF THE AVERAGES

Several approaches offer potential for computing the average number of rollbacks for a given system environment; the most prominent methods are simulation and probabilistic analysis.

Using simulation, one can generate the data distribution matrix (GD) based on the data distribution and replication policies of the given model. Similarly, one can generate different transactions (of different types) that can be received at the nodes in the network. Since the partition information is completely specified, by searching the relevant columns of the GD matrix, it is possible to determine whether a given transaction has been successfully executed in a given partition. Once all the successful transactions have been identified, and their data dependencies are identified, it is possible to identify the transactions that need to be rolled back at the time of merging. The generation and evaluation process may have to be repeated enough number of times to get the required confidence in the final result.

Probabilistic analysis is especially useful when interest is confined to deriving the average behavior of a system from a given model. Generally, it requires less computation time. In this paper, we present detailed analysis for model 6, and a summary of the analysis for models 1-5.

4.1 Derivations for Model 6

This model considers only two transaction types: read-only (Type 1) and read-write (Type 2). Both have the same average read-set size of r. A read-write transaction updates w of the data items that it reads. N_{1k} and N_{2k} represent the rate of arrival of types 1 and 2 respectively at partition k. The average degree of replication of a data item is given as c. The system has n nodes and d data items. The probability that two read-write transaction commute is m.

Let us consider an arbitrary transaction T_i received at one of the nodes in partition k with n_k nodes. Since the copies of a data item are randomly distributed among the n nodes, the probability that a single data item is accessible in partition k is given by

$$\alpha_k = 1 - \frac{\binom{n-n_k}{c}}{\binom{n}{c}} \tag{1}$$

Since each data item is independently allocated, the expected number of data items available in this partition is $d\alpha_k$. Similarly, since T_i accesses r data items (on the average), the probability that it will be successfully executed is α_k^r. From here, the number of successful transactions in k is estimated as $\alpha_k^r N_{1k}$ and $\alpha_k^r N_{2k}$ for types 1 and respectively.

In computing the probability of rollback of T_i due to case (i), we are only interested in transactions that update a data item in the write-set of T_i and not commuting with T_i. The probability that a given data item (updated by T_i) is not updated in another partition k' by a non-commuting transaction (with respect to T_i) is given by

$$\beta_{k'} = \left(1 - \frac{w}{d\alpha_{k'}}\right)^{(1-m)\alpha_{k'}^r N_{2k'}} \tag{2}$$

Given that a data item is available in k, probability that it is not available in k' is given as

$$\gamma(k,k') = \frac{\binom{n-n_{k'}}{c} - \binom{n-n_k-n_{k'}}{c}}{\alpha_k \binom{n}{c}} \tag{3}$$

From here, the probability that a data item available in k is not updated any other transaction in higher order partitions is given as

$$\delta_k = \prod_{\forall k', O(k')>O(k)} \left[\gamma(k,k') + (1-\gamma(k,k'))\,\beta_{k'}\right] \tag{4}$$

The probability that transaction T_i is not in write-write conflict with any other non-commuting transaction of higher-order partitions is now given as

$$\mu_k = \frac{\binom{d\alpha_k\delta_k}{w}}{\binom{d\alpha_k}{w}} \tag{5}$$

From here, the number of transactions rolled back due to category (i) may be expressed as $H_1 = \sum_{k=1}^{P}(1-\mu_k)\alpha_k^r N_{2k}$

To compute the rollbacks of category (ii), we need to determine the probability that T_i is rolled back due to the rollback of a dependency parent in the same partition. If T_2 is a read-write transaction in partition k, then the probability that T_1 depends on T_2 (i.e. read-write conflict) is given by:

$$\lambda_k = 1 - \frac{\binom{d\alpha_k - w}{r}}{\binom{d\alpha_k}{r}} \quad (6)$$

The probability that T_i is not rolled back due to the roll back of any of its dependency parents is now given by:

$$\lambda_k = \sum_{i=1}^{\alpha_k^r N_k} \frac{(\lambda_k \mu_k + 1 - \lambda_k)^m}{\alpha_k^r N_k} \quad (7)$$

where $N_k = N_{1k} + N_{2k}$ and $u = N_{2k}/(N_{1k} + N_{2k})$.

The total number of rolled back transactions due to category (ii) is now estimated as $R_2 = \sum_{k=1}^{p}(1 - \lambda_k)\alpha_k^r(N_{1k} + \mu_k N_{2k})$. The total number of rolled back transactions is $R = R_1 + R_2$.

5. COMPARISON OF THE MODELS

As mentioned in the introduction, the main objective of this paper is to determine the effect of data distribution, replication, and transaction models on the estimation of rollbacks. To achieve this, we evaluate the desired measure using six different data distribution and replication models. The comparison of these evaluations is based on computational time, storage requirement, and the average values obtained.

Due to the limited space, we could not present the detailed derivations for the average values for models 2-6. The final expressions, however, are presented in [Mukkamala 1990].

5.1 Computational Complexity

We now analyze each of the evaluation methods (for models 1-6) for their computational complexity.

- In model 1, all t transactions are completely specified, and the data distribution matrix is also known. To determine if a transaction is successful, we need to the scan the distribution matrix. Similarly, determining if a transaction in a lower order partition is to be rolled back due to a write-write conflict with a transaction of higher order partition requires comparison of write-sets of the two transactions. Determining if a transaction needs to be rolled back due to the rollback of a dependency parent also requires a search. All this requires $O(ndt + p^2t^2 + pt^2N)$, where t is the number of transaction types and N is the maximum number of transactions executed in a partition prior to the merge.
- Models 2-6 have a similar computation structure. The number of transaction types (t) is high for model 2 and low for model 6. Each of these models require $O(p^2t^2c + pt^2N)$ time. As before, t is the number of transaction types and N is the maximum number of transactions executed in a partition prior to the merge.

Thus, model 1 is the most complex (computationally) and model 6 is the least complex.

5.2 Space Complexity

We now discuss the space complexity of the six evaluation methods:

- Model 1 requires $O(dn)$ to store the data distribution matrix, $O(n)$ to store the partition information, $O(dt)$ to store the data access information, and $O(nt)$ to store the transaction arrival information. It also requires $O(t^2)$ to store the commutativity information. Thus, it requires $O(dn + dt + nt + t^2)$ space to store model information.
- Models 4-6 require similar information: $O(t)$ to store the average size of read- and write- sets of transaction types, $O(nt)$ for transaction arrival, $O(n)$ for partition information, and $O(t)$ for commute information. Thus they require $O(nt)$ space.
- Model 3, in addition to the space required by models 4-6, also requires $O(t^2)$ for commutativity matrix. Thus it requires $O(nt + t^2)$ space.
- Model 2, in addition to the space required by model 3, also requires t^2 space to store the data overalp information. Thus, it requires $O(nt + t^2)$ storage.

Thus, model 1 has the largest storage requirement and model 6 has the least.

5.3 Evaluation of the Averages

In order to compare the effect of each of these models on the evaluation of the average rollbacks, we have run a number of experiments. In addition to the analytical evaluations for models 1-6, we have also run simulations with Model 1. The results from these runs are summarized in Tables 1-7. Basically these tables describe the number of transactions successfully executed before partition merge (*Before Merge*), number of rollbacks due to class 1 (R_1), rollbacks due to class 2 (R_2), and transactions considered to be successful at the completion of merge (*After Merge*). Obviously, the last term is computed from the earlier three terms. In all these tables, the total number of transaction arrivals into the system during partitioning is taken to be 65000. Also, each node is assumed to receive equal share of the incoming transactions.

- Table 1 summarizes the effect of number of partitions as measured with Models 1-6. Here, it is assumed that each of the data items in the system has exactly $c = 3$ copies. The other assumptions in models 1-6 are as follows:
  1. Model 1 considers 130 transaction types in the system. Each is described by its read- and write-sets and whether it commutes with the other transactions. 90 of the 130 are read-only transactions. The rest of the 40 are read-write. Among the read-write, 15 commute with each other, another 10 commute with each other, and the rest of the 15 do not commute at all. The simulation run takes the same inputs but evaluates the averages by simulation.
  2. Model 2 maps the 130 transaction types into 4 classes. To make the comparisons simple, the above four classes (90+15+10+15) are taken as four types. The data overlap is computed from the information provided in model 1.
  3. Model 3, to facilitate comparison of results, considers the above 4 classes. This model, however, does not capture the data overlap information.
  4. Model 4 considers three types: read-only, read-write that commute among themselves with some probability, and read-write that do not commute at all.
  5. Model 5 considers read-only transactions with read-set size of 3 and read-write transactions with read-set size of 6. Read-write transactions commute with a given probability.
  6. Model 6 only considers the average read-set size (computed as 4 in our case), the portion of read-write transactions (=45/130), and the average write-set size for a read-write (= 2). Probability that any two transactions commute is taken to be 0.4.

From Table 1 it may be observed that:

- The analytical results from analysis of Model 1 is a close approximation of the ones from simulation.
- The evaluation of number of successful transactions prior to the merge is well approximated by all the models. Model 6 deviated the most.
- The difference in estimations of R_1 and R_2 is significant across the models. Model 1 is closest to the

simulation. Model 6 has the worst accuracy. Model 5, surprisingly, is somewhat better than Models 2,3,4, and 6.

- The estimation of R_2 from models 2-6 is about 50 times of the estimation from Model 1. The estimations from Model 1 and the simulation are quite close. From here, we can see that, Models 2-6 yield overly conservative estimates of the number of rollbacks at the time of partition merge. While Model 1 estimated the rollbacks as 1200, Model 2-6 have approximated them as about 13000.
- This difference in estimations seems to exist even when the number of partitions is increased.

- Table 2 summarizes the effect of number of copies on the evaluation accuracies of the models. It may be observed that
  - The difference between evaluations from Model 1 and the others is significant at low ($c = 3$) as well as high ($c = 8$) values of c. Clearly, the difference is more significant at high degrees of replication.
  - The case $p_1 = 4, p_2 = 6, c = 8$ corresponds to a case where each of the 500 data items is available in both the partitions. This is also evident from the fact that all the 65000 input transactions are successful prior to the merge.
  - The results from the analysis and simulation of Model 1 are close to those from simulation.
- Table 3 shows the effect of increasing the number of nodes from 10 (in Table 1) to 20. For large values of n, all the six models result in good approximations of successful transactions prior to merge. The differences in estimations of R_1 and R_2 still persist.
- Table 4 compares models 5 and 6. While model 6 only retains average read-set size information for any transaction, model 6 keeps this information for read-only and read-write transactions separately. This additional information enabled model 5 to arrive at better approximations for R_1 and R_2. In addition, the effect of commutativity on R_1 and R_2 is not evident until $m \geq 0.99$. This is counterintuitive. The simplistic nature of the models is the real cause of this observation. Thus, even though these models have resulted in conservative estimates of R_1 and R_2, we can't draw any positive conclusions about the effect of commutativity on the system throughput.
- The comments that were made about the conservative nature of the estimates from models 5 and 6 also applies to model 2. These results are summarized in Table 5. Even though this model has much more system information than models 5 and 6, the results (R_1 and R_2) are not very different. However, the effect of commutativity can now be seen at $m \geq 0.95$.
- Having observed that the effect of commutativity is almost lost for smaller values of m in models 2-6, we will now look at its effect with model 1. These results are summarized in Table 6. Even at small values of m, the effect of commutativity on the throughput is evident. In addition, it increases with m. This observation holds at both small and large values of c.
- In Table 7, we summarize the effect of variations in number of copies. In Tables 1-6, we assumed that each data item has exactly the same number of copies. This is more relevant to Model 1. Thus we only consider this model in determining the effect of copy variations on evaluation of R_1 and R_2. As shown in this table, the effect is significant. As the variation in number of copies is increased, the number of successful transactions prior to merge decreases. Hence, the number of conflicts are also reduced. This results in a reduction of R_1 and R_2. AS long as the variations are not very significant, the differences are also not significant.

6. CONCLUSIONS

In this paper, we have introduced the problem of estimating the number of rollbacks in a partitioned distributed database system. We have also introduced the concept of transaction commutativity and described its effect on transaction rollbacks. For this purpose, the data distribution, replication, and transaction characterization aspects of distributed database systems have been modeled with three parameters. We have investigated the effect of six distinct models on the evaluation of the chosen metric. These investigations have resulted in some very interesting observations. This study involved developing analytical equations for the averages, and evaluating them for a range of parameters. We also used simulation for one of these models. Due to lack of space, we could not present all the obtained results in this paper. In this section, we will summarize our conclusions from these investigations.

We now summarize these conclusions.

- Random data models that assume only average information about the system result in very conservative estimates of system throughput. One has to be very cautious in interpreting these results.
- Adding more system information does not necessarily lead to better approximations. In this paper, the system information is increased from model 6 to model 2. Even though this increases the computational complexity, it does not result in any significant improvement in the estimation of number of rollbacks.
- Model 1 represents a specific system. Here, we define the transactions completely. Thus it is closer to a real-life situation. Results (analytical or simulation) obtained from this model represent actual behavior of the specified system. However, results obtained from such a model are too specific, and can't be extended for other systems.
- Transaction commutativity appears to significantly reduce transaction rollbacks in a partitioned distributed database system. This fact is only evident from the analysis of model 1. On the other hand, when we look at models 2-6, it is possible to conclude that commutativity is not helpful unless it is very very high. Thus, conclusions from model 1 and models 2-6 appear to be contradictory. Since models 3-6 assume average transactions that can randomly select any data item to read (or write), the evaluations from these models are likely to predict higher conflicts and hence more rollbacks. The benefits due to commutativity seem to disappear in the average behavior. Model 1, on the other hand, describes a specific system, and hence can accurately compute the rollbacks. It is also able to predict the benefits due to commutativity more accurately.
- The distribution of number of copies seems to affect the evaluations significantly. Thus, accurate modeling of this distribution is vital to evaluation of rollbacks.

In addition to developing several system models and evaluation techniques for these models, this paper has one significant contribution to the modeling, simulation, and performance analysis community.

> If an abstract system model with average information is employed to evaluate the effectiveness of a new technique or a new concept, then we should only expect conservative estimates of the effects. In other words, if the results from the average models are positive, then accept the results. If these are negative, then repeat the analysis with a less abstracted model. Concepts/techniques that are not appropriate for an average system may still be applicable for some specific systems.

Table 1. Effect of Number of Partitions on Rollbacks

Model #	$p_1 = 4, p_2 = 6, c = 3$ Before Merge	R_1	R_2	After Merge	$p_1 = 4, p_2 = p_3 = 3, c = 3$ Before Merge	R_1	R_2	After Merge
Sim.	50200	1000	205	48995	31450	0	0	31450
1	50200	1000	199	49001	31450	0	0	31450
2	48315	3597	10322	34397	27069	3460	8945	14664
3	48315	3464	10194	34657	27069	2798	9410	14861
4	48618	3667	10243	34708	27657	3255	9444	14958
5	47276	2679	10238	34360	24207	1507	9106	13594
6	46593	3852	8570	34171	22356	2937	6673	12747

Table 2. Effect of Number of Copies on Rollbacks

Model #	$p_1 = 4, p_2 = 6, c = 2$ Before Merge	R_1	R_2	After Merge	$p_1 = 4, p_2 = 6, c = 8$ Before Merge	R_1	R_2	After Merge
Sim.	34600	200	15	34385	65000	4000	4970	56030
1	34600	200	0	34400	65000	4000	4981	56019
2	31069	1998	5119	23952	65000	8000	17777	39223
3	31069	1601	5331	24131	65000	8000	17786	39214
4	31595	1798	5420	24377	65000	8000	17786	39214
5	23203	1568	2326	19309	65000	8000	17875	39125
6	27138	3413	1701	22024	65000	8000	17860	39140

Table 3. Effect of Number of Nodes on Rollbacks

Model #	$p_1 = 10, p_2 = 10, c = 5$ Before Merge	R_1	R_2	After Merge	$p_1 = 10, p_2 = 10, c = 12$ Before Merge	R_1	R_2	After Merge
Sim.	61250	4000	6240	51010	65000	5000	6231	53769
1	61250	4000	6231	51019	65000	5000	6231	53769
2	61024	9090	21183	30751	65000	10000	22277	32723
3	61024	8992	21286	30746	65000	10000	22286	32714
4	61100	9031	21326	30743	65000	10000	22286	32714
5	60968	9064	21292	30613	65000	10000	22375	32625
6	60876	9363	20936	30577	65000	10000	22360	32640

ACKNOWLEDGEMENT

This research was sponsored in part by the NASA Langley Research Center under contract NAG-1-1154.

REFERENCES

Coffman, E. G., E. Gelenbe, and B. Plateau (1981), "Optimization of Number of Copies in a Distributed Database," *IEEE Transactions on Software Engineering* 7, 1, 78-84.

Davidson, S.B. (1982), "An optimistic protocol for partitioned distributed database systems," Ph.D. thesis, Department of EECS, Princeton University.

Davidson, S.B. (1984), "Optimism and consistency in partitioned distributed database systems," *ACM Transactions on data' systems* 9, 3, 456-481.

Davidson, S.B., H. Garcia-Molina, and D. Skeen (1985), "Consistency in partitioned networks," *ACM Computing Surveys* 17, 3, 341-370.

Davidson, S.B. (1986), "Analyzing partition failure protocols," Technical Report MS-CIS-86-05, Department of Computer and Info. Sci., Univ. of Pennsylvania.

Garcia-Molina, H. (1983), "Using semantic knowledge for transaction processing in a distributed system," *ACM Trans. on Database Systems* 8, 2, 186-213.

Jajodia, S. and P. Speckman (1985), "Reduction of conflicts in partitioned databases," In *Proceedings of the 19th Annual Conference on Information Sciences and Systems*, 349-355.

Jajodia, S. and R. Mukkamala (1990), *Measuring the Effect of Commutative Transactions On Distributed Database Performance*, To appear in *Computer Journal*.

Mukkamala, R. (1987), "Design of Partially Replicated Distributed Database Systems," Technical Report 87-04, Department of Computer Science, University of Iowa.

Mukkamala, R. (1990), "Measuring the Effects of distributed database models on transaction rollback measures," Technical Report 90-38, Department of Computer Science, Old Dominion University.

