# Performance of Time Warp Protocols for Transaction Management in Object-Oriented Systems

John A. Miller

Department of Computer Science
415 Graduate Studies
University of Georgia
Athens, Georgia 30602

Nancy D. Griffeth

Bell Communications Research (Bellcore)
435 South Street
Morristown, New Jersey 07960

**Abstract**

It has been pointed out (e.g., by Richard Fujimoto) that protocols developed for the difficult problem of parallel and distributed simulation can also be adapted to other distributed applications. Indeed, in the mid 1980's David Jefferson adapted his Time Warp protocol to the problem of database concurrency control. In the present work, we consider our own variants of the Time Warp for database concurrency control, and perform a simulation study to compare their performance with other more traditional protocols. The performance results obtained for the Time Warp protocol are very encouraging. The initial motivation for developing and analyzing these variants of Time Warp was to find a high performance protocol suitable for a Query Driven Simulation system. Such a system will need to enforce the consistency of concurrent/parallel operations from both a database viewpoint and a simulation viewpoint.

**Keywords:** Database Protocols, Transaction Management, Concurrency Control, Recovery, Parallel/Distributed Simulation Protocols, Object-Oriented Databases.

## 1. Introduction

The initial motivation for this work was to select protocols suitable for concurrency control in a Query Driven Simulation system [1-7]. Query Driven Simulation is an approach to simulation modeling and analysis that uses database technology to automate the storage and retrieval of simulation data and simulation models. The fundamental tenet of Query Driven Simulation is that simulationists or even naive users should see a simulation system as a sophisticated information system. This system should be able to store or (just as easily) generate information about the behavior of systems that users are studying. Some of the application areas to which we are applying Query Driven Simulation include model management [3], decision support systems [9], and most recently genome mapping [10]. Since *object-oriented database systems* have such powerful data and behavioral structuring mechanisms, they provide an ideal foundation to sup-

This research was partially supported by a University of Georgia Faculty Research Grant.

port demanding applications like Query Driven Simulation†. The interactive nature of Query Driven Simulation requires the rapid execution of simulation models. Ideally, such simulations should be run on multiprocessors (either tightly-coupled or loosely-coupled), since they can provide significant improvements in performance [11, 12]. Reinforcing this idea is the fact that Query Driven Simulation can spawn several largely independent model executions which can take advantage of the availability of processors [2]. On such a system, the concurrency control protocols used by the object-oriented database will have a major impact on performance [13-15]. They will effect the performance of both ordinary database operations (queries and updates) and the execution of simulation models.

The use of common concurrency control protocols such as Two-Phase Locking may lead to an unacceptable performance bottleneck [16], particularly for long-duration transactions. Although suitable for today's machines and applications, as machines become more parallel and applications become more complex, alternatives should be sought. In particular, for Query Driven Simulation it is convenient that there are protocols that apply to both problems, since the approach used for the concurrency control protocol needs to be compatible with the approach used for distributed event synchronization. If one protocol blocks while the other attempts to proceed, we will negate the advantages of both protocols.

One way to achieve acceptable performance is to avoid protocols that block processors, whether they are protocols for synchronization of simulation events or protocols for concurrency control. This claim is based on the intuition that it is better to use available processors, even if the work done using those processors turns out to be incorrect and must be thrown away, than to block processing when processors are available. This argument assumes that the useful work done by transactions in the forward direction outweighs the non-productive work that has to be reversed. The simulation results presented in this paper and in [15] indicate that this assumption is justified. A second way to improve performance is to have multiple versions of objects [17,

---

† The foundation of our Query Driven Simulation system is Active KDL (Knowledge/Data Language) [2, 3, 5, 6, 7]. Active KDL is a functional object-oriented database system supporting Query Driven Simulation by providing access to integrated data, knowledge, and model bases. The three sublanguages of Active KDL provide support for simulation modeling and analysis: Complex simulations may be structured using the schema definition language (SDL); results may be retrieved using the query language (QL); and models may be implemented using the database programming language (DBPL).

18] (rather than updating an object new versions of the object are created). This reduces the kinds of conflicts that can occur (e.g., write-write conflicts are no longer a problem). Many object-oriented database systems provide versioning capabilities [16] for application reasons [19].

