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 nonempty. 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 transactions 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_1 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_1 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_1 due to case (i), we are only interested in transactions that update a data item in the write-set of T_1 and not commuting with T_1. The probability that a given data item (updated by T_1) is not updated in another partition k' by a non-commuting transaction (with respect to T_1) 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

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

The probability that transaction T_1 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 $R_1 = \sum_{k=1}^{p}(1-\mu_k)\alpha_k^r N_{2k}$

To compute the rollbacks of category (ii), we need to determine the probability that T_1 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_1 is not rolled back due to the roll back of any of its dependency parents is now given by:

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

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 $R_2 = \sum_{k=1}^{p}(1 - \chi_k)\alpha_k^r(N_{1k} + \mu_k N_{2k})$. 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 #	$p_1 = 4, p_2 = 6, c = 3$				$p_1 = 4, p_2 = p_3 = 3, c = 3$			
	Before Merge	R_1	R_2	After Merge	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 #	$p_1 = 4, p_2 = 6, c = 2$				$p_1 = 4, p_2 = 6, c = 8$			
	Before Merge	R_1	R_2	After Merge	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	5334	24134	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 #	$p_1 = 10, p_2 = 10, c = 5$				$p_1 = 10, p_2 = 10, c = 12$			
	Before Merge	R_1	R_2	After Merge	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 datab 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: $p_1 = 4, p_2 = 6, w/c : m = 0.27, wo/c : m = 0.0$)