Wright, D. D. (1983a), "Managing distributed databases in partitioned networks," Ph.D. thesis, Department of Computer Science, Cornell University, (also TR 83-572).

Wright, D. D. (1983b), "On merging partitioned databases," *ACM SIGMOD Record* 13, 4, 6-14.

Table 4. Effect of m on Rollbacks (Models 5 and 6: $p_1 = 4, p_2 = 6, c = 3$

	Model 5				Model 6			
m	Before Merge	R_1	R_2	After Merge	Before Merge	R_1	R_2	After Merge
0.00	47276	2679	10238	34360	46593	3852	8570	34171
0.50	47276	2679	10238	34360	46593	3852	8570	34171
0.80	47276	2679	10238	34360	46593	3852	8570	34171
0.90	47276	2679	10238	34360	46593	3848	8574	34171
0.95	47276	2678	10239	34360	46593	3774	8774	34175
0.99	47276	2208	10665	34403	46593	2182	10109	34301
1.00	46726	0	0	46726	46593	0	0	46593

Table 5. Effect of m on Rollbacks (Model 2: $p_1 = 4, p_2 = 6$)

	$c = 3$				$c = 8$			
m	Before Merge	R_1	R_2	After Merge	Before Merge	R_1	R_2	After Merge
0.0	48315	3597	10322	34397	65000	8000	17973	39027
0.27	48315	3597	10322	34397	65000	8000	17973	39027
0.40	48315	3597	10322	34397	65000	8000	17973	39027
0.77	48315	3597	10322	34397	65000	8000	17973	39027
0.95	48315	3205	10708	34402	65000	7660	18312	39028
0.99	48315	986	12882	34447	65000	4321	21642	39037
1.0	48315	0	0	48315	65000	0	0	65000

Table 6. Effect of m on Rollbacks (Model 1: $p_1 = 4, p_2 = 6$)

	$c = 3$				$c = 8$			
m	Before Merge	R_1	R_2	After Merge	Before Merge	R_1	R_2	After Merge
0.0	50200	4000	1199	45001	65000	8000	6379	50621
0.27	50200	1000	199	49001	65000	4000	4981	56019
0.40	50200	800	199	49201	65000	1800	2793	60407
0.77	50200	0	0	50200	65000	0	0	65000
1.0	50200	0	0	50200	65000	0	0	65000

Table 7. Effect of Variations in # of Copies on Rollbacks
(Model 1: $p_1 = 4, p_2 = 6, w/c : m = 0.27, wo/c : m = 0.0$)

$p_1 = 4, p_2 = 6, c = 3$					
Copy Distribution		Before Merge	R_1	R_2	After Merge
$d_3 = 500$	w/c	50200	1000	199	49001
	wo/c	50200	4000	1199	45001
$d_2 = d_4 = 100, d_3 = 300$	w/c	48300	1000	997	46303
	wo/c	48300	4200	1793	42307
$d_2 = d_3 = 167, d_4 = 166$	w/c	41400	200	0	41200
	wo/c	41400	2000	597	38803
$d_1 = d_2 = d_3 = d_4 = d_5 = 100$	w/c	40400	200	0	40200
	wo/c	40400	1600	797	38003
$d_1 = d_5 = 250$	w/c	28700	0	0	28700
	wo/c	28700	1200	199	27301

Measuring the Effects of Distributed Database Models On Transaction Availability Measures

Ravi Mukkamala

Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529.
email: mukka@cs.odu.edu

Abstract

Data distribution, data replication, and system reliability are key factors in determining the availability measures for transactions in distributed database systems. In order to simplify the evaluation of these measures, database designers and researchers tend to make unrealistic assumptions about these factors. In this paper, we investigate the effect of such assumptions on the computational complexity and accuracy of such evaluations. We represent a database system with five parameters related to the above factors. Probabilistic analysis is employed to evaluate the availability of read-only and read-write transactions. We consider both the read-one/write-all and the majority-read/majority-write replication control policies. We conclude that transaction availability is more sensitive to variations in degrees of replication, less sensitive to data distribution, and insensitive to reliability variations in a heterogeneous system. The computational complexity of the evaluations is found to be mainly determined by the chosen distributed database model, while the accuracy of the results are not so much dependent on the models.

Keywords and phrases: Availability, Database models, Distributed Systems, Distributed Database Systems, Performance Evaluation, Probabilistic Analysis, Reliability, Transaction Availability

Measuring the Effects of Distributed Database Models
On Transaction Availability Measures

1 Introduction

A distributed database system is a collection of cooperating nodes each containing a set of data objects [1]. A user transaction can enter such a system at any of these nodes. The receiving node, some times referred to as the *coordinating* or *initiating* node, undertakes the task of locating the nodes that contain the data objects required by a transaction.

When we consider systems that require high guarantees for successful execution of transactions (especially for read-only transactions), it is important to consider transaction availability. Even though there are a number of availability (and reliability) metrics defined for computer systems [2,9], in this paper we choose two metrics: Start availability (TSA) and finish availability (TFA).

Transaction start availability (TSA) defines the probability with which a transaction can successfully start its execution. By our definition, a transaction is said to have a successful start when it can access all the required[2] copies of the data objects that it needs for its execution. For simplicity, we consider a data copy at a node to be *available* for access when that node is up and it is accessible from the node that is currently coordinating the execution of the transaction. A transaction can start its execution as soon as all the required data object copies are available.

Transaction finish availability (TFA) defines the probability with which a transaction can complete its execution, given that it has started its execution successfully. If execution times for transactions are negligible (as compared to the mean-time-to-fail of the components), then this reliability will be close to 1. However, since transactions take a finite but significant amount of time to execute, it is quite possible that the nodes that are involved in the execution of a transaction (and available at the start of execution) may

[1] In this paper, the basic unit of access in a database is referred to as a data object.

[2] The number of copies of an object that are required to be accessed by a transaction depends on the operation (read or write) and the replica copy control (e.g. read-one/write-all, majority) [3,18].

fail during its execution. In this case, the transaction is said to be aborted. In such cases, the execution needs to be restarted.

Formal definitions and evaluation of these two metrics (TSA and TFA) depend on several factors such as the fault model of the system (including the reliabilities of the system components), the transaction execution policy, the data distribution policy, the degree of data replication, the concurrency and commit protocols, and the characteristics of the given transaction [4,7,9]. In addition, TFA depends on the execution times of transactions.

Even though it is theoretically possible to formulate equations expressing the two metrics in terms of the above mentioned factors, the evaluation of these equations is extremely cumbersome and requires unreasonably high computation times. The evaluation of the exact values for these measures generally involves both analysis and simulation. Evaluation tools with such large execution times are certainly not acceptable to a database designer who needs to evaluate a number of such possible database configurations before arriving at a final design.

To overcome these problems, designers and researchers generally resort to approximation techniques [7,8,16]. These techniques reduce the computation time by making simplifying assumptions regarding data distribution, data replication, and transaction execution. The time complexity of these techniques primarily depends on the underlying model as well as the evaluation technique.

The effect of data distribution and replication models on evaluation of transaction response time has been measured in earlier studies [13]. These studies indicate that the *computational complexity* of a selected database model does not necessarily reflect the accuracy of the resulting performance evaluations. In fact, a model requiring computational time of [3] $O(n^2)$ has yielded results very close to those from a complex model with $O(n^n)$ complexity.

In this paper, we study the effect of data distribution, data replication, and fault models on the accuracy of transaction availability evaluations. We employ probabilistic analysis to arrive at the estimates for the desired values for six typical models.

The balance of this paper is outlined as follows. Section 2 formally

[3] Here, n denotes the number of nodes in a distributed system.

defines the problem under consideration. In Section 3, we describe a classification scheme for data distribution and replication policies. Section 4 illustrates the advantages of probabilistic analysis over simulation, and employs this technique to evaluate the measures for two different models. In Section 5, we compare the analysis methods for six models based on computational complexity, space complexity, and the accuracy of the measures. Finally, in Section 6, we summarize the obtained results, and suggest a general approach for design and analysis of these systems.

2 Problem Description

In this paper, a read-only transaction is characterized by the average number of data objects that it reads (i.e., its read-set size). Similarly, a read-write transaction is characterized by the number of data objects that it reads (read-set size), and the number of data objects that it updates (write-set size).

The problem of estimating the availability of a read-only transaction may be formulated as:

> Given the following parameters, estimate TSA_s and TFA_s for a read-only transaction that requires s data objects for read access.
>
> - n, the number of nodes in the database[4]
> - N, the index set for the nodes in the database; $N = \{1, 2, \ldots, n\}$
> - d, the number of data objects in the database
> - D, the index set for the data objects in the database; $D = \{1, 2, \ldots, d\}$
> - GD, the global data directory that contains the location of each of the d data objects; the GD matrix contains d rows and n columns, each of which is either a 0 or a 1; i.e., $GD_{ij} = 0$ or 1, $\forall i \in D$ and $\forall j \in N$
> - the reliability of the nodes in the network.

The problem of estimating the metrics for a read-write transaction can be similarly defined.

[4]Table 1 summarizes the notation used in this paper

Symbol	Description
a_i	The number of data objects accessed from the i^{th} group
c	The average number of copies of a data object
c_l	The number of copies of a data object in the l^{th} class
d	The number of data objects in the database
d_i	The number of data objects in the i^{th} class
g	The number of data object groups
k	Number of live nodes
n	Number of nodes
n_i	The number of nodes in the i^{th} class
p	The number of copy classes
q	The number of reliability classes
r	The average node reliability
r_i	The reliability of a node in the i^{th} class
s	The size of the read-set
A_1, A_2	Policies representing the data grouping
B_1, B_2	Policies representing limits on the data objects per node
C_1, C_2	Policies representing the degree of replication
D_1, D_2	Policies representing the copy distribution
E_1, E_2	Policies representing the component reliability
D	The index set for the data objects in the database
GA	Group access vector representing the number of objects accessed from each class or group
GD	Global data directory (or dictionary)
N	The index set for the nodes in the database
TSA_s	Transaction start availability of a read-only transaction with read-set size s (read-one/write-all policy)
$TSA'_{x,y}$	Transaction start availability of a read-write transaction with read-set size $x + y$ and write-set size y (read-one/write-all policy)
TSA''_s	Transaction start availability of a transaction with read-set size s (read-majority/write-majority policy)
x	The size of the read-only object set
y	The size of the read-write object set

Table 1: Notation

3 Model Description

As stated in the introduction, the primary objective of this paper is to investigate the effect of data distribution, replication, and fault models on availability estimations and the computational complexity of these evaluations.

To describe a data distribution, replication, and fault model, we characterize it with five orthogonal parameters:

A - Object grouping (or clustering)

B - Limits on the number of data objects per node

C - Degree of object replication (or the number of copies)

D - Constraints on distribution of object copies

E - Constraints on component reliability

We now discuss each of these parameters in detail.

Some distributed database systems allocate individual data objects [5, 10]. We categorize this strategy as A_1. In other systems, data objects are first partitioned into disjoint groups, and then the resulting groups are allocated [12,16,17]. Thus, the copies of all the data objects in a given group are allocated to the same set of nodes. We refer to this strategy as A_2.

Some database designers place no explicit limit on the number of data objects that may be placed at a node [7]. This strategy is named as B_1. Others restrict the number of data objects that may be placed at a given node. This may be attributed to storage limitations or for security reasons [11]. We refer to this strategy as B_2.

For simplicity, several analysis techniques assume that each data object has the same number of copies (or degree of replication) in the database system [6,16]. Some other techniques characterize the degree of replication of a database by the average degree of replication of data objects in that database [7]. In this paper, both these categories are referred to as C_1. Others treat the degree of replication of each data object independently. We refer to this as strategy C_2.

Some database designers and analysts assume that each data object (or group) copy is randomly distributed among the nodes in the distributed system [7]. We refer to this as D_1. Others assume some specific allocation schemes for data object (or group) copies [11]. Assuming complete knowledge of data copy distribution (GD) is one such assumption. Depending on the type of allocation, such assumptions may simplify the performance analysis [13]. This category is referred to as D_2.

Again for simplicity, some database designers and analysts assume that all components (nodes and links) in a distributed system have the same reliability factor [1]. In this paper, we only consider node failures and node repairs[5]. We let E_1 denote a policy where all nodes are assumed to have the same reliability characteristics, and E_2 denote a policy where nodes are classified based on their reliability characteristics.

Using this classification, any known data distribution, replication, and reliability policies may be categorized by these five parameters. For example, $< A_2, B_1, C_1, D_2, E_1 >$ represents a policy where

1. Data objects are *first grouped* and then allocated.
2. There is *no explicit limit* placed on the number of data objects (or groups) allocated to any node.
3. Each group has the same average degree of replication.
4. The copies of a group are distributed in some systematic manner among the nodes in the system.
5. All nodes in the system have identical reliability characteristics.

With these five parameters, we can describe thirty two basic policies. Several variations of these basic schemes are possible due to variations in systematic distributions (D_2), variations on the limits of data objects per node (B_2), and the types of grouping (A_2). Due to space limitations, in this paper we chose to present the results for six of these policies. Interested reader may refer to [14] for an analysis of other policies.

[5]That is, the underlying network structure almost always facilitates communication among live nodes.

We chose the following six policies to study the effect of the above mentioned parameters on availability computations:

Model 1: $< A_1, B_1, C_1, D_1, E_1 >$

Model 2: $< A_2, B_1, C_1, D_1, E_1 >$

Model 3: $< A_1, B_2, C_1, D_1, E_1 >$

Model 4: $< A_1, B_1, C_2, D_1, E_1 >$

Model 5: $< A_1, B_1, C_1, D_2, E_1 >$

Model 6: $< A_1, B_1, C_1, D_1, E_2 >$

Among these, Model 1 represents a simple system that is computationally attractive (as shown in Table 2). Model 2 reflects the effect of data grouping on the evaluation. Similarly, Model 3 reflects the effect of placing limits on number of data objects. Model 4 represents the effect of variations in number of copies of data objects on availability evaluation. Model 5 shows the effect of biased or *non-random* distributions of data objects on the evaluation. Finally, Model 6 reflects the effect of non-homogeneous environment (i.e., different node reliability characteristics) on transaction availability evaluation.

In the following section, we derive closed-form expressions for the average transaction availabilities for Models 1 and 2.

4 Probabilistic Computation of the Availabilities

There are several approaches for computing the availability of a given transaction in a database. These computations assume a given data distribution, data replication, and fault models. We now look at two such methods: simulation and probabilistic analysis.

Using simulation, one can generate the data distribution matrix (GD) based on the data distribution and replication model. One can also generate the reliabilities for each of the nodes in the system [6]. Similarly, one can generate all possible transactions (with different read-sets and write-sets) that

[6] Here, we ignore the possibility of network partitioning, and thereby ignore link reliability factor.

can be received at each of the nodes in the network. For each such transaction received by the system, the data distribution matrix can be searched, and its ability to access all the required data objects may be verified. In addition to generating transactions, we should also generate node failures and node repairs in the time domain. Thus, some transactions may not be successful due to the inaccessibility of one or more data objects that they require (due to node failures). With such statistics (of successful/unsuccessful transactions) in hand, we can obtain the average availability of a transaction of a given size. This average corresponds to a single distribution matrix. The generation and evaluation process may have to be repeated sufficient times to get the required confidence in the final result. Since there are d data objects, there are $\binom{d}{s}$ possible transactions with read-set [7] size s, and there are n nodes where each of these may be received. Given a *transaction*, and the *node* where it is received, determining the state (successful/unsuccessful) of a transaction takes at least $O(nd)$ computations (i.e., to scan the columns of the GD matrix corresponding to available nodes). If the distribution matrix is generated k times, then the evaluation of the desired average set size for a transaction of size s takes $O(kn^2d\binom{d}{s})$ time. In general, k is a function of the number of copies, the number of data objects, the number of nodes, and the data distribution model, and it could be very high. Suppose $d = 100$, $s = 10$, and $n = 10$, then this method requires approximately $10^{17}k$ computations. Even for reasonable values of k, this is an unreasonably high computation time.

To avoid this large evaluation time, we adopt probabilistic analysis. In this analysis, we essentially study the given data distribution and reliability model and arrive at an expression for the average transaction availability for a given read-set (or write-set) size. With probabilistic analysis, some data distribution models (e.g., Models 1 and 3) may require insignificant amounts of computation. Some may need moderate computation times (e.g., Models 2 and 6), whereas others may need large computation times (e.g., Models 4 and 5). Regardless of the model, all these need considerably less computation time (with more accuracy of results) than the corresponding simulation methods.

We now illustrate the probabilistic method of analysis by applying it for

[7] The corresponding term for write-sets of update transactions may be easily written.

Models 1 and 2. Expressions for other models may be derived in a similar manner. Interested reader may find the details of these derivations in [14].

4.1 Derivation of Reliability Metrics for Model 1

Model 1, designated as $< A_1, B_1, C_1, D_1, E_1 >$ assumes the following about the data distribution and replication:

[R1] The data objects are allocated individually (i.e. not grouped) to the nodes.

[R2] There are no limits placed on the number of data objects that may be placed at each node.

[R3] The average degree of replication (c) of a data object is given.

[R4] The copies of a data object are allocated randomly.

[R5] Each node in the system has identical reliability ($= r$).

Further, to simplify the illustration of the current analysis, we make the following assumptions regarding the distribution of groups, and the participating node set determination:

[R6] Each transaction is equally likely to access any data object.

[R7] The transactions that enter the distributed system are coordinated by a set of *reliable servers* that search the distributed database system (i.e., the availability of nodes and their dictionaries) for the availability of the required data objects.

Due to Rule R7, we will not distinguish transactions that are received at different locations in the system. Thus, we will disregard the originating node as a parameter in this analysis[8].

[8]The analysis can easily be extended to a situation where transactions received at an unavailable node are automatically considered as unsuccessful.

4.1.1 Derivation of Availability for Read-only Transactions

Let us consider a read-only transaction T_1 with s objects in its read-set and received at one of the servers. Let us also assume that the copy control algorithm follows a read-one/write-all policy. Thus T_1 needs to access *any one* of the c copies of a data object that it requires.