Richard Fujimoto has pointed out in [11, 12] that the protocols developed for the difficult problem of parallel and distributed simulation can be adapted to other parallel and distributed applications. One such application is database transaction management, as there are considerable similarities between protocols for database concurrency control and protocols for synchronization of simulation events. Indeed, in the mid 1980's David Jefferson [20, 21] had adapted his *Time Warp* protocol to serve as a concurrency control protocol for database transactions. As Time Warp is one of the more successful optimistic protocols used for parallel and distributed simulation, its adaptation to high performance database applications should be investigated, particularly for databases integrating simulation capabilities (e.g., Query Driven Simulation). A similar database concurrency control protocol, which has been found to have good performance [15] in highly concurrent systems, is the *Multiversion Timestamp Ordering* protocol [22]. Protocols such as Multiversion Timestamp Ordering and Time Warp have the following advantages: (1) high effective concurrency levels, (2) deadlock free operation (because there is no (or limited) blocking), and (3) efficient operation unless there is a conflict. These types of protocols resolve conflicts by partially rolling back (undoing some operations) or completely aborting transactions. The fact that transactions are not blocked leads to higher effective concurrency levels. Thus, if aborts or partial rollbacks do not occur with great frequency, these protocols should exhibit excellent performance.

In this paper, we extend Jefferson's work on the Time Warp protocol, developing several variants of the Time Warp protocol and analyzing the performance of some of the more promising variants using simulation models. (Some preliminary work on this was done in [4], while complete specifications of the protocols and proofs of correctness are given in [23, 24].) The performance of the protocols is also compared with that of other, more traditional database pro-

tocols. We model transactions as Markov chains [13-15], where each state represents a database operation such as reading or writing. (Our modeling approach was inspired by the operational analysis techniques used by Y.C. Tay [25-27].) We use the models to study the effect that database protocols have on performance measures such as throughput. In [14], Markov models were applied to four database protocols for ensuring that recovery to a consistent state is always possible. The studies in this paper apply the same method to database protocols enforcing the stronger condition of serializability. The analysis of performance is carried out by elements of analytic modeling (Markov models) and simulation modeling (SIMODULA† programs). The simulation models, although based on the analytic models, allow more details to be captured.

The central result of the paper is a performance comparison of the Multiversion Timestamp Ordering and Time Warp protocols. The Realistic Recovery protocol is also analyzed. Since it only deals with recovery, one would expect much higher performance. (In this study Realistic Recovery is incorporated into Multiversion Timestamp Ordering, and can be incorporated with Time Warp, see [23, 28].) This paper also introduces the idea of a Hybrid protocol that combines elements from both Multiversion Timestamp Ordering and Time Warp (see [29] for performance results).

## 2. Transactions and Protocols

Active objects provide a nice abstraction [31] for implementing [7] and/or simulating [6] parallel and distributed systems/applications. Following the process-interaction simulation world view, it is natural to model entities in a simulation as processes (or active objects). For example, in the ubiquitous bank simulation [6], each customer in the bank is modeled as an active object, which enters the bank, requests service from a teller, and eventually leaves the bank. For the sake of consistency, the execution of active objects must be controlled. An active object will periodically perform a transaction and must be ready to handle requests from other

---

† SIMODULA is a simple yet powerful process-oriented simulation system written in Modula-2 [1, 8, 30].

active objects. A *transaction* is a (typically small) group of related operations, that together perform some useful task (e.g., withdrawing money from a checking account).

To achieve acceptable performance transactions must be executed concurrently. Left uncontrolled, this concurrent behavior leads to inconsistency. Therefore *protocols* must be used to ensure that transactions do not violate consistency constraints. A strong consistency condition for the execution of a set of concurrent transactions is serializability. An execution of a set of concurrent transactions is said to be serializable, if their effects are equivalent to some serial execution. Protocols that enforce serializability are called *concurrency control protocols* [32-36]. Closely related are the *recovery protocols* that ensure that a database can recover to a consistent state after non-catastrophic failures and aborts of transactions [34, 37, 38, 39]. Also, for parallel and distributed simulation, the timing of simulation events must be controlled so that causal events are executed before their resultant events [12, 20, 40, 41].

The two primary correctness conditions to maintain in a database are serializability and recoverability. Of the two, serializability is the more restrictive, and is thus the bigger hindrance to high performance. Often a recovery protocol is adapted and incorporated into a protocol that enforces serializability. *Serializability* maintains the consistency of data, by guaranteeing that if the individual transactions are correct then their combined effect will be correct [42, 43]. Serializability is maintained by constraining the ordering of operations so that their effect is equivalent to some serial (i.e., one at a time) execution of the transactions. We also need to guarantee that the effects of successfully completed work are permanent. To this end, we provide a *commit* operation for a transaction to signal its successful completion and require that after it has committed, its effects will be part of the database forever. *Recoverability* guarantees that after a failure (either transaction failure or system crash) the database can be recovered to a consistent state, at which point things can carry on as usual [39]. Recoverability is maintained by not allowing a transaction to commit until all the data it has read is committed. After a failure, we recover the database to a state including the effects of exactly those transactions that committed before the failure.