$p_1 = 4, p_2 = 6, \check{c} = 3$					
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

```
/* This program creates a menu and facilitates updates, inserts and
deletes of records in a database.  EMPP is an employee database and
the program assumes it to be already created with the following
fields:
EMPNO (employee number) of type numeric
ENAME (employee name) of type character
SAL (salary) of type numeric with provision for 2 places after
decimal
DEPTNO ( department number) of type numeric
JOB (job name) of type character
*/

#include <stdio.h>
#include <ctype.h>


EXEC SQL BEGIN DECLARE SECTION;
VARCHAR uid[80]; /* variable for user id */
VARCHAR pwd[20]; /* variable for password */

int  empno; /* host variable for primary key - employee number */
VARCHAR ename[15]; /* host variable for employee name */

int  deptno; /* department number */

VARCHAR job[15]; /* host variable for job */

int  sal; /* host variable for salary */

int l = 0; /* host variable to hold the length of the srting - a
value returned by the asks() function. */

int count; /* a variable to obtain number of records in the
database with the same primary key value */

int reply = 0; /* variable to obtain the whether a new value exists
*/

int choice = 0; /* variable defined to obtain value for the menu */


int code; /* variable to print to the ascii file to indicate wether
the record was updated (value=1), inserted (value=2) and deleted
(value=3) */

EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLCA;

FILE *fp;

main()

{
/* open ascii file in append mode */
```

```
   fp = fopen("outfile", "a");
/* give the login and password to logon to the database */
   strcpy(uid.arr,"rsp");
   uid.len = strlen(uid.arr);
   strcpy(pwd.arr,"prs");
   pwd.len = strlen(pwd.arr);

/* exit in case of an unauthorized accessor to the database */

   EXEC SQL WHENEVER SQLERROR GOTO errexit;
   EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;
for (;;)
{ /* infinite loop begins */
 /* menu for selecting update, insert, and delete options */
     printf("\n \n 1.  Update a record \n");
     printf("\n \n 2.  Insert a record \n");
     printf("\n \n 3.  Delete a record \n");

     printf("\n \n Select an option 1/2/3 ? \n");

     choice = getche();
     if (choice == '1') goto update;
          else if (choice == '2') goto insert;
                    else if (choice =='3') goto delete;
                         else { printf("invalid selection");
                            exit(1);}
update: /* label for the update option */

  {
 /* To ensure that the employee with the given employee number
exists, before update could be made. */
     code = 1;
     printf(" \n count is %d \n", count);
     askn("Enter employee number to be updated: ", &empno);

/* using the COUNT supported by oracle, the number of records
having the desired employee number is assigned the variable count
*/

     EXEC SQL SELECT COUNT(EMPNO) INTO :count
     FROM EMPP
     WHERE EMPNO = :empno;
     printf("count is %d \n", count);

     if (count == 0)
     { printf("Employee with employee number %d does not exist \n",
empno);
       exit(1); }

/* retrieve the information from the database whose employee-number
has been requested for, and place the contents of the fields into
C variables for update purposes.  */

     EXEC SQL SELECT ENAME, SAL, DEPTNO, JOB
     INTO :ename, :sal, :deptno, :job
     FROM EMPP
     WHERE EMPNO = :empno;

/* displays the already existing value for employee name */
/* assign the new employee name if it should be updated */

     printf("ename is %s \n", ename.arr);
     printf("Do you want to update ENAME:(y/n)?");
     reply = getche();

     if (reply == 'n'){
        ename = ename;
        printf("\n ename is %s \n", ename.arr);}
     if (reply == 'y') {
        l = asks("\n enter employee name : ", ename.arr);
        printf("new ename is %s \n", ename.arr);}

/* displays the already existing value for job name */
/* assign the new job if it should be updated */

     printf("do you want to update job-name:(y/n)?");
     reply = getche();

     if (reply == 'n'){
        job = job;
        printf("\n job-name is %s \n", job.arr);}
     if (reply == 'y'){
        job.len = asks("\n enter employee's job :", job.arr);
        printf("new job-name is %s \n", job.arr);}

/* displays the already existing value for salary */
/* assign new salary if it should be updated */

   printf("Do you want to update salary:(y/n)" );
   reply = getche();

     if (reply == 'n'){
        sal = sal;
        printf("\n salary is %d \n", sal);}
     if (reply == 'y'){
        askn("\n enter employee's salary: ", &sal);
        printf("new salary is %d \n", sal);}

/* displays the already existing value for department number */
/* assign the new department number if it should be updated */

   printf("Do you want to update deptno :(y/n)");
   reply = getche();

     if (reply == 'n'){
        deptno = deptno;
        printf("\n deptno is %d \n", deptno);}
     if (reply == 'y'){
        askn("\n Enter employee dept  :   ",&deptno);
        printf("new deptno is %d \n", deptno);}

/* update the database with the new values */

     EXEC SQL UPDATE EMPP
     SET ENAME = :ename, SAL = :sal, DEPTNO = :deptno, JOB = :job
     WHERE EMPNO = :empno;

     printf("\n %s with employee number %d has been updated\n",
ename.arr,empno);

        fprintf(fp,"%10d %1d %15s", empno, code, ename.arr);
        fprintf(fp,"%6d %3d %4s\n",  sal, deptno, job.arr);

        printf("%10d %15s %6d", empno, ename.arr, sal);
        printf("%3d %4s\n", deptno, job.arr);
  }

insert: /* label for insertion of record based on the employee
number */

  {
     code = 2;

/* To prevent insertion of a record whose primary key is the same
as the primary key of an already existing record */

     askn("\n Enter employee number to be inserted:", &empno);
     EXEC SQL SELECT COUNT(EMPNO) INTO :count
     FROM EMPP
     WHERE EMPNO = :empno;

     printf("count is %d \n", count);

     if (count > 0){
       printf("Employee with %d employee number already exists \n",
empno);
       exit(1);}
     else { /* obtain values for various fields to be inserted */
          l = asks("Enter employee name : ", ename.arr);
          job.len = asks("Enter employee job :", job.arr);
          askn("Enter employee salary :", &sal);
          askn("Enter employee dept number :", deptno);

/* insert the values obtained into the database */

          EXEC SQL INSERT INTO EMPP(EMPNO,ENAME,JOB,SAL,DEPTNO)
          VALUES (:empno, :ename, :job, :sal, :deptno);

/* append the insert into the ascii file */

          fprintf(fp,"%10d %1d %15s ", empno, code, ename.arr);
          fprintf(fp,"%6d %3d %4s \n", sal, deptno, job.arr);
          printf("%10d %15s %6d", empno, ename.arr, sal);
          printf("%3d %4s", deptno,job.arr); }
  }

delete:  /* label for deletion of records based on employee number
*/

  {
  int code = 3;

/* obtain the employee number of the employee to be deleted */

  askn("Enter employee number to be deleted :", &empno);
  EXEC SQL SELECT COUNT(EMPNO) INTO :count
  FROM EMPP
  WHERE EMPNO = :empno;

  if (count > 0){ /* delete record if it exists */
     EXEC SQL DELETE FROM EMPP WHERE EMPNO = :empno;
     printf("Employee number %d deleted \n",empno);

     fprintf(fp, "%10d %1d\n", empno, code);}

  else {
     printf("Employee with number %d does not exist \n", empno);
     exit(1);}
  }

   EXEC SQL COMMIT WORK RELEASE; /* make the changes permanent */
   printf ("\n End of the C/ORACLE example program.\n");
   return;
     fclose(fp);

     EXEC SQL WHENEVER SQLERROR CONTINUE;
     EXEC SQL ROLLBACK WORK RELEASE; /* in case of inconsistency */

     return;

errexit:
   errrpt();

}
} /* infinite loop ends */

/* function takes the text to be printed and accepts a srting
variable from standard input and converts it into numeric - hence
is used to obtain values for numeric fields */

int askn(text,variable)
     char text[];
     int *variable;
     {
     char s[20];
     printf(text);
     fflush(stdout);
     if (gets(s) == (char *)0)
          return(EOF);

     *variable = atoi(s);
     return(1);
     }
/* function takes the text to be printed and prints it, accepts
string values for character variables and is thus used to obtain
values for fields of type character.  It returns the length of the
string value */

int asks(text,variable)
     char text[],variable[];
     {
     printf(text);
     fflush(stdout);
     return   (   gets(variable)   ==   (char   *)   0   ?   EOF   :
strlen(variable));
     }

errrpt()
     {
     printf("%.70s   (%d)\n",   sqlca.sqlerrm.sqlerrmc,
-sqlca.sqlcode);
     return(0);
     }
```