Given that exactly k of the n nodes are available (i.e., up), the probability that at least one copy of a given data object is available is given by:

$$P_{k,1} = 1 - \frac{\binom{n-k}{c}}{\binom{n}{c}} \tag{1}$$

By definition of the read-one/write-all policy, $P_{k,1}$ represents the probability that a data object is available for read access in the system. Since each data object is allocated independently to the nodes in the system (by Rules R1 and R2), the probability that all s data objects required by T_1 are available for read access within these k nodes can then be expressed as:

$$P_{k,s} = P_{k,1}^s = \left[1 - \frac{\binom{n-k}{c}}{\binom{n}{c}}\right]^s \tag{2}$$

Assuming the reliability of any given node to be r (from Rule R5), the probability that T_1 has successfully started is:

$$\begin{aligned} TSA_s &= \sum_{k=1}^{n} \binom{n}{k} r^k (1-r)^{n-k} P_{k,s} \\ &= \sum_{k=1}^{n} \binom{n}{k} r^k (1-r)^{n-k} \left[1 - \frac{\binom{n-k}{c}}{\binom{n}{c}}\right]^s \end{aligned} \tag{3}$$

Given that T_1 has successfully started, we will now compute the probability with which it can be successfully completed. Let us assume that n_s nodes are involved in the execution of T_1, and that it has an execution time of t units. Now, in order for T_1 to be successful, all these n_s nodes have to be available for at least t units of time, given that they were available at the start of execution. Assuming an exponential distribution for time between node failures with a failure rate of λ, the probability that a node which is available at time zero is available throughout time t is given by:

$$A_t = e^{-t\lambda} \tag{4}$$

From here, the probability that none of the n_s nodes have failed during time t is given by:

$$\begin{aligned} TFA_s &= A_t^{n_s} \\ &= e^{-n_s t\lambda} \end{aligned} \tag{5}$$

Estimating n_s for transaction T_1 is a complex problem. This problem has been well investigated and the details of the solutions may be found in [15]. In this paper, we assume that n_s for T_1 has been obtained *a priori* for a given data distribution and fault model.

4.1.2 Derivation of Availability for Read-write Transactions

Let us now consider a read-write transaction T_2 with s objects in its read-set and y objects in its write-set. Let us assume that for a given read-write transaction write-set$\subseteq$read-set [3,7]. Thus, among the s data objects, y objects are both read and written, while $x = s - y$ data objects are only read. (Note that the intersection of the read-only and the read-write sets of the data objects is empty.) Since the replication control algorithm follows a read-one/write-all policy, T_2 needs to access all c copies of the y data objects and any one copy of the x data objects.

Given that exactly k of the n nodes are available (i.e., up), the probability that *all* c copies of a given data object are available is given by:

$$P'_{k,1} = \frac{\binom{k}{c}}{\binom{n}{c}} \tag{6}$$

Since each data object is allocated independently to the nodes in the system (by Rules R1 and R2), the probability that all y data objects required by T_2 are accessible for update is expressed as:

$$P'_{k,y} = \left[\frac{\binom{k}{c}}{\binom{n}{c}}\right]^y \tag{7}$$

Similarly, the probability that all x data objects are available for read access may be computed as:

$$P_{k,x} = \left[1 - \frac{\binom{n-k}{c}}{\binom{n}{c}}\right]^x \tag{8}$$

From here, the probability that T_2 is successfully started may be computed as:

$$\begin{aligned} TSA'_{x,y} &= \sum_{k=1}^{n} \binom{n}{k} r^k (1-r)^{n-k} P'^{y}_{k,y} P^{x}_{k,x} \\ &= \sum_{k=1}^{n} \binom{n}{k} r^k (1-r)^{n-k} \left[\frac{\binom{k}{c}}{\binom{n}{c}}\right]^y \left[1 - \frac{\binom{n-k}{c}}{\binom{n}{c}}\right]^x \end{aligned} \tag{9}$$

The finish availabilities for T_2 may be similarly computed using Equations (4) and (5) where n_s is now replaced by $n_{x,y}$ [14].

4.1.3 Derivation of Availability for Transactions with Majority Consensus

In the above two sections, we dealt with read-one/write-all replication control policy. The majority consensus protocols [18] which require the accessibility of at least a majority of the total copies of a data object for both read and write operations are very attractive in a failure prone environment. Since both read and write operations require the same number of copies of a data object, in this analysis we do not distinguish between read-only and update transactions. Here, we simply refer to T_1 as a transaction.

Let $m = \lceil \frac{c+1}{2} \rceil$ represent the majority of copies. Then the expression for start availability for T_1 is given as:

$$TSA''_s = \sum_{k=m}^{n} \binom{n}{k} r^k (1-r)^{n-k} \left[\sum_{l=m}^{c} \frac{\binom{k}{l}\binom{n-k}{c-l}}{\binom{n}{c}}\right]^s \tag{10}$$

Similarly, the expression for the finish availability for T_1 may be expressed as:

$$\begin{aligned} TFA_s &= A_t^{n_s} \\ &= e^{-n_s t\lambda} \end{aligned} \tag{11}$$

where n_s now represents the average number of nodes accessed for executing T_1 with the majority consensus protocol [15].

4.2 Derivation of Transaction Availability for Model 2

Model 2, designated as $< A_2, B_1, C_1, D_1, E_1 >$ is similar to Model 1, except that the data objects are now grouped, and the groups are then allocated to nodes in the system. This may be described as:

[R9] The data objects are first grouped and the groups are then allocated, to the nodes. Let the d data objects be partitioned into t distinct groups. Let d_k represent the number of data objects in group k. Thus, $\sum_{i=1}^{t} d_i = d$.

[R10] There are no limits placed on the number of groups that may be placed at each node.

[R11] The degree of replication is the same for each group (c).

[R12] The copies of a group are allocated randomly.

[R13] Each node in the system has identical reliability (r).

Again, to simplify analysis, we make the following assumptions:

[R14] Each transaction is equally likely to access any data object.

[R15] The transactions that enter the distributed system are coordinated by a set of *reliable servers* that search the distributed database system (i.e., the availability of nodes and their dictionaries) for the availability of required data objects.

4.2.1 Derivation of Availability for Read-only Transactions

Once again let us consider transaction T_1 executing under a read-one/write-all policy. Given that k of the n nodes are available (i.e., up), the probability that at least one copy of *group* k is available is given by:

$$1 - \frac{\binom{n-k}{c}}{\binom{n}{c}} \tag{12}$$

If the vector $GA =< a_1, a_2, \ldots, a_t >$ represents the number of data objects accessed by T_1 from each of the t groups, then the probability that T_1 is

successfully started may be computed as:

$$TSA_s = \sum_{GA} Pr(GA) \sum_{l=1}^{n} \binom{n}{l} r^l (1-r)^{n-l} \prod_{k=1}^{t} \left[1 - \frac{\binom{n-l}{c}}{\binom{n}{c}} \right]^{f(k)} \quad (13)$$

$$Pr(GA) = \frac{\binom{d_1}{a_1}\binom{d_2}{a_2}\cdots\binom{d_t}{a_t}}{\binom{d}{s}} \quad (14)$$

$$f(k) = \begin{cases} 1 & \text{if } a_k > 0 \\ 0 & \text{otherwise} \end{cases} \quad (15)$$

$$GA = < a_1, a_2, \ldots, a_t >, \quad \sum_{k=1}^{t} a_k = s \text{ and } \forall k\ 1 \le k \le t\ 0 \le a_k \le d_k \quad (16)$$

When data objects are equally distributed among the groups (i.e., $d_1 = d_2 = \ldots = d_t = \frac{d}{t}$), then this expression may be further simplified as:

$$TSA_s = \sum_{l=1}^{n} \sum_{k=1}^{t} \binom{n}{l} r^l (1-r)^{n-l} \binom{t}{k} \left[1 - \frac{\binom{n-l}{c}}{\binom{n}{c}} \right]^k \left[\frac{\binom{n-l}{c}}{\binom{n}{c}} \right]^{t-k} \frac{\binom{\frac{dk}{t}}{s}}{\binom{d}{s}} \quad (17)$$

The expression for TFA_s is the same as in Equation (5).

4.2.2 Derivation of Availability for Read-write Update Transactions

Let us consider transaction T_2 which requires x objects for read-only operations and y data objects for read and write operations ($s = x + y$). Thus we need to define two GA vectors for read-only and read-write data object sets:

$$GA' = < a'_1, a'_2, \ldots, a'_t >, \quad \sum_{k=1}^{t} a'_k = x \text{ and } \forall k\ 1 \le k \le t\ 0 \le a' \le d_k$$

$$GA'' = < a''_1, a''_2, \ldots, a''_t >, \quad \sum_{k=1}^{t} a''_k = y \text{ and } \forall k\ 1 \le k \le t\ 0 \le a'' \le d_k - a'_k$$

In computing $TSA'_{x,y}$ we should recall that if a data object is write accessible under a given node availability conditions, it is also read accessible. However the reverse is not true. These two facts are made use of in deriving the following expression for $TSA'_{x,y}$:

$$
\begin{aligned}
TSA'_{x,y} &= \sum_{GA'} \sum_{GA''} Pr(GA') Pr(GA'') \sum_{l=1}^{n} \binom{n}{l} r^l (1-r)^{n-l} \\
&\quad \prod_{k=1}^{t} \left[1 - \frac{\binom{n-l}{c}}{\binom{n}{c}} \right]^{f'(k)} \prod_{k=1}^{t} \left[\frac{\binom{l}{c}}{\binom{n}{c}} \right]^{f''(k)} \qquad (18) \\
Pr(GA') &= \frac{\binom{d_1}{a'_1}\binom{d_2}{a'_2} \cdots \binom{d_t}{a'_t}}{\binom{d}{x}} \\
Pr(GA'') &= \frac{\binom{d_1-a'_1}{a''_1}\binom{d_2-a'_2}{a''_2} \cdots \binom{d_t-a'_t}{a''_t}}{\binom{d-x}{y}} \\
f'(k) &= \begin{cases} 1 & \text{if } a''_k = 0 \wedge a'_k > 0 \\ 0 & \text{otherwise} \end{cases} \\
f''(k) &= \begin{cases} 1 & \text{if } a''_k > 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

As before, when data objects are equally distributed among the groups (i.e. $d_1 = d_2 = \ldots = d_t = \frac{d}{t}$), this expression may be simplified as:

$$
\begin{aligned}
TSA'_{x,y} &= \sum_{l=1}^{n} \sum_{k_1=1}^{t} \sum_{k_2=0}^{t-k_1} \binom{n}{l} r^l (1-r)^{n-l} \binom{t}{k_1} \binom{t-k_1}{k_2} \left[\frac{\binom{l}{c}}{\binom{n}{c}} \right]^{k_1} \\
&\quad \left[1 - \frac{\binom{n-l}{c} + \binom{l}{c}}{\binom{n}{c}} \right]^{k_2} \left[\frac{\binom{n-l}{c}}{\binom{n}{c}} \right]^{t-k_1-k_2} \frac{\binom{dk_1}{t}}{\binom{d}{y}} \frac{\binom{d(k_1+k_2)}{t} - y}{\binom{d-y}{x}} \qquad (19)
\end{aligned}
$$

The finish availability $TFA_{x,y}$ may be computed using Equation (5) where n_s is now replaced by $n_{x,y}$ which is assumed to be known a priori in this paper.

4.2.3 Derivation of Availability for Transactions with Majority Consensus

As described in Section 4.1.3, under the majority consensus protocol both the read-set and read-write set are treated in the same way for access probability computations. Thus, we only consider a read-only transaction with a read-set size of s. The expression for TSA''_s can now be written as:

$$TSA''_s = \sum_{GA} Pr(GA) \sum_{l=1}^{n} \binom{n}{l} r^l (1-r)^{n-l} \prod_{k=1}^{t} \left[\sum_{l'=m}^{c} \frac{\binom{l}{l'}\binom{n-l}{c-l'}}{\binom{n}{c}} \right]^{f(k)} \quad (20)$$

$$m = \left\lceil \frac{c+1}{2} \right\rceil \quad (21)$$

where $Pr(GA)$ and $f(k)$ are as defined in Equations (14) and (15).

Once again, when data objects are equally distributed among the groups (i.e. $d_1 = d_2 = \ldots = d_t = \frac{d}{t}$), this expression may be written as:

$$TSA''_s = \sum_{l=m}^{n} \sum_{k=1}^{t} \binom{n}{l} r^l (1-r)^{n-l} \binom{t}{k} \left[\sum_{l_1=m}^{\min(l,c)} \frac{\binom{l}{l_1}\binom{n-l}{c-l_1}}{\binom{n}{c}} \right]^{k} \left[1 - \sum_{l_1=m}^{\min(l,c)} \frac{\binom{l}{l_1}\binom{n-l}{c-l_1}}{\binom{n}{c}} \right]^{t-k} \frac{\binom{\frac{dk}{t}}{s}}{\binom{d}{s}} \quad (22)$$

5 Comparison of the Availabilities for the Six Models

As mentioned in the introduction, the main objective of this paper is to determine the effect of data distribution, replication, and fault models on the estimation of transaction availability. To achieve this, we evaluate the desired measure using six different models. The comparison of these evaluations is based on computational time, storage requirement, and the average values obtained.

Due to space limitations, we cannot present the detailed derivations for the average values for Models 3-6. The final expressions, however, are summarized in the appendix.

5.1 Computational Complexity

We now analyze each of the evaluation methods (for Models 1-6) for their computational complexity.

- Let us refer to Model 1. From Equations (3) and (9), it is clear that computation of TSA_s and $TSA_{x,y}$ take $O(cn^2)$ time [9]. Similarly, from Equation (10), it is clear that the computation of TSA''_s requires $O(c^2n^2)$ time.

- We now derive this complexity term for Model 2. Let us first look at the computation of TSA_s. From Equation (14), we derive that the computation of $Pr(GA)$ requires $O(s)$ time. The number of GAs generated is approximately $O(s^t)$ where t represents the number of data object groups. Given a GA vector and $Pr(GA)$, computation of TSA_s requires $O(nct+n^2)$ arithmetic operations (from Equation (18)). Thus the evaluation of TSA_s requires $O(s^t\,(nct + n^2 + s))$ time. Similarly, we can conclude that $TSA'_{x,y}$ requires $O(x^t y^t(nct + n^2 + s))$ time (Equation (19)), and TSA''_s requires $O(s^t(nc^2t + n^2 + s))$ time (Equation (20)).

- For Model 3, the computational complexity for TSA_s is $O(n^2+n(s+c))$ (Equation (23)). Similarly, $TSA'_{x,y}$ and TSA''_s require $O(n^2+n(c+s))$ and $O(n^2 + n(c^2 + s))$ respectively (Equations (24) and (25)).

- The computational complexity for Model 4 depends on the number of copy categories. Assuming that $s < d_k$ for $k = 1, 2, \ldots p$, we can generate approximately s^p different CA vectors. Thus the computation of TSA_s requires $O(s^p(n^2+npc+s))$ time. To compute TSA', we need to compute the number of possible CA' and CA'' vectors. There are approximately x^p CA' vectors and y^p CA'' vectors. Thus, $TSA'_{x,y}$ requires $O(x^p y^p(npc + n^2 + s))$ time. Similarly, we can conclude that TSA''_s requires $O(s^p(npc^2 + n^2 + s))$.

- In Model 5, we assume that the entire data dictionary information is available to us. Given a GD matrix and a node status vector S,

[9] Here, we are assuming that the evaluation of the terms $\binom{p}{q}$ and p^q takes $O(q)$ and $O(1)$ time respectively.

computation of $f(S)$, $f'(S)$, and $f''(S)$ require $O(nd)$ time to search the matrix. Given n, there are 2^n possible S vectors. Thus the computations of TSA, TSA', and TSA'' require $O(2^n(nd+s))$ time.

- In Model 6, the number of NA vectors generated is $(n_1+1)(n_2+1)\ldots(n_q+1)$. For simlification, we approximate it as $\left(\frac{n}{q}+1\right)^q$. Given a NA vector, the computation of TSA, TSA', and TSA'' require $O(s+c+q)$, $O(s+c+q)$ and $O(sc+c^2+qc)$ time respectively. Thus the three metric evaluations require $O((\frac{n}{q}+1)^q(s+c+q))$, $O((\frac{n}{q}+1)^q(s+c+q))$, and $O((\frac{n}{q}+1)^q(cs+c^2+cq))$ time respectively.

These complexities are summarized in Table 2. From this table it may be observed that models 1 and 2 are computationally very attractive. The complexity of evaluations with models 2,4, and 6 depend on the number of groups, the number of copy variations, and the number of reliability variations respectively. For systems with a large number of nodes, evaluations with model 5 are very expensive.

5.2 Space Complexity

We now discuss the space complexity for the six models:

- Models 1 and 3 just require the values of d, c, s, r and n. Thus the storage requirement is $O(1)$
- Since Model 2 requires that the d_i values be stored, and that the GA vectors be generated, it requires $O(t)$ storage, where t is the number of data groups.
- Model 4 requires $O(p)$ storage to contain the p copy classes.
- Model 5 requires $O(nd)$ storage for the GD matrix.
- Model 6 requires $O(q)$ storage to contain the node reliability class information.

Thus, Model 5 has the largest storage requirement. These complexities are summarized in Table 3.

Model	Computational Complexity		
	Read-only	Read-write	Majority
1	$O(cn^2)$	$O(cn^2)$	$O(c^2n^2)$
2	$O(s^t(nct + n^2 + s))$	$O(x^ty^t(nct + n^2 + s))$	$O(s^t(nc^2t + n^2 + s))$
3	$O(n^2 + nc + ns)$	$O(n^2 + nc + ns)$	$O(n^2 + nc^2 + ns)$
4	$O(s^p(npc + n^2 + s))$	$O(x^py^p(npc + n^2 + s))$	$O(s^p(npc^2 + n^2 + s))$
5	$O(2^n(nd + s))$	$O(2^n(nd + s))$	$O(2^n(nd + s))$
6	$O((\frac{n}{q} + 1)^q(s + c + q))$	$O((\frac{n}{q} + 1)^q(s + c + q))$	$O((\frac{n}{q} + 1)^q(cs + c^2 + cq))$

Table 2: Computational Complexities for the Evaluation of Availabilities

Model	Space Complexity
1	$O(1)$
2	$O(t)$
3	$O(1)$
4	$O(p)$
5	$O(nd)$
6	$O(q)$

Table 3: Space Complexities for the Evaluation of Availabilities

5.3 Comparison of the Availabilities

In order to compare the effectiveness of each of these models, we have evaluated availabilities for a wide range of parameters. Due to space limitations, in this paper, we only present a small subset of these results. Similarly, since TFA_s, $TFA'_{x,y}$, and TFA''_s are found to be insensitive to variations in models, we are not presenting these results here. We only present the results for the transaction start availabilities. These results are summarized in Figures 1-7.