In the following descriptions of the protocols, we assume that the only operations transactions invoke are *read* and *write*. Object-oriented database systems can also provide operations such as increment, which may facilitate better performance [44, 45].

## 3. A Recovery Protocol

The Realistic Recovery protocol described in [39] is a good choice for a multiversion database. Also, because the Realistic Recovery protocol is simple to analyze, we are able to validate the simulation results with analytic results. In addition, it provides an ideal vehicle for explaining the analytic models. (For the two concurrency control protocols only simulation results were produced.) To describe the Realistic Recovery protocol, let us consider how recovery protocols must operate.

Data that has been written by transactions that have yet to commit is termed *dirty data*. From the definition of recoverability, we can see immediately that recoverability must be enforced by not allowing commits of transactions that have read dirty data. To do this, we can block either reads of dirty data or we can block commits of transactions that have read dirty data. In either case we have to block; from simulation results [13-15], the better choice appears to be to block reading of dirty data.

The Realistic Recovery protocol requires that each object maintain a list of operations that have been requested (some of which may have also been executed) and that each transaction maintain a list of objects at which it has requested write operations. When a read or write operation is requested of an object, the operation is added to the object's list of operations. When a transaction requests a commit or abort, it notifies each object at which it has requested a write. The object may then remove some writes from its list of operations or dispatch some reads. Committed writes must be left on the operation list until they are immediately followed by another committed write. Reads are removed once they have been dispatched. The following summarizes how each operation is handled.

WRITE: Add the write operation (including the value to be written) to the end of the

operation list. Mark the write operation as uncommitted.

READ: Dispatch the read if it is immediately preceded by a committed write operation. Otherwise, add the read to the end of the operation list.

COMMIT: Notify the objects to mark all write operations of the transaction as committed. If any of these writes is immediately preceded by a committed write, the earlier committed write is removed from the operation list. If any of these writes is immediately followed by one or more read operations, the reads are dispatched.

ABORT: Notify the objects to remove all write operations of the transaction. If after removal, any committed write is immediately preceded by another committed write, the earlier one may be removed. If any committed write is immediately followed by one or more reads, the reads may be dispatched.

## 4. Multiversion Timestamp Ordering Protocol

Under the Multiversion Timestamp Ordering protocol, each transaction is timestamped upon initiation. In the simulation this is simply done by calling the Time () function. In a parallel or distributed database, the generation of good unique timestamps is a bit more challenging (see [35] for techniques to do so). Each version of an object is also timestamped with the timestamp of the transaction writing it. Additionally, it is important to keep track of the latest reader of each version. Hence, the following information needs to be maintained for the successful operation of the protocol,

$ts(T_i)$ = timestamp of transaction $T_i$
$ws(v_k)$ = write-stamp of version $v_k$
$rs(v_k)$ = read-stamp of version $v_k$

where $T_i$ is the $i^{th}$ transaction to initiate ($ts(T_i) < ts(T_j)$ for $i < j$) and $v_k$ is the $k^{th}$ version of some object $O$.

The Multiversion Timestamp Ordering protocol supports high concurrency since reads can be executed in a relatively unrestricted way. Except for the concern for recoverability, there are no restrictions at all (simply the appropriate version is chosen and read). Writes, however, may

cause a transaction to be aborted. In particular, if transaction $T_i$ attempts to write a version of object $O_j$ that interferes with an existing information transfer, then this write cannot be allowed to proceed. This occurs under the following conditions

$$ws(v_k) < ts(T_i) < rs(v_k)$$

where $v_k$ is the version that immediately precedes the version $T_i$ is attempting to write. In other words, $T_i$ is attempting to insert a new version immediately after $v_k$, but since the read-stamp on $v_k$ is greater than the timestamp of $T_i$ this cannot be allowed. The problem is that a transaction reading version $v_k$ should have really read the version that transaction $T_i$ is attempting to write. For example, consider the following schedule,

$$SCH1\dagger: \quad W_2(O_{10})\,R_4(O_{10})\,W_5(O_{20})\,R_7(O_{20})\,R_9(O_{20})$$

where $R_i(O)/W_i(O)$ denotes a read/write of object $O$ by transaction $i$. If transaction $T_8$ attempts to write a version of $O_{20}$, it must be aborted since transaction $T_9$ has already read the version written by transaction $T_5$.