Figures 1-3 compare the availabilities obtained from the six models. The following assumptions are made for models 1-6:

1. In Model 2, we assume that the d data objects are grouped into n data groups each containing d/n data objects. This is similar to the assumptions in [13].

2. In Model 3, we assume that each of the n nodes in the system is allocated *exactly* the same number of data objects (equal to dc/n).

3. In Model 4, we assume that $d/2$ data objects have c copies, $d/4$ data objects have $c+1$ copies, and the rest have $c-1$ copies. This keeps the average copies the same (i.e., c) but brings a copy variation factor into consideration.

4. In Model 5, we assume that the d data objects are allocated systematically so that the copies of the i^{th} data object are allocated, in a circular manner, to the nodes starting from $(i \oplus n) + 1$.

5. In Model 6, we assume that $n/3$ nodes have reliability $r - 0.1$, $n/3$ have reliability $r + 0.1$ and the rest have a reliability r. [10]

Figure 1 summarizes the results for read-only transactions with read-one/write-all policy. Figure 2 presents these results for transactions (read-only or read-write) with majority-read/majority-write protocol. Finally, Figure 3 summarizes the results for read-write transactions with read-one/write-all policy. From these results, we make the following observations:

[10] When $r = 0.95$, we assume that $n/3$ nodes have reliability $r - 0.5$, $n/3$ have reliability $r + 0.05$ and the rest have a reliability r.

- For read-only transactions (with read-one/write-all policy),

  (i) Evaluations with models 1 and 3 are close over the entire range of s and r.

  (ii) Evaluations with models 2 and 5 are also close over the entire range of s and r. This may be explained by the fact that the number of groups $g = n = 10$ for model 2 and the systematic distribution for model 5 implicitly results in 10 groups. However, they do differ in the manner in which these groups are distributed.

  (iii) For $r \geq 0.95$, evaluations with all models, excepting model 4, are quite close.

  (iv) Evaluations with model 4 appear to significantly deviate from all other models for $r \geq 0.75$. This implies that modeling of the degree of replication is a very important task in availability evaluations.

- For transactions with majority-read/majority-write policy,

  (v) Evaluations with models 1 and 3 appear to be close. Similarly, evaluations with models 2 and 5 are close. In addition, evaluations with model 6 are close to evaluations with models 1 and 3.

  (vi) For $s \geq 25$, the availabilities appear to be independent of the read-set size. This implies that computations for $s > 25$ are redundant.

  (vii) The evaluations with models 2 and 5 seem to differ at higher values of n. The evaluations with the other four models are close for $n = 20$. This is an interesting observation.

  (viii) Once again, the variations in degree of replication of individual data objects appears to have a dominating effect on availability evaluations.

- For read-write transactions with read-one/write-all policy,

  (ix) The availabilities for $s \geq 5$ are significant only when $r \geq 0.99$.

(x) Since the availabilities are generally low, the effect of the differences in the models seem to be insignificant. At high reliabilities (i.e. $r \geq 0.99$), the evaluations with model 4 seem to deviate from the evaluations with the other models.

We will now study the effect of the individual model parameters.

- Models 1 and 3 are very simple, and need no further investigation.
- Evaluations with model 2 represent the effect of data object grouping on availability (Figure 4). As the number of groups is increased, the availability seems to be decreasing. This effect seems to diminish for $g \geq 25$. This effect is insignificant for read-write transactions. Similarly, this effect seems to vanish at high node reliabilities.
- Evaluations with model 4 represent the effect of variations in degrees of replication of data objects (Figure 5). The effect of these variations seem to be insignificant on read-write transactions. The effect of copy variations seem to be more apparent at high node reliabilities. Similarly, this effect seems to be more pronounced on read-only transactions (with read-one/write-all policy) than the other two classes.
- Model 5 represents the effect of data distribution on the availability evaluations. From Figure 6, it may be observed that the distribution effect is only evident at $s \geq 10$. In addition, the effects are more significant for read-only transactions than the other two classes. The effect is less evident at high node reliabilities.
- Model 6 represents the effect of node reliability variations on availabilities. From Figure 7, it may be observed that the variations have almost no effect on availability evaluations.

6 Conclusions

The current investigations on measuring the effect of data distribution, replication, and fault models on transaction availability evaluation have resulted in some very interesting observations. As part of this study, we chose six

models representing six different parametric assumptions that researchers and designers generally tend to make in their analysis. Using probabilistic analysis, we derived expressions for transaction availability for three classes of transactions: read-only (read-one/write-all policy), transactions with majority-read/majority-write policy, and read-write transactions (with read-one/write-all policy). The effect of the six parameters is measured by evaluating availabilities (for different read-set sizes). From here, we conclude that:

- By choosing a proper distributed database model, the computational complexity of transaction availability evaluations can be significantly reduced.
- For values of $s \leq 10$, all models result in almost the same transaction evaluation.
- It is not necessary to evaluate transaction availabilities for values of $s > 25$.
- Evaluations for the read-only transactions (with read-one/write-all policy) are more sensitive to database modeling than the other two classes of transactions.
- The degree of replication of individual (or group) data objects seems to have a significant effect on transaction availabilities. Thus, when different data objects have different copies, adopting average degree of replication to represent ant object in a system, may not result in accurate availability evaluations.
- The actual distribution of data object copies has some, if not significant, impact on availability evaluation.
- In a heterogeneous environment where different nodes may have different reliabilities, it is sufficient to represent each node by the average node reliability, without affecting the availability evaluations.
- Data object grouping (logical or physical) does not seem to effect the accuracy of availability evaluations as long as the number of groups is not too small (e.g. When $d = 1000$, $g \geq 25$ is sufficient).

Distributed database designers and researchers can utilize these results in choosing appropriate parameters that would result in reduced computational requirements without sacrificing the resulting accuracy of the design and analysis of these systems.

Appendix

Model 3 $< A_1, B_2, C_1, D_1, E_1 >$:
Here, we assume that each node has exactly the same number of data objects $(= \frac{d*c}{n})$.

$$TSA_s = \sum_{k=1}^{n} \binom{n}{k} r^k (1-r)^{n-k} \frac{\binom{x_k}{s}}{\binom{d}{s}} \tag{23}$$

$$TSA'_{x,y} = \sum_{k=1}^{n} \binom{n}{k} r^k (1-r)^{n-k} \frac{\binom{y_k}{y}}{\binom{d}{y}} \frac{\binom{x_k-y}{x}}{\binom{d-y}{x}} \tag{24}$$

$$TSA''_s = \sum_{k=m}^{n} \binom{n}{k} r^k (1-r)^{n-k} \frac{\binom{z_k}{s}}{\binom{d}{s}} \tag{25}$$

$$TFA_s = e^{-n_s t\lambda} \tag{26}$$

$$TFA'_{x,y} = e^{-n'_{x,y} t\lambda} \tag{27}$$

$$TFA''_s = e^{-n''_s t\lambda} \tag{28}$$

$$x_k = d\left[1 - \frac{\binom{n-k}{c}}{\binom{n}{c}}\right]$$

$$y_k = d\left[\frac{\binom{k}{c}}{\binom{n}{c}}\right]$$

$$z_k = d\sum_{l=m}^{c} \frac{\binom{k}{l}\binom{n-k}{c-l}}{\binom{n}{c}}$$

$$m = \left\lceil \frac{c+1}{2} \right\rceil$$

Model 4 $< A_1, B_1, C_2, D_1, E_1 >$:
Here, each data object may have its own degree of replication specified. For an efficient computation, we classify the data objects into p categories $(1 \leq p \leq n)$ based its degree of replication. d_l denoted the number of data objects in the l^{th} category where each object has c_l $(1 \leq c_l \leq n)$ copies.

$$TSA_s = \sum_{CA} Pr(CA) \sum_{k=1}^{\lceil \frac{n}{2} \rceil} \binom{n}{k} r^k (1-r)^{n-k} \prod_{l=1}^{p} \left[1 - \frac{\binom{n-k}{c_l}}{\binom{n}{c_l}}\right]^{a_l} \tag{29}$$

$$TSA'_{x,y} = \sum_{CA'} Pr(CA') \sum_{CA''} Pr(CA'') \sum_{k=1}^{n} \binom{n}{k} r^k (1-r)^{n-k} \prod_{l=1}^{p} \left[1 - \frac{\binom{n-k}{c_l}}{\binom{n}{c_l}} \right]^{a'_l} \prod_{l=1}^{p} \left[\frac{\binom{k}{c_l}}{\binom{n}{c_l}} \right]^{a''_l} \quad (30)$$

$$TSA''_s = \sum_{CA} Pr(CA) \sum_{k=m}^{n} \binom{n}{k} r^k (1-r)^{n-k} \prod_{l=1}^{p} \left[\sum_{l'=m_l}^{c_l} \frac{\binom{k}{l'} \binom{n-k}{c_l - l'}}{\binom{n}{c_l}} \right]^{a_l} \quad (31)$$

$$CA = < a_1, a_2, \ldots, a_p >, \sum_{k=1}^{p} a_k = s, \forall k\ 1 \le k \le p\ 0 \le a_k \le d_k$$

$$CA' = < a'_1, a'_2, \ldots, a'_p >, \sum_{k=1}^{p} a'_k = x,\ \forall k\ 1 \le k \le p\ 0 \le a_k \le d_k$$

$$CA'' = < a''_1, a''_2, \ldots, a''_p >, \sum_{k=1}^{p} a''_k = y,\ \forall k\ 1 \le k \le p\ 0 \le a''_k \le d_k - a'_k$$

$$Pr(CA) = \frac{\binom{d_1}{a_1}\binom{d_2}{a_2}\cdots\binom{d_p}{a_p}}{\binom{d}{s}}$$

$$Pr(CA') = \frac{\binom{d_1}{a'_1}\binom{d_2}{a'_2}\cdots\binom{d_p}{a'_p}}{\binom{d}{x}}$$

$$Pr(CA'') = \frac{\binom{d_1-a'_1}{a''_1}\binom{d_2-a'_2}{a''_2}\cdots\binom{d_p-a'_p}{a''_p}}{\binom{d-x}{y}}$$

$$m_l = \left\lceil \frac{c_l + 1}{2} \right\rceil$$

The expressions for TFA_s, $TFA'_{x,y}$, and TFA''_s are the same as in Equations (26) - (28).

Model 5 $< A_1, B_1, C_1, D_2, E_1 >$:
Here, we assume that the entire data distribution is available as a dictionary, GD.

$$TSA_s = \sum_{S} Pr(S) \frac{\binom{f(S)}{s}}{\binom{d}{s}} \quad (32)$$

$$TSA'_{x,y} = \sum_{S} Pr(S) \frac{\binom{f'(S)}{y}}{\binom{d}{y}} \frac{\binom{f(S)-y}{x}}{\binom{d-y}{x}} \quad (33)$$

$$TSA''_s = \sum_S Pr(S) \frac{\binom{f''(S)}{s}}{\binom{d}{s}} \quad (34)$$
$$Pr(S) = r^{f'''(S)}(1-r)^{n-f'''(S)}$$

where

S - Node status vector; $S_j = 1 \Rightarrow$ Node j is up; $S_j = 0 \Rightarrow$ Node j is down.

$f(S)$ - The number of data objects available for read with the given node status vector (S). This is computed by scanning the columns of the GD matrix corresponding to the live nodes (as given by S).

$f'(S)$ - The number of data objects available for update (i.e. all c copies of these data objects are available at the live nodes) with the given node status vector (S). This is also computed by scanning the columns of the GD matrix corresponding to the live nodes (as given by S).

$f''(S)$ - The number of data objects available with a majority of copies among the available nodes. As before this is computed by scanning the columns of the GD matrix corresponding to the live nodes (as given by S).

$f'''(S)$ - The number of nodes available (or up) as indicated by the vector S.

Model 6 $< A_1, B_1, C_1, D_1, E_2 >$:

Here each node may have its own reliability. For computational purpose, we categorize the nodes based on their reliability. We assume that there are q $(1 \leq q \leq n)$ such categories. We let n_i to represent the number of nodes with reliability r_i, and a_i to represent the number of currently active (or up) nodes with this reliability.

$$TSA_s = \sum_{NA} Pr(NA) \left[1 - \frac{\binom{n-\sum_{i=1}^{q} a_i}{c}}{\binom{n}{c}}\right]^s \prod_{k=1}^{q} \left[\binom{n_k}{a_k} r_k^{a_k} (1-r_k)^{n_k - a_k}\right] \quad (35)$$

$$TSA'_{x,y} = \sum_{NA} Pr(NA) \left[\frac{\binom{\sum_{i=1}^{q} a_i}{c}}{\binom{n}{c}}\right]^y \left[1 - \frac{\binom{n-\sum_{i=1}^{q} a_i}{c}}{\binom{n}{c}}\right]^x \prod_{k=1}^{q} \left[\binom{n_k}{a_k} r_k^{a_k} (1-r_k)^{n_k - a_k}\right] \quad (36)$$

$$TSA''_s = \sum_{NA} Pr(NA) \left[\sum_{k=m}^{c} \frac{\binom{\sum_{i=1}^{q} a_i}{k} \binom{n-\sum_{i=1}^{q} a_i}{c-k}}{\binom{n}{c}}\right]^s$$

$$\prod_{k=1}^{q}\left[\binom{n_k}{a_k} r_k^{a_k}(1-r_k)^{n_k-a_k}\right] \tag{37}$$

$$Pr(NA) = \frac{\binom{n_1}{a_1}\binom{n_2}{a_2}\cdots\binom{n_q}{a_q}}{\binom{n}{\sum_{k=1}^{q} a_k}}$$

$$NA = < a_1, a_2, \ldots, a_q >,\ \forall i = 1, 2, \ldots, q\ 0 \leq a_i \leq n_i$$

$$\sum_{i=1}^{q} n_i = n$$

References

[1] M. Ahamad and M.H. Ammar, "Performance characterization of quorum-consensus algorithms for replicated data," *Tech. Report*, GIT-ICS-86/123, Georgia Institute of Technology, 1986.

[2] M.D. Beaudry, "Performance-related reliability measures for computing systems," *IEEE Trans. Computers*, Vol. C-27, pp. 540-547, June 1978.

[3] P.A. Bernstein and N. Goodman, "Concurrency control in distributed database systems," *ACM Computing Surveys*, Vol. 13, pp. 185-221, June 1981.

[4] B. Bhargava and L. Lilien, "A review of concurrency and reliability issues in distributed database systems," *Concurrency Control and Reliability Issues in Distributed Systems*, B. Bhargava (Ed.), Van Nostrand Reinhold Co, pp. 1-84, 1987.

[5] S. Ceri, G. Martella, and G. Pelagatti, "Optimal file allocation for a distributed database on a network of minicomputers," *Proc. International Conference on Data Bases*, University of Aberdeen, pp. 216-237, July 1980.

[6] E.G. Coffman, E. Gelenbe, and B. Plateau, "Optimization of number of copies in a distributed database," *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 1, pp. 78-84, 1981.

[7] S.B. Davidson, "Analyzing partition failure protocols," *Technical Report*, MS-CIS-86-05, Dept. of Computer Science, University of Pennsylvania, January 1986.

[8] H. Garcia-Molina, "Performance evaluation of the update algorithms for replicated data in a distributed database," *Ph.D. Dissertation*, Computer Science Department, Stanford University, June 1979.

[9] H. Garcia-Molina and J. Kent, "Performance evaluation of reliable distributed systems," *Concurrency Control and Reliability in Distributed Systems*, B.K. Bhargava (Ed.), pp. 454-488, 1987.

[10] B. Gavish and H. Pirkul, "Computer and database location in distributed computer systems," *IEEE Transactions on Computers,* Vol. C-35, No. 7, pp. 583-590, July 1986.

[11] R. Mukkamala, "Design of partially replicated distributed database systems," *Technical Report,*, TR 87-04, Department of Computer Science, University of Iowa, July 1987.

[12] R. Mukkamala, S.C. Bruell, and R.K. Shultz, "Design of partially replicated distributed database systems: an integrated approach," *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 187-196, May 1988.

[13] R. Mukkamala, "Measuring the effect of data distribution and replication policies on performance evaluation of distributed database systems," *Proc. Fifth International Conference on Data Engineering,* February 1989.

[14] R. Mukkamala, "Measuring the effect of data replication and fault models on transaction availability analysis," *Technical Report*, TR 89-35, Department of Computer Science, Old Dominion University, May 1989.

[15] R. Mukkamala, "Performance evaluation of distributed database systems," *Technical Report*, TR 89-43, Department of Computer Science, Old Dominion University, June 1989.

[16] K.C. Sevcik, "Comparison of concurrency control methods using analytic methods," *Proc. Information Processing 83*, R.E.A. Mason (Ed.), North-Holland, September 1983.

[17] L.E. Stanfel, "Applications of clustering to information system design," *Information processing and Management*, Vol. 19, No. 1, pp. 37-50, 1983.

[18] R.B. Thomas, "A majority consensus approach to concurrency control for multiple copy databases," *ACM Transactions on Database Systems,* Vol. 4, No. 2, pp. 180-209, June 1979.

Figure 1a. n=10, d=1000, c=3, r=0.4

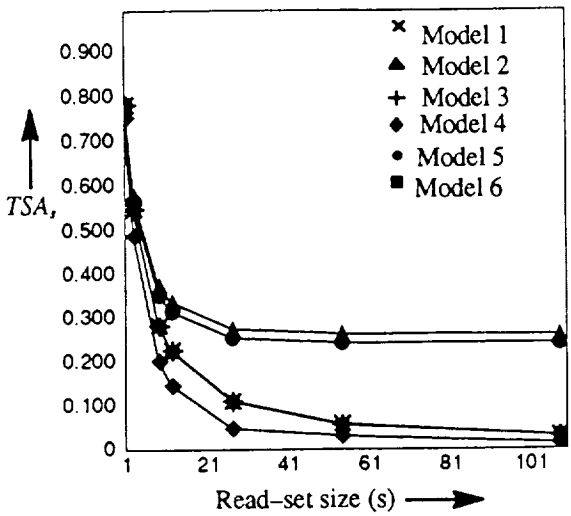


Figure 1b. n=10, d=1000, c=3, r=0.75

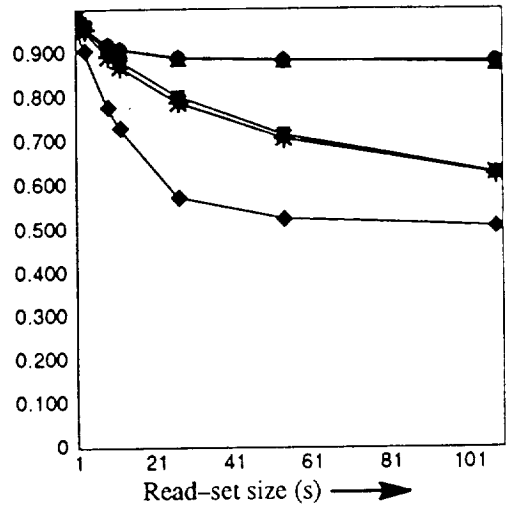


Figure 1c. n=10, d=1000, c=3, r=0.90

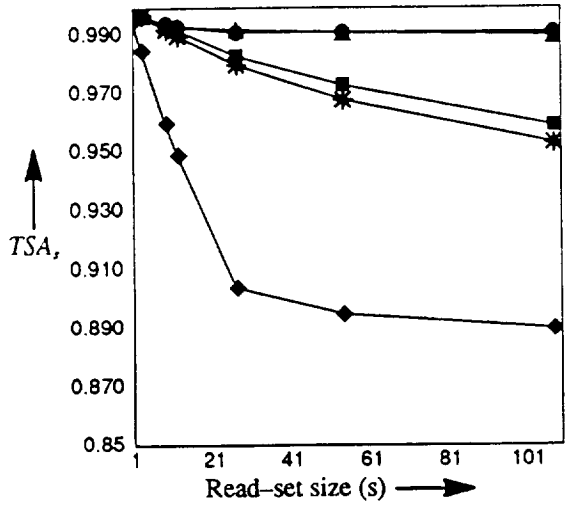


Figure 1d. n=10, d=10000, c=3, r=0.75

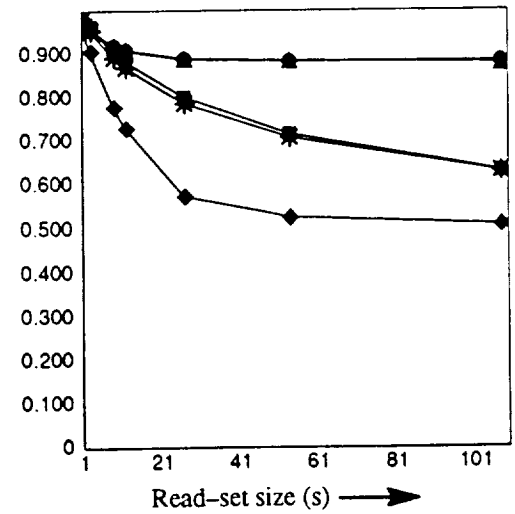
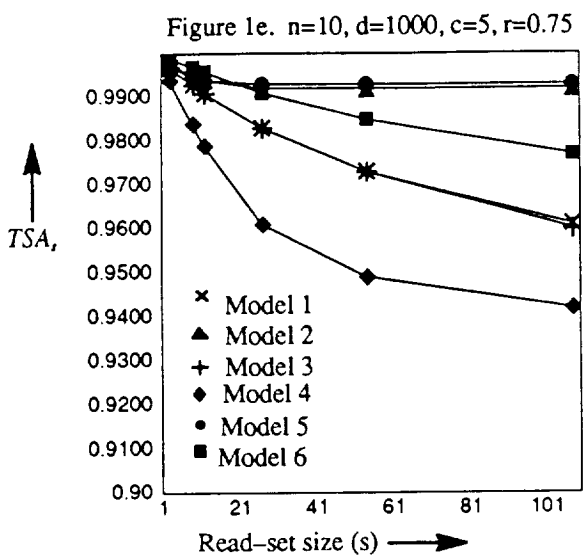
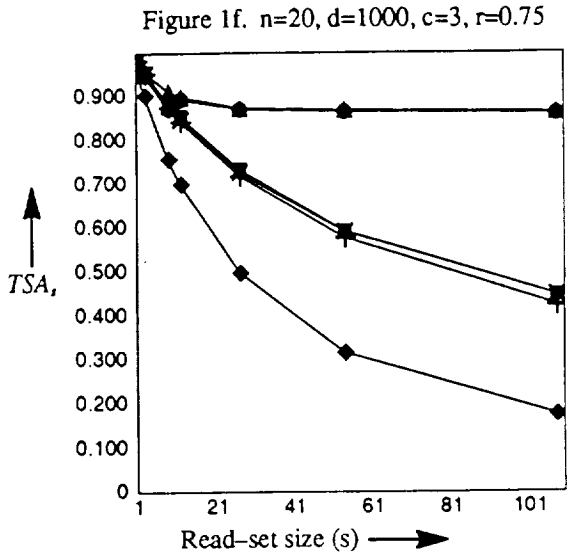


Figure 1. Transaction Start Availabilities for Read–Only Transactions (with Read–one/Write–all policy)





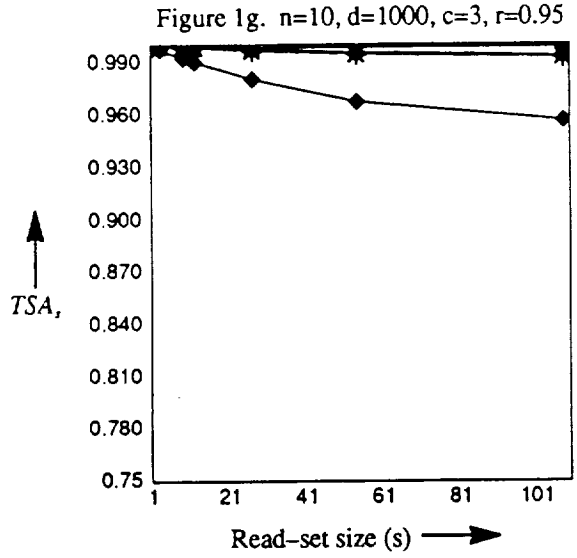


Figure 1 (Continued). Transaction Start Availabilities for Read–only Transactions (Read–one/Write–all Policy)

Figure 2a. n=10, d=1000, c=3, r=0.4

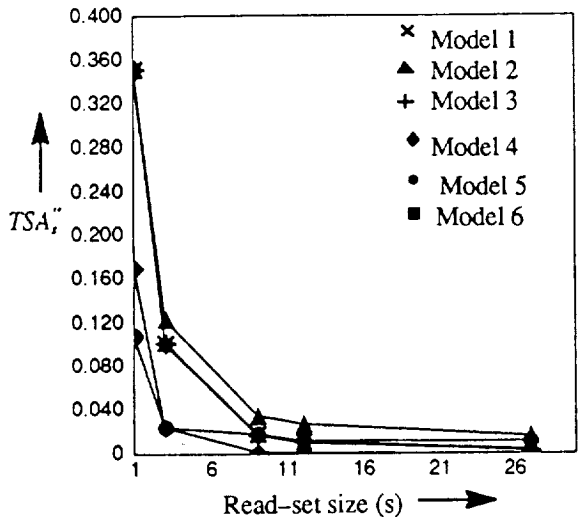


Figure 2b. n=10, d=1000, c=3, r=0.75

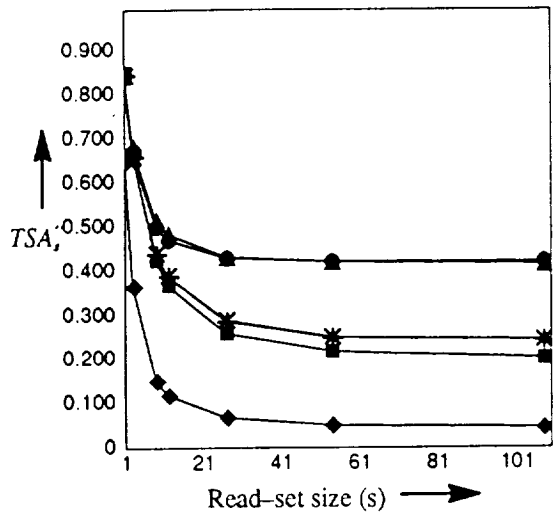


Figure 2c. n=10, d=1000, c=3, r=0.90

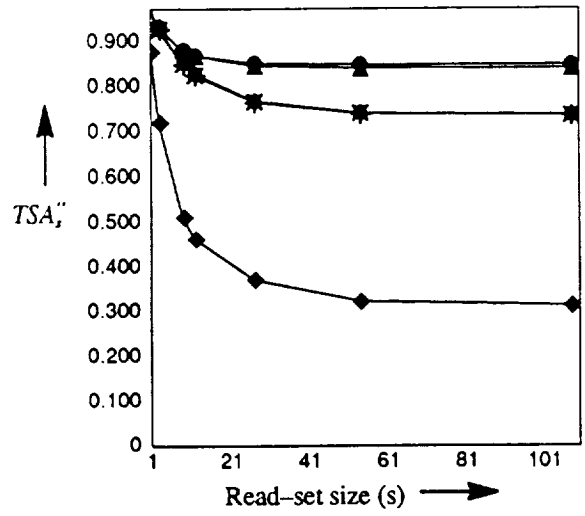


Figure 2d. n=10, d=10000, c=3, r=0.75

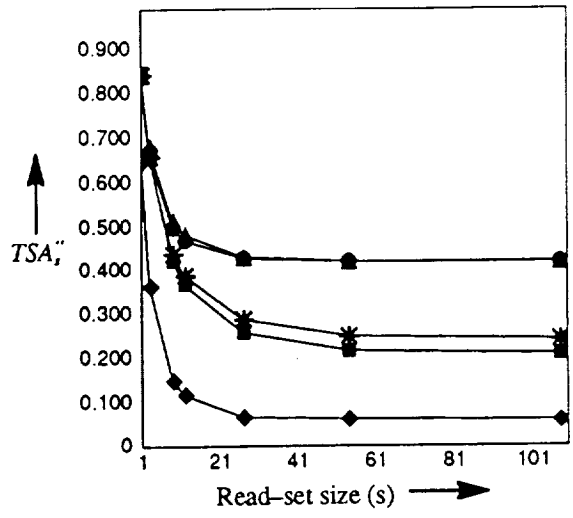


Figure 2. Transaction Start Availabilities with Read–Majority/Write–Majority Protocol

Figure 2e. n=10, d=1000, c=5, r=0.75

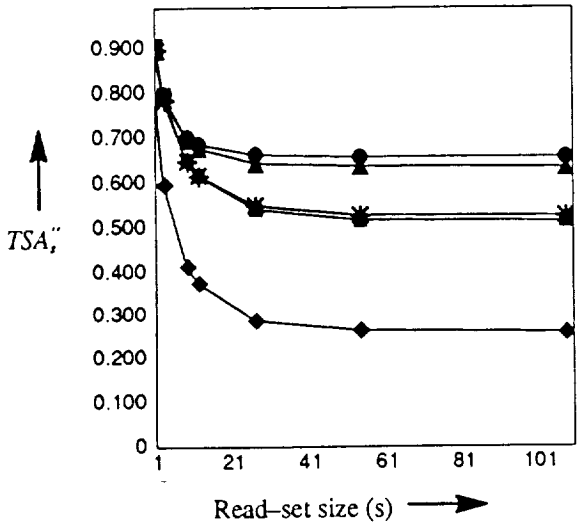


Figure 2f. n=20, d=1000, c=3, r=0.75

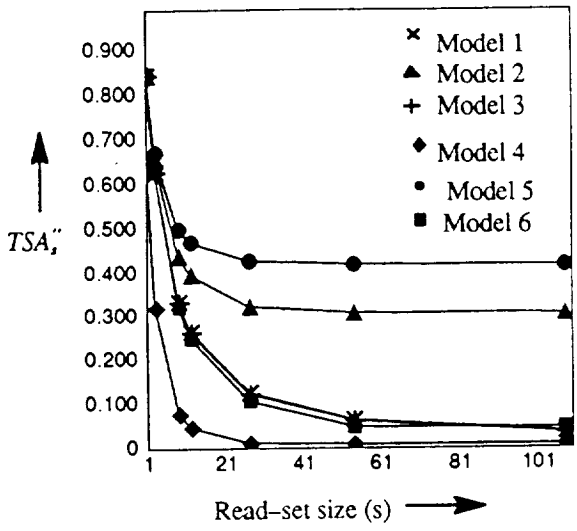


Figure 2g. n=10, d=1000, c=3, r=0.95

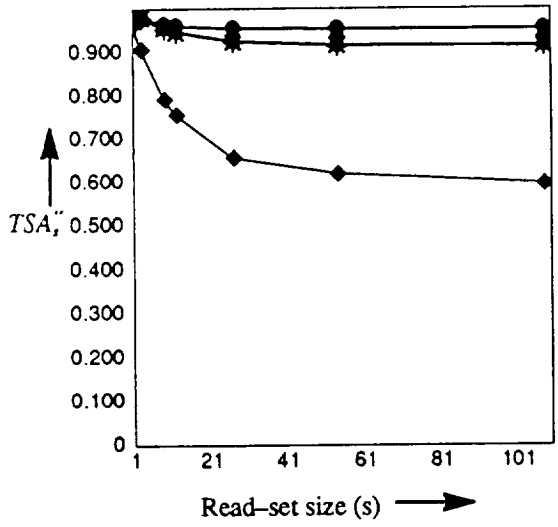


Figure 2 (Contd.). Transaction Start Availabilities with Read–Majority/Write–Majority Protocol

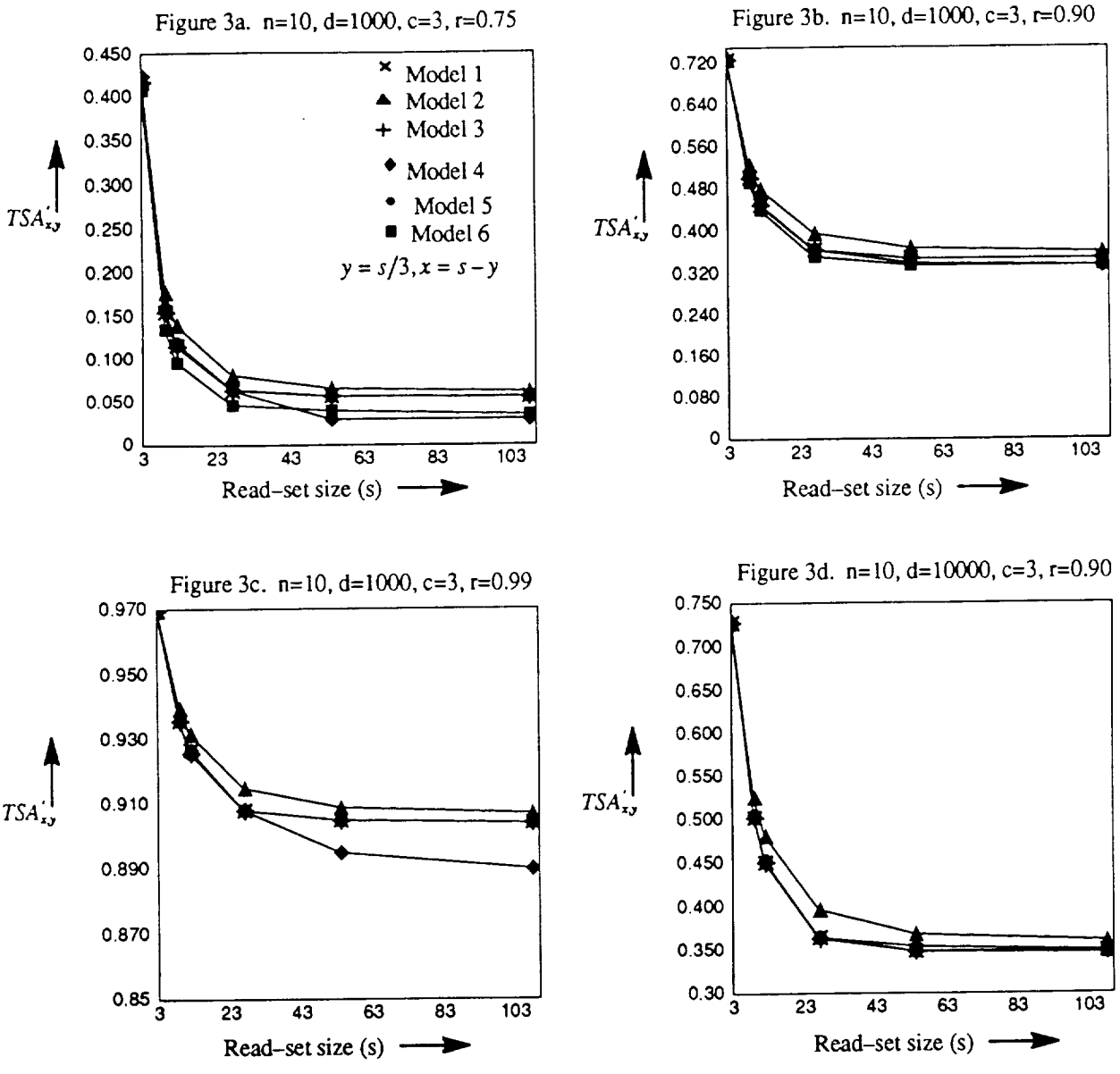


Figure 3. Transaction Start Availabilities for Read-write Transactions (with Read-one/Write-all Policy)