Since writes may be aborted, some type of recovery protocol is necessary to maintain consistency. One that appears particularly well suited is the Realistic Recovery protocol. As mentioned before, it will block reads until the versions they are attempting to read are committed. This will introduce some extra delay into the system, but the amount of blocking will be rather small in comparison to the Pessimistic Recovery protocol or the commonly used Two-Phase Locking concurrency control protocol [15]. Furthermore, so long as the transactions are two-stage (see Section 6) no deadlocks can occur. (Another possibility is to use the Optimistic Recovery protocol.)

---

† Schedules for parallel and distributed databases represent a partial ordering of the operations; for simplicity, $SCH1$ is a total ordering.

## 5. Time Warp Protocol

The Time Warp protocol has been studied extensively for parallel and distributed simulation. It allows active objects (processes) to advance their state forward as rapidly as possible, until a message (from some other active object) is received whose receive time is in the past. When this happens, the active object must be rolled back to this time so that the event specified in the message can be executed. This same Time Warp protocol can be adapted for use as a database concurrency control protocol [20, 21].

In schedule $SCH1$, transaction $T_7$ also reads the version written by transaction $T_5$. Unlike the read by transaction $T_9$, this presents no problem for transaction $T_8$ in its write attempt. To maintain serializability, it suffices to have transaction $T_9$ change its read from transaction $T_5$'s version to transaction $T_8$'s version. Hence, we need to be able to partially roll back a transaction. The simplest approach is to back up transaction $T_9$ to the point of the read of $O_{20}$ (i.e., the *read-in-question*) and then begin redoing the transaction. A more detailed analysis shows that the following would suffice:

1. Redo the read-in-question. In addition, perform any reads that were previously skipped because a condition that uses the value of the read-in-question evaluated to false. We say such reads are conditionally dependent on the read-in-question.
2. Undo and then redo writes that are either conditionally or value dependent on the read-in-question. A write is value dependent, if the value obtained by the read-in-question is used in the write's calculation.

Note that in Time Warp, since transactions are only partially rolled back, their internal structure (including if statements) becomes important. Choosing to partially roll back rather than abort as is done with the Multiversion Timestamp Ordering protocol, has appeal since aborting is a more drastic operation. However, when the writes of a transaction are undone, they may cause other transactions to be partially rolled back leading to cascaded rollbacks. This would suggest that as the rate of conflict gets high, the performance of this protocol would degrade rapidly.

### 5.1. Anti-Transaction Variant

The thorniest problem in using the Time Warp protocol is that some of the transactions that need to be partially rolled back cannot, since they have already committed. The first remedy assumes that transactions can be reversed through the use of anti-transactions. An anti-transaction is a system spawned transaction that reverses or counterbalances the effects of some transaction.

### 5.2. Deferred Commitment Variant

In some applications, the use of anti-transactions is infeasible. For example, if money is dispensed from an automated teller machine, reclamation of this money is not possible. The deferred commitment variant of the Time Warp protocol avoids the necessity of anti-transactions by delaying the commitment of transactions. Using this variant, transactions are committed in timestamp order. (Actually, commitment occurs when the system's estimate of Global Virtual Time (GVT) exceeds the timestamp of the transaction [21]. At this time, external outputs may be physically performed and older versions may be purged.) Although this may introduce a significant amount of delay, it should not dramatically degrade throughput [21]. The main effect would be to increase the transaction sojourn time. However, it is possible that a few very long duration transactions could increase the delay encountered by average transactions to an unacceptable level.

### 5.3. Jefferson's Original Protocol

The adaptation of the Time Warp algorithm, as originally done by Jefferson [20, 21] is similar to the deferred commitment variant discussed above. However, it requires that all messages be processed in timestamp order. Thus the relative order in which reads are processed is significant. This requirement is above and beyond what is necessary to enforce serializability, so one would expect a tradeoff of possibly higher consistency for lesser performance.

### 5.4. Hybrid Protocol

A final possibility is to use a Hybrid protocol. The Hybrid would use Time Warp so long as all of the reads-in-question belong to uncommitted transactions, and use Multiversion Timestamp Ordering otherwise. Thus, when a write of a new version is attempted by $T_i$ three possible actions may ensue: (1) continue normally -- the read-stamp is smaller than $ts(T_i)$ so there is no conflict, (2) partially roll back the transactions with questionable reads -- there is a conflict and all of the reads-in-question belong to uncommitted transactions, or (3) abort transaction $T_i$ -- there is a conflict and at least one of the reads-in-question belongs to a committed transaction. (See [23, 28, 29] for more details about this protocol.)