Figure 3e. n=10, d=1000, c=5, r=0.90

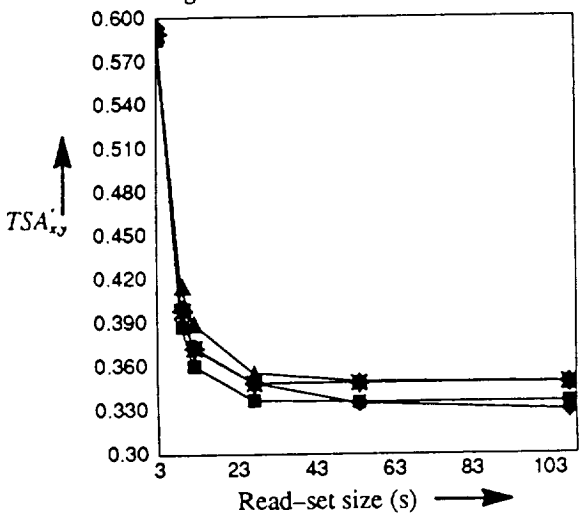


Figure 3f. n=20, d=1000, c=3, r=0.90

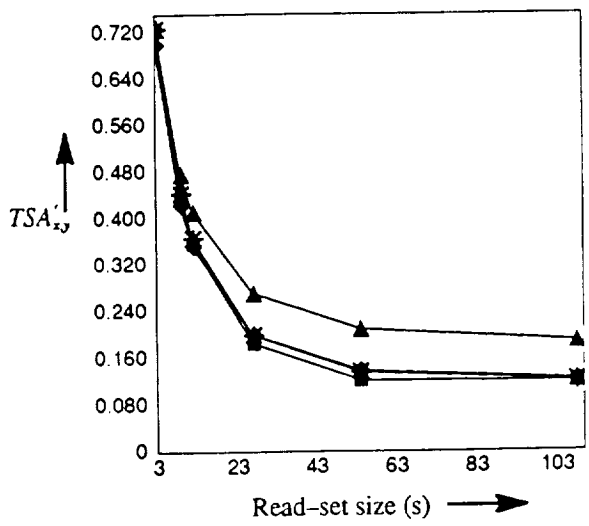


Figure 3 (Contd.) . Transaction Start Availabilities for Read-write Transactions (with Read-one/Write-all Policy)

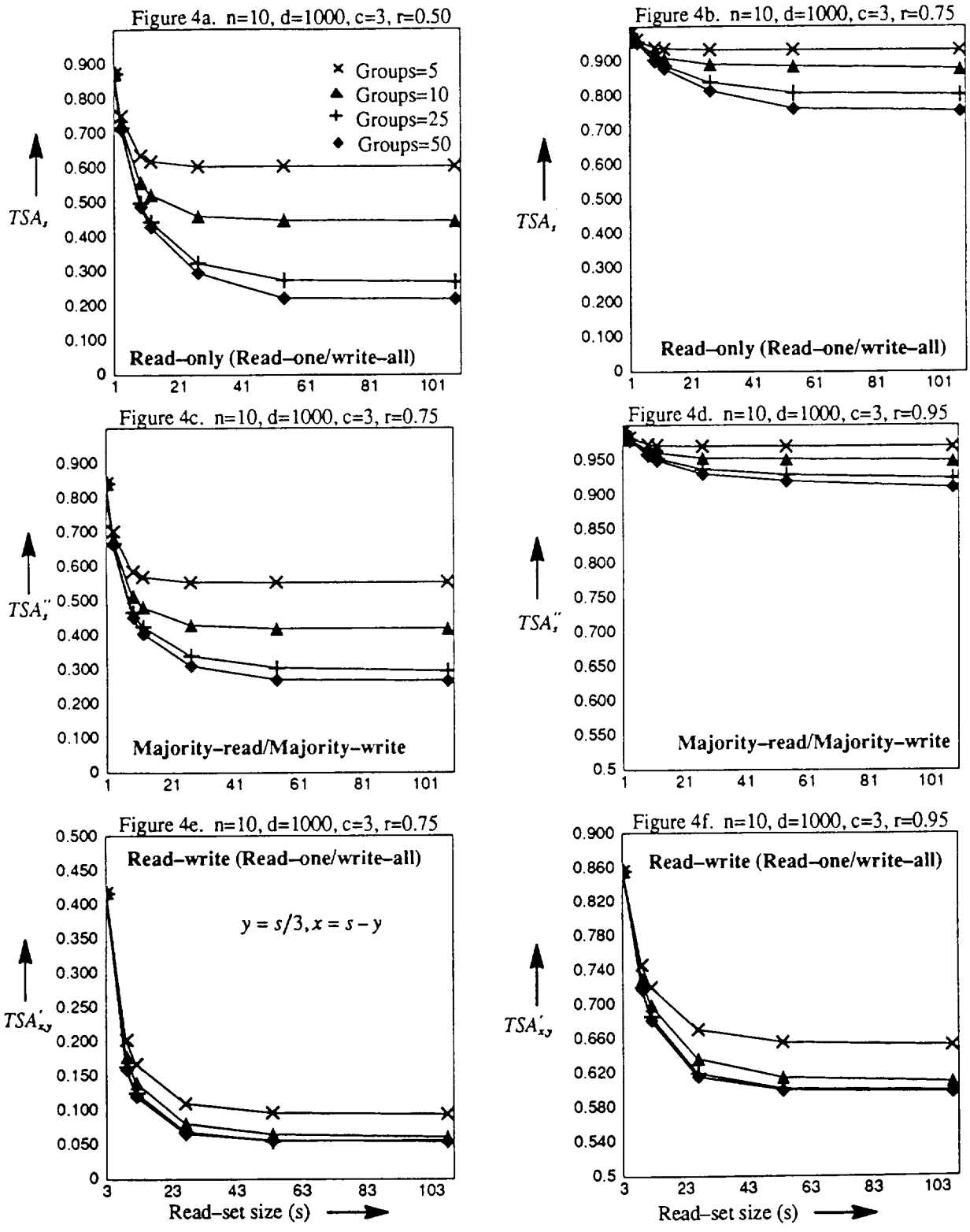


Figure 4. Illustration of the Effects of the Number of Groups on Availability Metrics (Model 2)

Figure 5a. n=10, d=1000, c=3, r=0.5

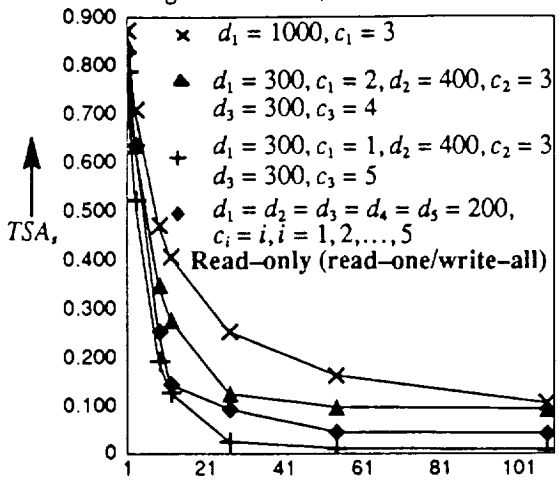


Figure 5b. n=10, d=1000, c=3, r=0.75

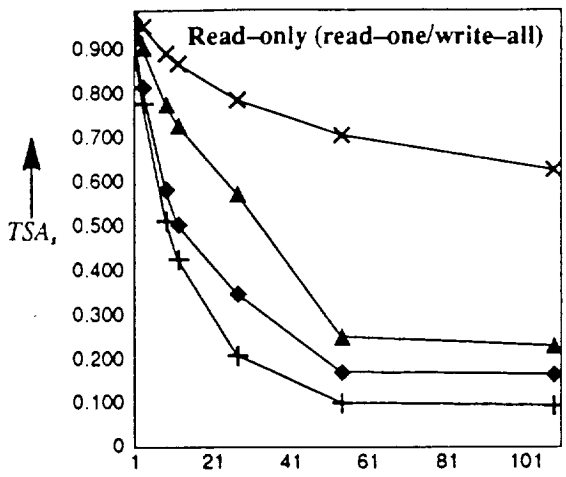


Figure 5c. n=10, d=1000, c=3, r=0.75

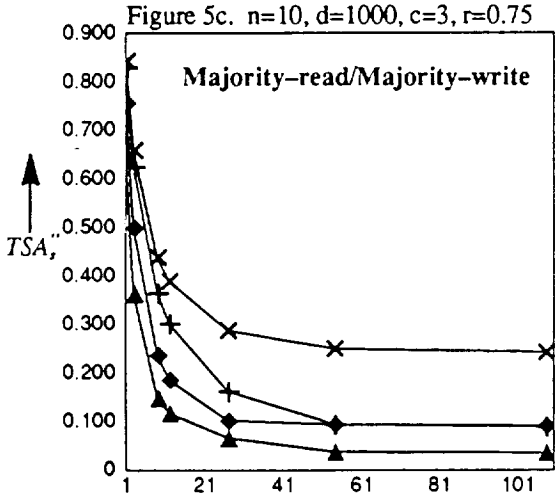


Figure 5d. n=10, d=1000, c=3, r=0.95

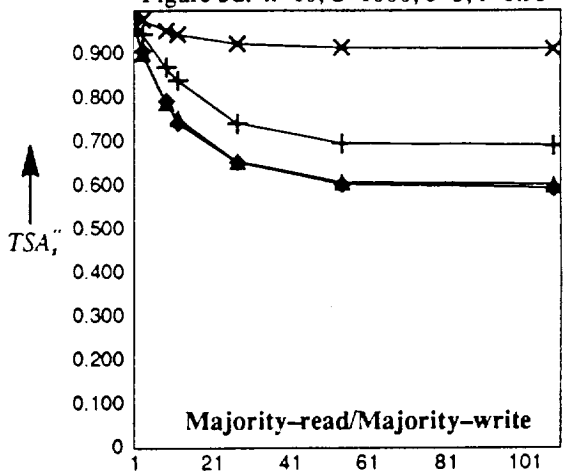


Figure 5e. n=10, d=1000, c=3, r=0.75

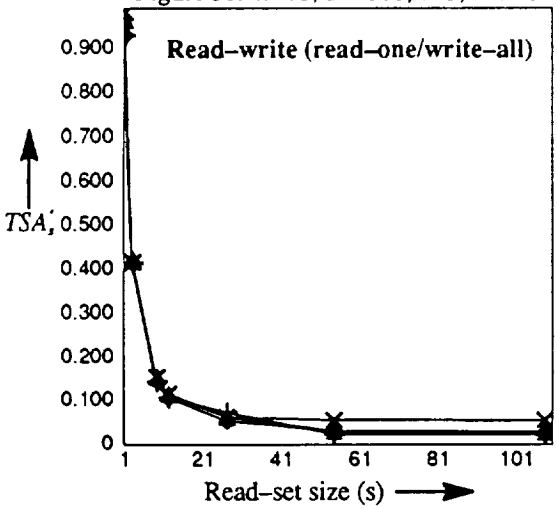


Figure 5f. n=10, d=1000, c=3, r=0.95

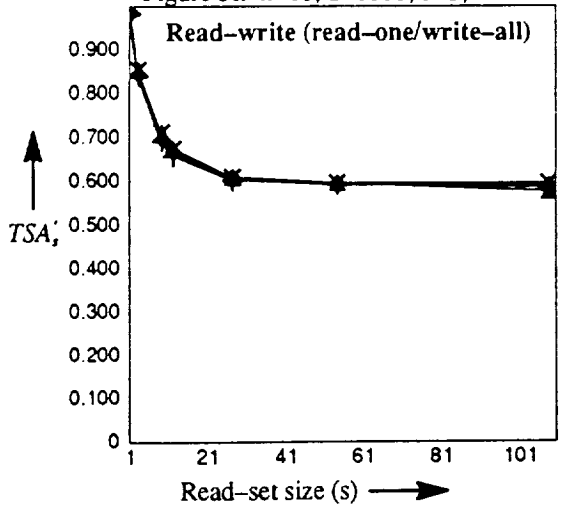


Figure 5. Illustration of the Effect of Copy variations on Availability (Model 4)

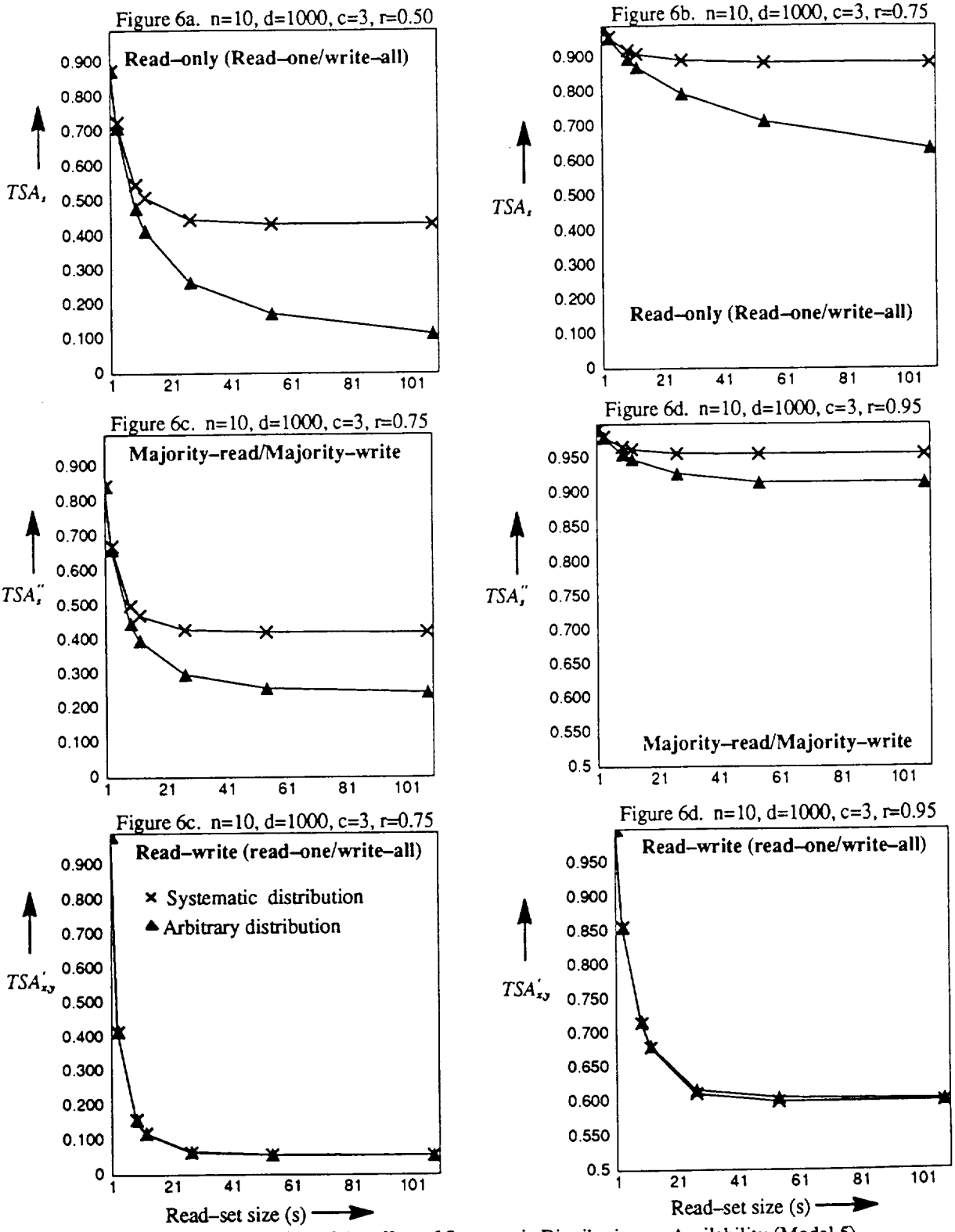


Figure 6. Illustration of the effect of Systematic Distribution on Availability (Model 5)

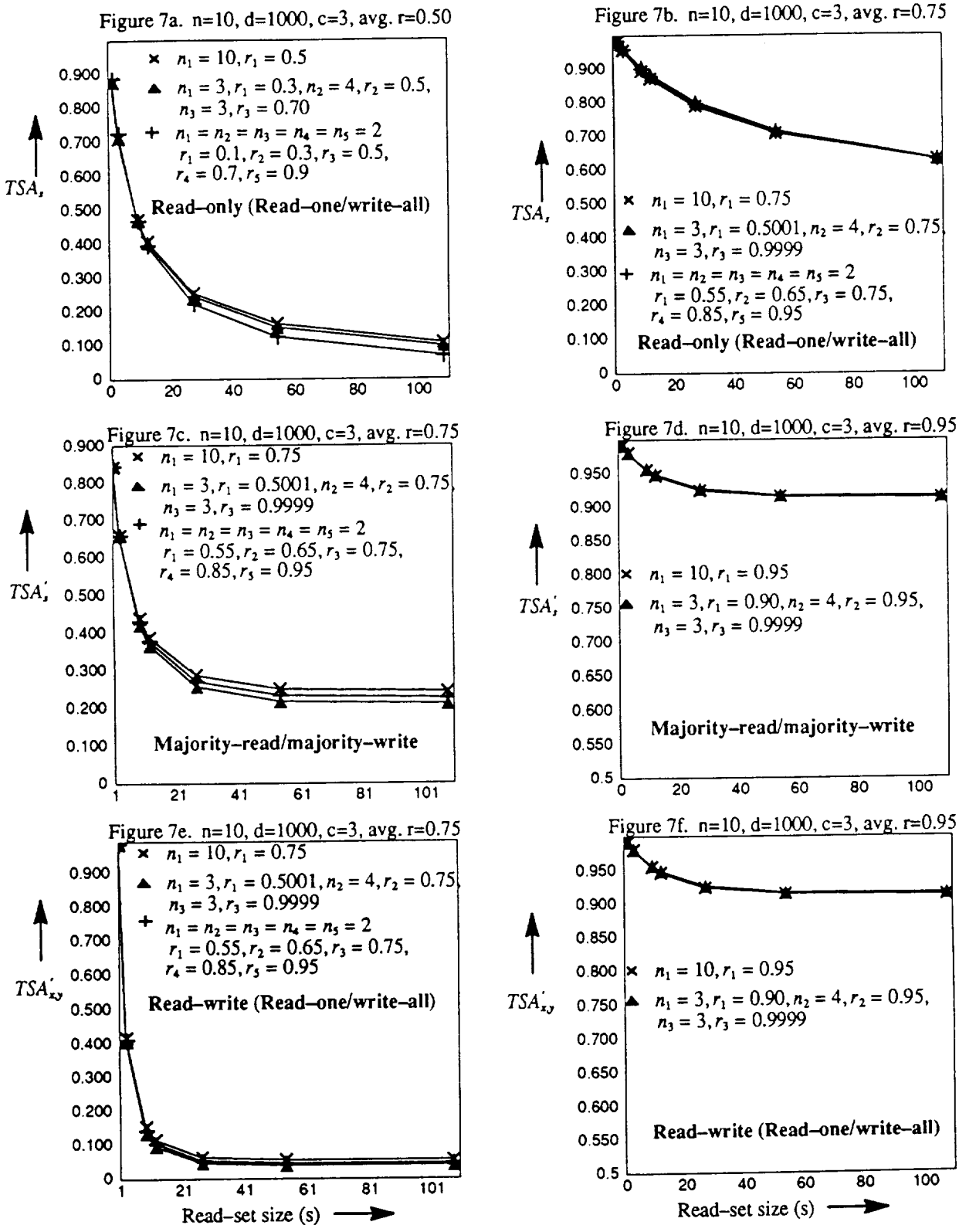


Figure 7. Illustration of the Effect of Reliability Variations on Availability (Model 6)

IEEE PROCEEDINGS OF THE

SOUTHEASTCON '91

NASA

Volume 2

91CH2998-3

N91-23974

Performance Analysis of Static Locking in Replicated Distributed Database Systems

Yinghong Kuang
Ravi Mukkamala
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529.