## 6. Modeling Techniques

In this paper, we consider only *two-stage* transactions which do all their reading before writing. In [15], non two-stage transactions with an intermix of reads and writes are considered. Two-stage transactions are very good from a performance standpoint, *deadlock recovery* is greatly simplified (indeed deadlocks cannot occur for several of the protocols†), and the throughput for two-stage transactions is significantly better than the throughput for non two-stage transactions [15]. Furthermore, any transaction can be turned into a two-stage transaction by a simple change to the execution of the write operations: The data to be written is held in a local work area until the transaction is complete, then all writes are output. Note that this requires that we also postpone scheduling activities (e.g., setting locks, timestamps, etc.) associated with writes.

### 6.1. A Model of Transaction Behavior

The concurrent behavior of multiversion object-oriented databases can be modeled as fol-

---

† For the Realistic Recovery protocol, only reads can be blocked; they must wait for writes to be committed; since transactions are two-stage, once into their write stage they cannot be blocked; therefore it is impossible for a circular wait condition to develop.

lows: The database consists of a set of $D$ active objects. At any time, $M$ of these active objects are requesting operations to be performed by other active objects. This is accomplished by sending messages between active objects. For the sake of consistency, a sequence of interrelated operations are grouped together to form a transaction. These $M$ active objects are said to be transacting. The other $D - M$ active objects sleep until the next operation request message comes in.

In the models, transacting active objects perform a sequence of transactions. Furthermore, in this modeling study only read and write operations are considered. (Note, some operations such as increment allow very general interleavings and therefore contribute to high concurrency [45].) A write operation will simply write a new version of an object. The different versions are distinguished via the unique timestamps of the writing transactions ($ts(T_i)$). Formally, a write by transaction $T_i$ to object $O_j$ has the following effect:

$$v_k = W_i(O_j)$$
$$O_j = O_j \cup \{ v_k \}$$

The write $W_i(O_j)$ produces a new version $v_k$ with write-stamp $ws(v_k) = ts(T_i)$. This version is inserted into the ordered list $O_j$ according to its write-stamp. Transactions will read the most recent version of an object. Specifically, the most recent from its time frame; versions from the future may not be read. Consequently, when transaction $T_i$ reads from object $O_j$, the version with the largest write-stamp not exceeding the timestamp of $T_i$ is returned.

$$R_i(O_j) = \max_{ws} \{ v_k \mid v_k \in O_j \text{ and } ws(v_k) \leq ts(T_i) \}$$

### 6.1.1. Analytic Markov Models

Each transaction goes through a sequence of $J$ stages each corresponding to an operation (i.e., after completing an operation a transaction goes to the next stage to perform its next operation). From an analytic modeling point of view, we can model each of the stages as a node in a queueing network [46-48]. Therefore, the system will be composed of $M$ concurrent transacting active objects traversing $J$ nodes of the queueing network, reading and writing versions of $D$

active objects. In [15], we developed the queueing network models in detail. After applying a mean substitution approximation, we were able to derive simple continuous-time Markov chain models where $X(t)$ represents the state of a transaction at time $t$. (In the derivation we showed that any service time distribution will work; it does not have to be exponential.)

We are interested in the steady-state behavior (equilibrium distribution -- where $X(t) \rightarrow X$ in distribution) of such chains which may be determined from the following set of linear equations [49, 50]

$$\pi_j \sum_{k \neq j} q_{jk} = \sum_{k \neq j} \pi_k q_{kj} \qquad j = 0 \cdots J-1$$

$$\sum_j \pi_j = 1$$

where $q_{jk}$ is the transition rate from state $j$ to state $k$, and $\pi_j = P\{X = j\}$ is the equilibrium probability of a transaction being in state $j$. The left hand side of the first equation represents the flux (rate of flow) out of state $j$, while the right hand side represents the flux into state $j$. Hence, it is referred to as the *balance equation*, while the second equation is referred as the normalization equation. The system can then be viewed as $M$ parallel Markov chains (the mean substitution approximation takes care of the interaction). Hence the number of transactions at node $j$ is simply $N_j = \pi_j M$.

### 6.1.2. SIMODULA Simulation Models

From a simulation modeling point of view, active objects in the database can be modeled as SIMODULA [1, 8, 30] processes. In the script procedure used for these processes, a transacting active object repeatedly performs transactions until the simulation is over. Within a transaction, $k$ (= Objects) read/write operations are performed followed by a commit. This usual behavior is modified if the transaction is aborted or partially rolled back. (A PreCommit operation is added for the Time Warp protocol to allow a transaction to be reversed up until the last moment of its execution.) An abbreviated version of the ActiveObject script for the Time Warp protocol is given in Figure 1. The script for the other protocols are very similar. The main differences

occur in the logic of the database operations (Read, Write, PreCommit, Commit, Rollback, and Abort). Each of these operations advance simulated time by calling the SIMODULA WORK procedure. A typical call looks like the following:

```
WORK (Uniform (0.0, Tmax, Stream))
```