Abstract

Data replications and transaction deadlocks can severely affect the performance of distributed database systems. Many current evaluation techniques ignore these aspects, because it is difficult to evaluate through analysis and time-consuming to evaluate through simulation. In this paper, we use a technique that combines simulation and analysis to closely illustrate the impact of deadlock and evaluate performance of replicated distributed database with both shared and exclusive locks.

1. Introduction.

A distributed database system (DDS) is a collection of cooperating nodes each containing a set of data items. A user transaction can enter such a system at any of these nodes. The receiving node, often referred to as the *coordinating* node, undertakes the task of locating the nodes that contain the data items required by a transaction.

In order to maintain database consistency and correctness in the presence of concurrent transactions, several concurrency control protocols have been proposed [1]. Of these, the most commonly used are time-stamping and locking protocols. Locking protocols have been widely used in both commercial and research environments. In static locking, prior to start of execution, a transaction needs to acquire either a shared-lock (for read operations) or an exclusive lock (for update operations) on each of the relevant data items.

Data replication is used to improve the performance of local transactions and the availability of databases. In replicated databases, one data item may have more than one copy in the system. Replica control algorithms are used to maintain the consistency among these copies. One of these is the read-one/write-all protocol. With this protocol an exclusive lock need to acquire an exclusive lock from every copy of the data item . For a shared lock to succeed, any one copy of the data item has to be share locked. When transactions with conflicting lock requests are initiated concurrently, they could be possibly blocked due to a deadlock.

There are two major ways to evaluate the performance of distributed systems: simulation and analysis. Simulation is a conceptually tractable technique, but requires large computation time. On the other hand, analysis is computationally faster but may not be tractable for all problems. In [4], Shyu and Li proposed an elegant analysis model to evaluate the response time and throughput of transactions in a non-replicated DDS. Assuming *exclusive* locking (i.e., only write operations), they model the queue of lock requests at an object as an M/M/1 queue [3]. This results in a closed-form for the waiting time distribution at a node, expressed in terms of the average rates of arrivals of requests and the average lock-holding time. With shared lock and replications added into the picture, it is very difficult to have a close model for it. Because of the limitations of simulation and analysis, we develop a technique that combines simulation and analysis.

This paper is organized as the follows. In Section 2, we describe the model used in our performance evaluation. In Section 3, we propose an evaluation technique. In Section 4, we illustrate the results. Finally, Section 5 has the conclusions.

2. Model

Our model has the following parameters:

- There are n nodes.
- There are d data items in a DDS.
- A data item may be located at exactly c number of nodes. The dc data copies are uniformly distributed across the n nodes.
- Each transaction accesses k data items.
- r is the read ratio. So among k data items to be accessed, rk are accessed only for read operations, and the rest are for read-write operations. Due to the read-one/write-all replica control policy, a transaction must procure rk shared locks for rk read operations and $(1-r)kc$ exclusive locks for the $(1-r)k$ read-write operations.
- Each data item is equally likely to be accessed by a transaction.
- Transaction arrivals into the system is a Poisson process with rate λ.
- The communication delay between any two nodes is exponentially distributed with mean $\bar{t}$.
- The average execution time of a transaction, once the locks are obtained, is $\bar{s}$.
- The deadlock mechanism is invoked every τ seconds.
- After an abortion of a transaction, it takes an average of ω seconds for this transaction to be restarted.
- μ is the service rate of transactions.
- b is the lock-holding time.
- λc is the arrival rate at each data copy.

[1]This research was supported in part by the NASA Langley Research Center under contracts NAG-1-1114 and NAG-1-1154.

CH2998-3/91/0000-0698$01.00©1991IEEE

3. Performance Evaluation Technique

Our technique consists of two stages. In the first stage, the average transaction response time and throughput are calculated by ignoring the deadlock. This is an iterative step involving simulation and analysis. In the second stage, the probabilities of transaction conflicts and deadlocks are computed by probability models. These probabilities are used, in turn, to compute the response time and throughput in the presence of deadlocks.

Stage 1:
Initially, we assume that there are no lock conflicts between transactions. Each transaction has to procure rk shared lock on data copies and $(1-r)kc$ exclusive locks on data copies. When a transaction has got all the lock grants from these data objects, it can go ahead with execution.

This procedure is summarized in the following 6 steps.

1. Initialize lock-holding time(b) to be $1/\mu$.
2. Given the total rate of transaction arrival(λ), the shared lock ratio(r), the number of data items(d), the number of data items required by each transaction(k) and the number of replications(c), derive the arrival rate at each data copy(λc).
3. With the arrival rate at each data copy(λc), the average lock-holding time(b), and the transmission time(t) we can simulate the queue at a data copy to arrive wait-time(w) distribution. With this distribution we can calculate the response time of transactions.
4. With the average service time of transactions($1/\mu$), and the transmission time, we can derive a new lock-holding time(b').
5. Set b to this new lock-holding time b'.
6. If the old and new lock-holding time are sufficiently close, stop the iteration. Otherwise, go back to step 3.

At the end of stage 1 the response time without the consideration of transaction deadlocks is obtained.

Stage 2:
This stage considers transaction conflicts and computes the deadlock probability. Here the probabilities of transaction deadlock and restart are computed. These are then used to compute response time and throughput in the presence of deadlocks.

Assume there are two transactions T1 and T2. Let RS, WS be the read and write sets of transactions respectively.

1. Let fs_i be the probability that the readset of T1 has i data items overlapping with the writeset of T2, i.e. $|RS(T1) \cap WS(T2)| = i$.
2. Let fe_{ij} be the probability that given $|RS(T1) \cap WS(T2)| = i$, the writeset of T1 has j data items overlaping with the readset and writeset of T2, i.e. the probability that $|WS(T1) \cap (RS(T2) \cup WS(T2))| = j$.

Clearly,

$$fs_i = \frac{\binom{k-rk}{i}\binom{d-k+rk}{rk-i}}{\binom{d}{rk}} \quad (1)$$

$$fe_{ij} = \frac{\binom{k-i}{j}\binom{d-rk-k+i}{k-rk-j}}{\binom{d-rk}{k-rk}} \quad (2)$$

It can also be noted that $fs_i fe_{ij}$ is the probability that:
$|\text{Read-set(T1)} \cap \text{Write-set(T2)}| = i$
$\wedge\ |\text{Write-set(T1)} \cap (\text{Write-set(T2)} \cup \text{Read-set(T2)})| = j$.
If PW_{ij} is the probability that T1 waits for T2,

$$PW_{ij} = p1 + p2 - p1 * p2 \quad (3)$$

$$p1 = 1 - [1 - (1/2)^c]^i \quad (4)$$

$$p2 = (1 - (1/2)^{cj}) \quad (5)$$

where $p1$ is the probability that T1 waits for T2 for shared locks in readset
and $p2$ is the probability that T1 waits for T2 for exclusive locks in writeset.
Probability that T1 waits for T2 is now given by

$$Pw = \sum_{i=0}^{\min(x,k-x)} \sum_{j=0}^{\min(k-x,k-i)} fs_i fe_{ij} PW_{ij} \quad (6)$$

With this probability of waiting and the formulas in [4] we can calculate the probability of a transaction deadlock, the probability of a transaction restart and the probability of a transaction to be blocked by other transactions. And with these probabilities and the time between deadlock detection(τ), we can calculate the response time with consideration of deadlock. (Details are omitted here.)

4. Results

Using this technique, we obtained a number of interesting results that illustrate the effect of deadlocks and number of replications on database performance. These are summarized in Figures 1-5. We make the following observations.

- Transaction response times are quite sensitive to the ratio of shared locks (Figure 1 and 2). Here, we compare the response times when deadlocks are ignored (DI, computed in Stage 1) with those obtained when deadlocks are considered (DC, computed in Stage 2). The effect of deadlocks is more predominant at higher transaction loads and with smaller values of r. When $r = 2/3$, the effect of deadlocks is not significant on response time.
- If we compare Figure 1 and 2 with Figure 3 and 4, it can be observed that the increase in replications results in the larger response time when read ratio is smaller than 1/3.
- Fig. 5 shows the response times with different replication numbers. Here we can see that with both cases when read ratio is 2/3 and 1/3, the response time increases as the number of replications increases. But with read ratio equals 1/3, the increasing rate is much smaller than that with read ratio equals 2/3.

5. Conclusions

In [4], Shyu and Li presented an elegant technique to evaluate the performance of distributed database systems in the presence of deadlocks. Their technique assumed only exclusive locks and thus representing the worst-case effects of deadlocks.

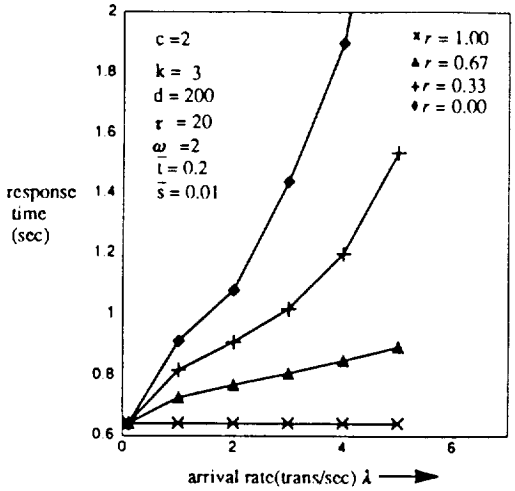


Figure.1 Comparison of response time with different read ratio when deadlock is ignored.

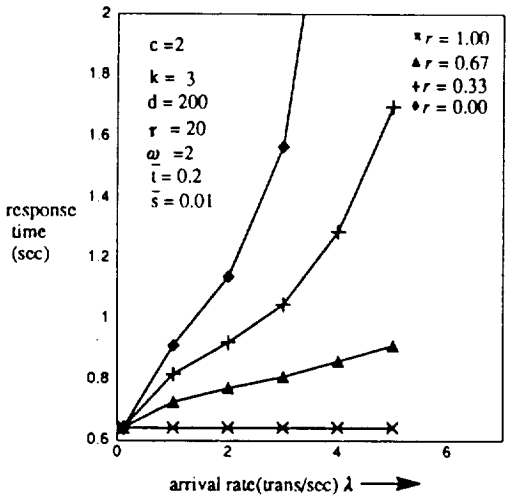


Figure.2 Comparison of response time with different read ratio when deadlock is considered.

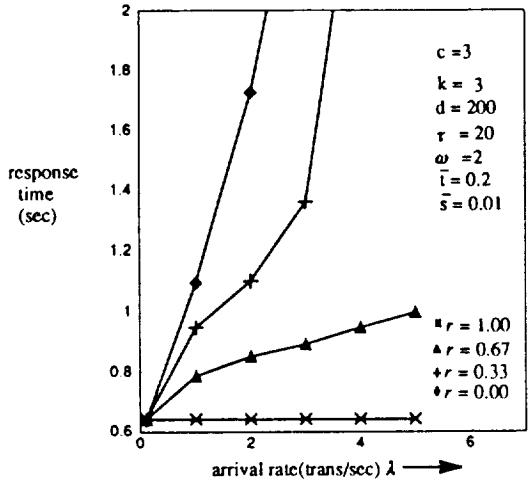


Figure.3 Comparison of response time with different read ratio when deadlock is ignored.

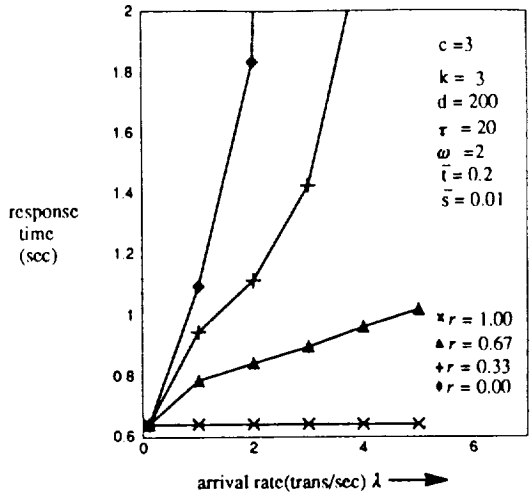


Figure.4 Comparison of response time with different read ratio when deadlock is ignored.

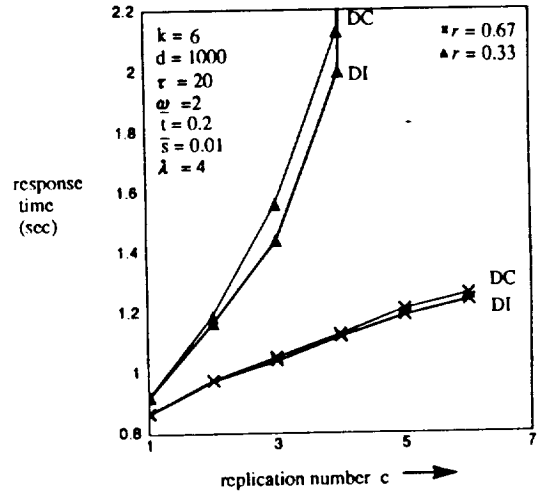


Figure. 5 Comparison of response time with different read ratio with and without deadlock.

DC: Deadlock Considered.
DI: Deadlock Ignored.

In this paper, we have extended their technique to combine sim ulation and analysis. And with this extended technique we allo both shared and exclusive locking and also replications in ou model. We evaluated the the effect of number of data items, th number of data items accessed by each transaction, the ratio read operations on transaction response time and the number replications. These results show the importance of considerin both shared and exclusive lock requests, the deadlock proba bilities as well as the number of replications of database fo response time evaluations.

References

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[2] A. Hac, "A decomposition solution to a queueing network model of a distributed file system with dynamic locking," *IEEE Trans. Software Eng.*, vol. SE-12, no. 4, pp. 521-530, Apr. 1986.

[3] L. Kleinrock, *Queueing Systems, Vol. I*, New York: Wiley-Interscience, 1975.

[4] S.-C. Shyu and V. O. K. Li, "Performance analysis of static locking in distributed database systems," *IEEE Trans. Computers*, vol. 39, no. 6, pp. 741-751, June 1990.

[5] M. Singhal and A. K. Agrawala, "Performance analysis of an algorithm for concurrency control in replicated database systems," *Proc. ACM SIGMETRICS Conf. Measurement Modeling Comput. Syst.*, 1986, pp. 159-169.

[6] Y. C. Tay, R. Suri, and N. Goodman, "A mean value performance model for locking in databases: The no-waiting case," *J. ACM*, vol. 32, no. 3, pp. 618-651, July 1985.

A Note on the Performance Analysis of Static Locking in Distributed Database Systems

Yinghong Kuang and Ravi Mukkamala
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529.

Abstract

Even though transaction deadlocks can severely affect the performance of distributed database systems, many current evaluation techniques ignore this aspect. In [4], Shyu and Li proposed an evaluation method which takes deadlocks into consideration. However, their technique is limited to exclusive locking. In this paper, we extend their technique to allow for both shared and exclusive locking. Using this technique, we illustrate the impact of deadlocks, in the presence of shared locking, on distributed database performance.

Index Terms: Distributed databases, exclusive locking, performance modeling, shared locking, static locking, two-phase locking.

1 Introduction

A distributed database system (DDS) is a collection of cooperating nodes each containing a set of data objects. A user transaction can enter such a system at any of these nodes. The receiving node, often referred to as the *coordinating* node, undertakes the task of locating the nodes that contain the data objects required by a transaction.

In order to maintain database consistency and correctness in the presence of concurrent transactions, several concurrency control protocols have been proposed [1]. Of these, locking protocols have been widely used in both commercial and research environments. In static locking, prior to start of execution, a transaction needs to acquire either a shared-lock (for read operations) or an exclusive lock (for update operations) on each of the relevant data objects. When transactions with conflicting lock requests are initiated concurrently, they could be possibly blocked due to a deadlock. Deadlocks are known to deteriorate performance in both centralized and distributed database systems [4,6]. In spite of this, several performance studies have ignored the deadlock problem in their analyses [2,5].