This will delay the current process for a random amount of time Uniformly distributed between 0.0 and Tmax. The Tmax parameter for PreCommits, Commits, Rollbacks, and Aborts is a few times greater than it is for Reads and Writes (see Section 8 for numerical values).

```
PROCEDURE ActiveObject;           (* Turned into a process by CREATE *)
VAR
   arrivalTime, timeStamp, delay: LONGREAL;
   oid: CARDINAL; aborted: BOOLEAN;
   attr: ObjectAttributes;
BEGIN
   Attributes (CurrentProcess (), attr);
   WITH attr^ DO
      IF Transacting THEN
         WritePhase := TRUNC (ReadProb * FLOAT (Objects)) + 1;

         (* Perform sequence of transactions *)
         WHILE Completions <= NumStop DO
            INC (Transaction);
            arrivalTime := Time ();
            aborted := TRUE;

            WHILE aborted DO
               aborted   := FALSE;
               timeStamp := Time ();

               (* Start/restart transaction *)
               WORK (0.1);              (* Start time delay *)
               OpNum := 1;
               LOOP
                  IF aborted THEN EXIT; END;
                  INC (Requests);
                  oid := Randi (1, NumObjects, Stream);
                  IF OpNum < WritePhase THEN      (* Select Read/Write *)
                     aborted := Read (oid, timeStamp);
                  ELSE
                     aborted := Write (oid, timeStamp);
                  END (* IF *);
                  HandleMessages;
                  WORK (1.0);
                  IF OpNum = Objects THEN PreCommit; END;
                  INC (OpNum);
                  IF OpNum = Objects + 1 THEN EXIT; END;
               END (* LOOP *);
               IF ~ aborted THEN Commit; END;
            END (* WHILE *);

            delay := Time() - arrivalTime;
            Tally (0, delay);
            INC (Completions);
         END (* WHILE *);
         DEC (NumExe);
      END (* IF *);
   END (* WITH *);

   (* Handle messages until simulation is over *)
   WHILE NumExe > 0 DO
      HandleMessages;
      IF NumExe > 0 THEN SUSPEND;
      ELSE WORK (1.0); END;
   END (* WHILE *);
   DESTROY;
END ActiveObject;
```

**Figure 1: SIMODULA Script for an ActiveObject.**

### 6.2. Modeling the Realistic Recovery Protocol

In a previous paper [14], we developed analytic models to predict the performance of four database recovery protocols described in a paper by Graham, Griffeth and Smith-Thomas [51]. The Graham paper established the correctness of the protocols in terms of enforcing recoverability and preserving (not to be confused with enforcing) serializability. In general, a *recovery scheduler* has two tools that it can use to resolve conflicts, *blocking* and *aborting*. For example, if a later transaction attempts to access an object that is being used by an earlier transaction, then the recovery scheduler can either block the later transaction until the earlier one is finished with the object, or abort the later transaction forcing it to start over. It can also defer its decision to a later operation (e.g., commit) hoping the conflict will go away.

All but one of the protocols work by not allowing transactions to read uncommitted data, thereby guaranteeing recoverability. The exception (Optimistic Recovery) allows such, giving potentially more concurrency, but risking difficulties such as cascaded aborts, and group commits or aborts. In order to simplify descriptions and facilitate analysis, we will use a locking abstraction. Thus, for all of the recovery protocols considered, the transactions acquire write *locks* on objects before writing the object. (We use locks in a generic sense to mean actual locks or something functionally equivalent.) The locks are then held until the transaction either commits or aborts. The protocols differ only in the semantics of the locks. The four recovery protocols can now be succinctly described as follows:

1. *Realistic* - transactions attempting to read *locked* objects are blocked;
2. *Optimistic* - transactions that have read *locked* objects that have as yet not been unlocked are blocked at the commit phase;
3. *Pessimistic* - transactions attempting to read or write *locked* objects are blocked;
4. *Paranoid* - transactions attempting to read *locked* objects are aborted.

The performance of protocols such as these may be analyzed using the procedure specified in Figure 2.

1. Determine the state *transitions* for each state of a transaction;
2. Solve the *balance equations* in terms of:

   $k$ = number of read and write operations in a transaction,
   $r$ = number of read operations in a transaction,
   $c$ = expected time required to commit a transaction,
   $a$ = expected time required to abort a transaction,
   $u$ = expected time required to partially roll back a transaction,
   $p$ = probability that an operation conflicts with a write,
   $w$ = expected time spent in a blocked state;
3. Determine $p$ & $w$ - these constants depend on global system properties;
4. Compute the dependent variables: throughput, delay and number executing.

**Figure 2: Markov Model Formulation and Analysis Procedure.**

We now develop in detail the analytic model for the Realistic Recovery protocol (the models for the other recovery protocols may be developed in a similar fashion). The key variables in analyzing the Realistic Recovery protocol are $p$ (probability that a read† must wait for a write to be committed) and $w$ (mean waiting time). These variables can be expressed as functions of the state of the system. We have assumed that the access pattern to the objects is uniform, implying that the probability that an access results in a conflict is simply the ratio of the number of locks held to the total number of objects.

$$p = L / D$$

The total number of locks held by all the transactions is computed by tallying $l_j$ locks for each transaction in state $j$.

$$L = \sum_j l_j N_j$$

The mean waiting time can be computed by a similar formula. Let $W_j$ be the expected time for a transaction to complete from state $j$. The probability that a blocked transaction is waiting for a lock held by a transaction in state $j$ is simply $l_j N_j / L$. Hence, the mean waiting time is given by the weighted sum.

---

† Because of the multiversion nature of the Realistic Recovery protocol, it has no write-write conflicts, so that the only operation that can conflict with a write is a read operation.

$$w = L^{-1}\sum_j l_j N_j W_j$$

The behavior of a typical transaction following the Realistic protocol is described by the Markov chain represented by the state transition diagram in Figure 3. The states in the diagram have the following meanings: State 0 is the *start state*, states 1 to $r$ are *read states*, states $r+1$ to $k$ are *write states*, state $k+1$ is the *commit state*, while states $k+2$ to $k+r+1$ are *blocked states*.

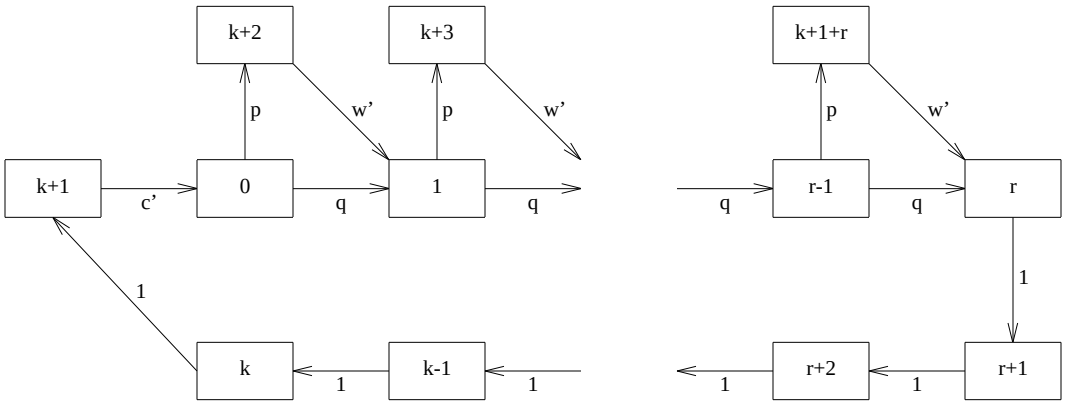


**Figure 3: Realistic Recovery Protocol.**

The transitions between states represent requests. For example, a transition out of states 0 to $r-1$ represents a read request; with probability $q = 1-p$ the request will be granted, so the next read state will be entered; and with probability $p$ the request will be denied, so that a blocked state will be entered. The basic rate at which operations are requested (state transitions) is given to be 1. It could as well have been given by μ (just multiply all of the rates by μ, which would then factor out).

To find the equilibrium solution, we apply the *balance equations* as follows. The transition rate into a read state is $q$ from the previous state and $1/w = w'$ from the preceding blocked state, while the rate out is 1. The rates into and out of a write state are simply equal to 1. The rate into the start state is $1/c = c'$ from the commit state, while the rate out is 1. Finally, the rate into a

blocked state is $p$ from a read (or the start) state, while the output rate is $1/w$.

| **Input** | = | **Output** | |
|---|---|---|---|
| $q\,\pi_{j-1} + 1/w\,\pi_{k+1+j}$ | = | $\pi_j$ | $j = 1 \cdots r$ |
| $\pi_{j-1}$ | = | $\pi_j$ | $j = r+1 \cdots k$ |
| $1/c\,\pi_{k+1}$ | = | $\pi_0$ | |
| $p\,\pi_{j-1}$ | = | $1/w\,\pi_{k+1+j}$ | $j = 1 \cdots r$ |

Hence solving for $\pi_j$ we get

$$\begin{aligned}
\pi_j &= \pi_0 && j = 0 \cdots k \\
&= c\,\pi_0 && j = k+1 \\
&= pw\,\pi_0 && j = k+2 \cdots k+r+1 \\
\sum_j \pi_j &= \pi_0(k + 1 + c + rpw) && = \pi_0(t + rpw) = 1
\end{aligned}$$

where $t = k+1+c$.