In [4], Shyu and Li proposed an elegant technique to evaluate the response time and throughput of transactions in a non-replicated DDS. (In the rest of the paper, we refer to this as the S-L technique.) Assuming *exclusive* locking (i.e., only write operations), they model the queue of lock requests at an object as a M/M/1 queue [3]. This results in a closed-form for the waiting time distribution at a node, expressed in terms of the average rates of arrivals of requests and the average lock-holding time. This technique consists of two stages. In the first stage, the average transaction response time and throughput are calculated by ignoring the deadlock. This is an iterative step that uses the known properties of the M/M/1 queue [3]. In the second stage, the probabilities of transaction conflicts and deadlocks are computed. These probabilities are used, in turn, to compute the response time and throughput in the presence of deadlocks.

In general, a database transaction reads from a set of data objects (the read-set) and writes on to a set of data objects (the write-set). Assuming that all accesses are write-only (as in S-L) results in the worst-case performance (with respect to deadlocks and response time) of a DDS. In this paper, we propose to extend the S-L technique to consider both the the read and the write operations of database transactions. Using the extended S-L, we evaluate the effect of deadlocks on distributed database systems.

2 Model

Except for the inclusion of read operations, our model is the same as in S-L. For the sake of completeness, we summarize the DDS model here.

- There are N nodes and D data objects (or data granules in S-L) in a DDS. The D data objects are uniformly distributed across the N nodes. A data object may be located at exactly one node.
- Each transaction accesses K data objects. Among these, $r \cdot K$ are for read-only purpose, and the rest are for read-write. (Obviously, $0 \leq r \leq 1$.) In other words, a transaction must procure $r \cdot K$ shared locks and $(1-r) \cdot K$ exclusive locks.
- Each data object is equally likely to be accessed by a transaction.
- Transaction arrivals into the system is a Poisson process with rate λ.
- The communication delay between nodes is exponentially distributed with mean $\bar{t}$.
- The average execution time of a transaction, once the locks are obtained, is $\bar{S}$.

3 Evaluation Procedure

Since we are only proposing extensions to the S-L model, we do not intend to repeat the description of their procedure. Instead, we will discuss only the salient features of their procedure that are relevant to describe the proposed extensions.

In Stage 1 of the S-L technique, an iterative procedure is used to evaluate the response time and throughput of a DDS ignoring the possibility of deadlocks. In each iteration, the average waiting time (for exclusive lock requests) at each of the data objects is computed using estimates of the average lock-holding times from the previous iteration. By definition, no two exclusive lock requests can have lock grants on the same object simultaneously. Also, assuming that the lock-holding time is exponentially distributed (with mean $1/\mu$) and that the lock request arrivals form a Poisson process (with rate $\lambda_r = \lambda \cdot K/D$), the distribution of waiting time W_i at an object i is expressed as (M/M/1 queueing formula [3])

$$f_{W_i}(y) = (1-\rho) \cdot \mu_0(y) + \lambda_r(1-\rho) \cdot e^{-\mu(1-\rho)y} \quad (1)$$

where $\mu_0(\cdot)$ is the impulse function and $\rho = \lambda_r/\mu$. Using the waiting time distribution, the waiting times at the K data objects are randomly generated. These are used, in turn, to derive new estimates for the lock-holding times $(1/\mu)$. The iterations stop when two successive computations of average waiting time estimates are very close.

When we consider both shared and exclusive locks, the problem of estimating the waiting time distributions becomes difficult. Since two shared lock grants on the same object may exist simultaneously, and an exclusive lock may not be granted while another shared or exclusive lock is already granted, the queueing discipline at a node is complex. Such complex queueing disciplines are analytically intractable [3]. For this reason, we propose to use simulation to solve the queueing model. Given the total rate of arrival of lock requests λ_r, the shared lock ratio (r), and the average lock-holding time $(1/\mu)$, the queue at an object may be simulated. From here, the waiting time distribution may be obtained in the form of a table. Once the waiting time distribution is obtained, the same iterative procedure as in Stage 1 of S-L may be adopted to compute the response time when deadlocks are ignored. As in S-L, transaction response time is defined as the time between the instance the lock requests are sent and the time the last grant request is received by the coordinating node.

In Stage 2, the probabilities of transaction deadlock and restart are computed. These are then used to compute response time and throughput in the presence of deadlocks. When we assume that transactions only make exclusive lock requests, the expression for the probability of conflict between any two transactions is given by,

$$P_c = 1 - \frac{\binom{D-K}{K}}{\binom{D}{K}} \tag{2}$$

However, when we consider both shared locks and exclusive locks, the probability of conflict is reduced. In this case the probability of conflict is given by,

$$P_c' = 1 - \frac{\binom{D-K}{K}}{\binom{D}{K}} - \frac{\sum_{i=1}^{K'} \binom{K'}{i} \cdot \binom{D-K}{K'-i} \cdot \binom{D-K-K'+i}{K-K'}}{\binom{D}{K} \cdot \binom{K}{K'}} \tag{3}$$

where $K' = r \cdot K$ and represents the average number of shared locks; $(K - K')$ is the average number of exclusive locks per transaction. Clearly, when $r = 0$, $P_c = P_c'$; when $r = 1$, $P_c' = 0$; and in all cases, $P_c \geq P_c'$.

By replacing P_c with P_c', the procedure suggested in S-L may be applied to obtain the desired performance metrics.

4 Results

Using the extended S-L technique, we obtained a number of interesting results that illustrate the effect of deadlocks on database performance. These are summarized in Figures 1-5. We have verified our results with those obtained in [4] for the all exclusive locks case ($r = 0$). We make the following observations.

- As expected, the presence of shared locks has a substantial impact on the probability of deadlock occurrence (Fig. 1). When only 1/3 of the accessed data objects are updated (i.e., $r = 2/3$), the probability of deadlock is considerably small as compared to when all objects are updated ($r = 0$).

- The observations about the deadlock probabilities are also valid for restart probabilities (Fig. 2).

- Transaction response times are also quite sensitive to the ratio of shared locks (Fig. 3). Here, we compare the response times when deadlocks are ignored (computed in Stage 1) with those obtained when deadlocks are considered (computed in Stage 2). The effect of deadlocks is more predominant at higher transaction loads and with smaller values of r. When $r = 2/3$, the effect of deadlocks is not significant on response time.

- The effect of deadlocks on response time is decreased with the increase in the number of data items (Fig. 4). Obviously, this is due to the decrease in probability of conflicts and hence a decrease in deadlock occurrence. For $r = 2/3$, this effect is almost insignificant. For $r = 1/3$ and $r = 0$, deadlocks seems to have a noticeable effect on response time.

- Fig. 5 summarizes the effect of the number of locks per transaction on response time. When K is small, the probability of deadlock is negligible, and hence its effect on response time is small. At higher values of K, the effect of deadlocks on response times is significant. Similarly, at smaller values of r, the effect of dedalocks is more apparent.

5 Conclusion

In [4], Shyu and Li presented an elegant technique to evaluate the performance of distributed database systems in the presence of deadlocks. Their technique assumed only exclusive locks and thus representing the worst-case effects of deadlocks. In this paper, we have extended their technique to allow both shared and exclusive locking. Using the extended technique, we evaluated the the effect of number of data objects, the number of data objects accessed, and the ratio of read operations on transaction response time. These results also indicate the importance of considering both shared and exclusive lock requests for response time evaluations.

References

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[2] A. Hac, "A decomposition solution to a queueing network model of a distributed file system with dynamic locking," *IEEE Trans. Software Eng.*, vol. SE-12, no. 4, pp. 521-530, Apr. 1986.

[3] L. Kleinrock, *Queueing Systems, Vol. I*, New York: Wiley-Interscience, 1975.

[4] S.-C. Shyu and V. O. K. Li, "Performance analysis of static locking in distributed database systems," *IEEE Trans. Computers*, vol. 39, no. 6, pp. 741-751, June 1990.

[5] M. Singhal and A. K. Agrawala, "Performance analysis of an algorithm for concurrency control in replicated database systems," *Proc. ACM SIGMETRICS Conf. Measurement Modeling Comput. Syst.*, 1986, pp. 159-169.

[6] Y. C. Tay, R. Suri, and N. Goodman, "A mean value performance model for locking in databases: The no-waiting case," *J. ACM*, vol. 32, no. 3, pp. 618-651, July 1985.

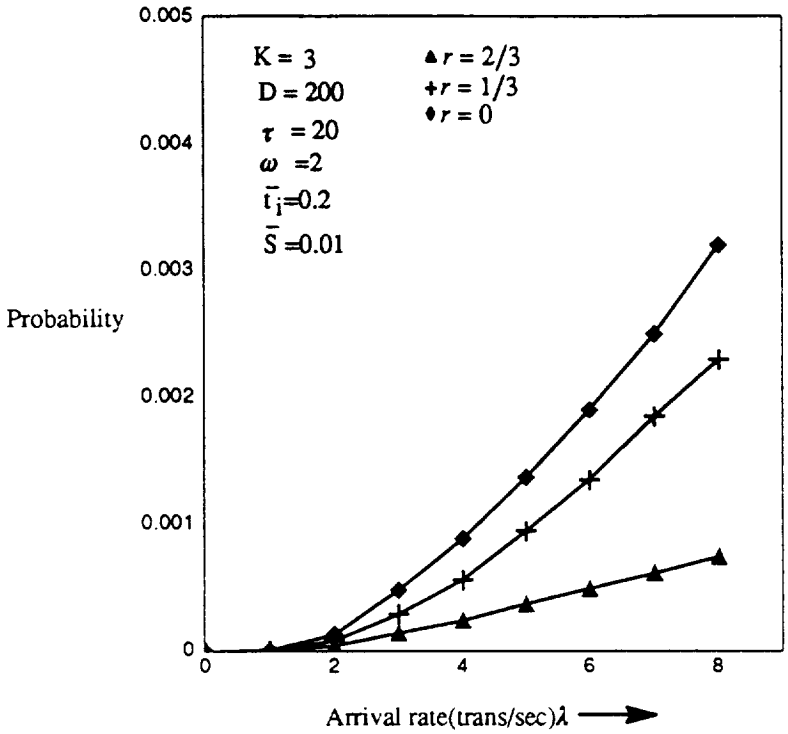


Fig.1. Deadlock probability with different read ratios

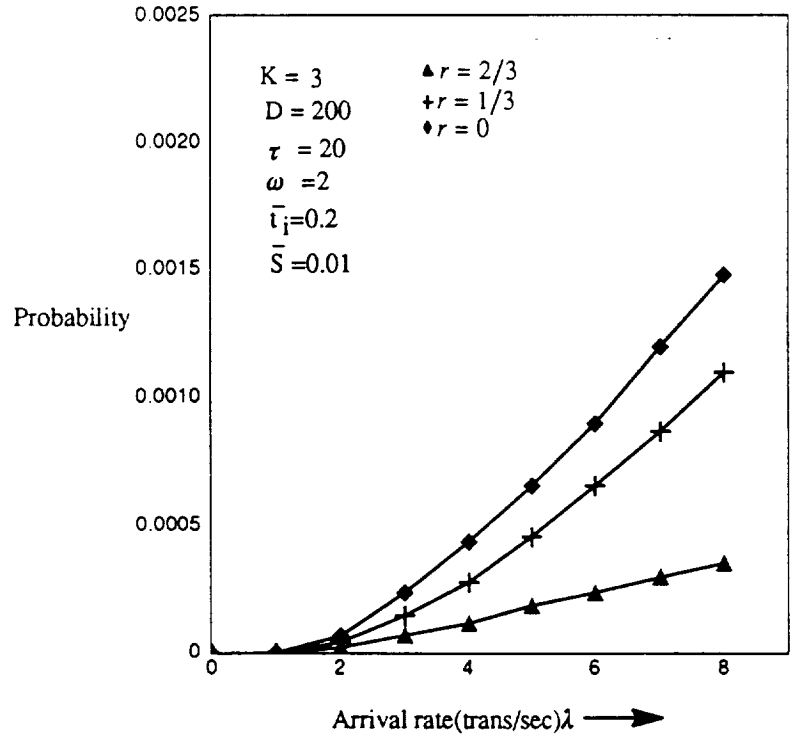


Fig.2. Restart probability with different read ratios

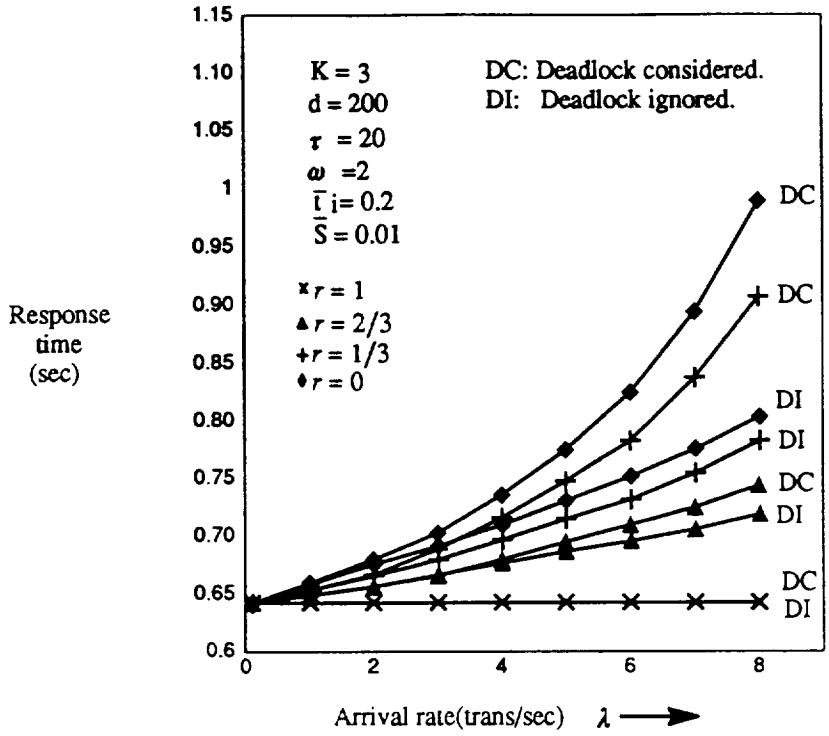


Fig.3 Comparison of response time when deadlock is considered and deadlock is ignored.

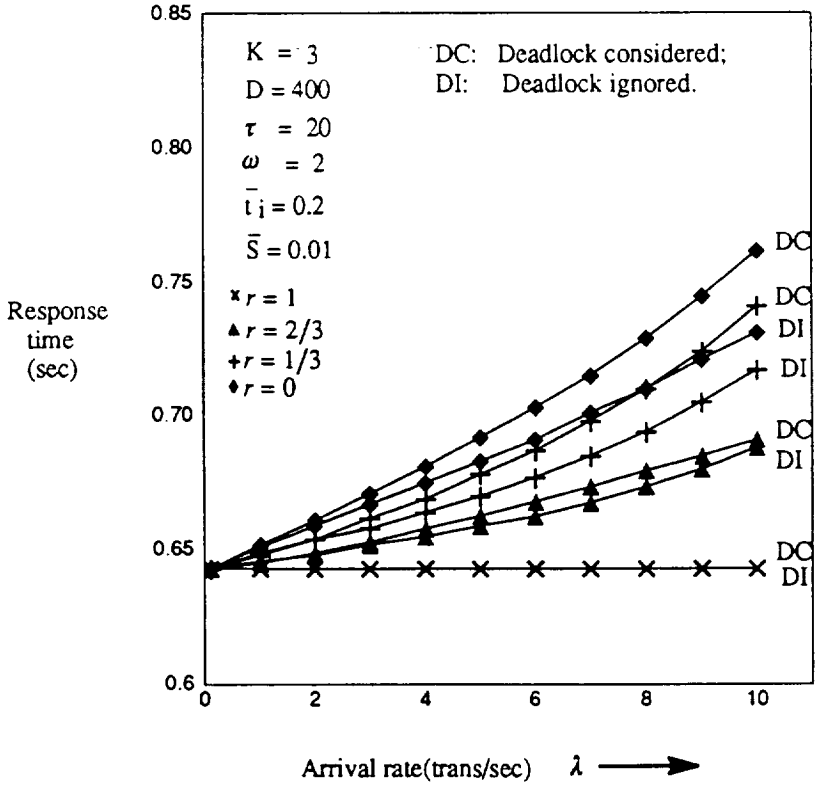


Fig. 4 Response time with high number of data objects.

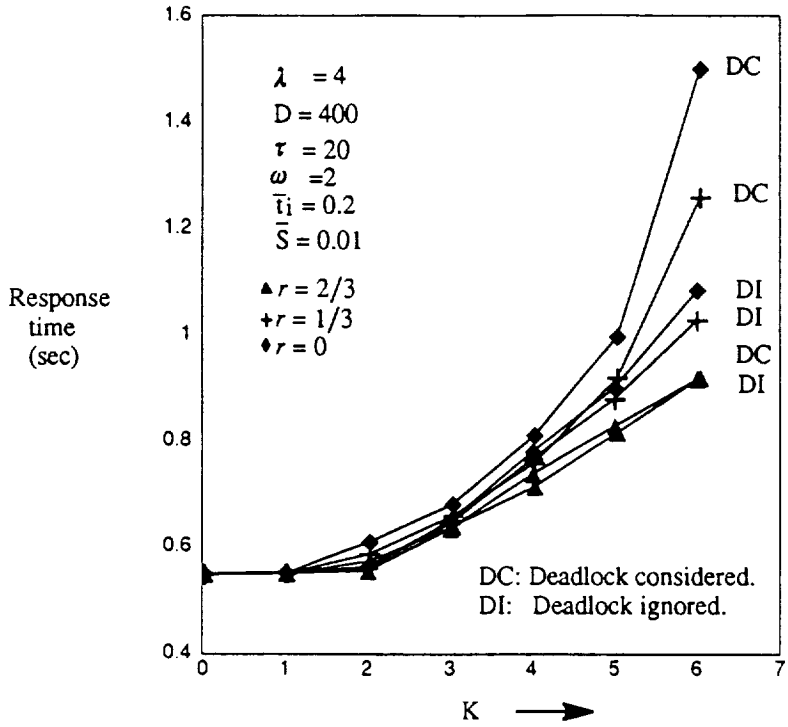


Fig.5. Comparison of response time with different number of lock requests.