We have now solved for the state probabilities, the $\pi_j's$, in terms of several constants, where $k$, $c$, and $r$ are input parameters, and $p$ and $w$ can be derived from the other parameters including $\lambda = M/D$, the *congestion ratio*. The probability of conflict $p$ equals the ratio of the mean number of locks held $L$ to the total number of objects $D$. Applying the formula $L = \sum_j l_j N_j$ where $l_{r+j} = j$ for $j = 1 \cdots s$ (write), $l_{k+1} = s$ (commit), and $l_j = 0$ otherwise, and recalling that $N_j = \pi_j M$, we get

$$p = L/D = \lambda \sum_j l_j \pi_j$$

$$= \lambda[\sum_{j=1}^{s} j\,\pi_{r+j} + s\,\pi_{k+1}] = \lambda\pi_0[\sum_{j=1}^{s} j + sc]$$

$$= \lambda\pi_0[s(s+1)/2 + sc] = \lambda\pi_0 s(s+1+2c)/2$$

$$= \frac{\lambda s(s+1+2c)/2}{t + rpw}$$

We are now left with the problem of determining the mean time $w$ that a transaction waits in a blocked state. A transaction can wait on only one other transaction which must be in either its write or commit phase. The mean times for a transaction to complete from one of these states are given by the following two formulae (the first for writes and the second for commits).

$$W_{r+j} = c + (s - j) + 2/3$$
$$W_{k+1} = 2c/3$$

Note that the mean time it takes a transaction to leave its current state can be determined using *residual-life theory* [50]. In this theory, if $Y$ is the holding time in a state, then the mean residual-life is $z\,E(Y)$ where $z = E(Y^2)/2E^2(Y)$. Since in our simulations we used the uniform distribution on $[0, b]$, we will use the $z$ value for this distribution which is $z = 2/3$. Therefore, applying the formula for $w$ we get

$$w = \sum_j l_j \pi_j W_j \,/\, \sum_j l_j \pi_j$$

$$= \frac{\sum_{j=1}^{s} j\,(c + s + 2/3 - j) + sc\,(2c/3)}{\sum_{j=1}^{s} j + sc}$$

$$= \frac{(c + s + 2/3)s\,(s+1)/2 + sc\,(2c/3) - s\,(s+1)(2s+1)/6}{s\,(s+1)/2 + sc}$$

$$= \frac{(s + 1 + 3c)(s+1) + 4c^2}{3(s + 1 + 2c)}$$

which can be directly computed. We may now find $p$ by solving the following quadratic equation

$$rwp^2 + tp - \lambda s\,((s+1)/2 + c) = 0$$

so that choosing the correct root, the one in (0, 1), we get

$$p = \frac{-t + [t^2 + 4rws\lambda((s+1)/2 + c]^{1/2}}{2rw}$$

For this protocol the equations turn out to be particularly simple. In general, we will have two

simultaneous nonlinear equations for the unknowns $p$ and $w$. There are several good numerical algorithms (e.g., Newton's method) for solving such equations.

Having determined the values for $p$ and $w$, we can now compute the *performance measures* of primary interest. The probability a transaction is in state 0 is

$$\pi_0 = (t + rwp)^{-1}$$

The mean number of transactions executing may be computed by summing over all nonblocked nodes.

$$N_E = \sum_{j=0}^{k+1} N_j = t\,\pi_0 M$$

The throughput in committed transactions per unit time is simply the mean rate at which transactions leave node $k+1$. Since in node $k+1$ we have $N_{k+1}$ transactions each completing at rate $\mu_{k+1} = 1/c$ we get

$$\theta = N_{k+1}\mu_{k+1} = \pi_0 M$$

Finally, the mean total sojourn time $T$ (from start to completion) for transactions can be found by using Little's law

$$T = M/\theta$$

## 7. Concurrency Control Protocol Models

In this section, we discuss models for the two concurrency control protocols, the Multiversion Timestamp Ordering protocol and the Time Warp protocol. The two protocols are similar in that they both are based upon the use of timestamps to order the execution of transactions. Also, they both attempt to achieve high performance by exploiting the fact that there are multiple versions of objects. A Markov model for the Multiversion Timestamp Ordering protocol may be constructed by starting with the Markov chain for the Realistic Recovery protocol, and adding arcs leading to an abort state to each of the write states.

In our modeling study, we consider two possible dependency scenarios for the anti-transaction variant of the Time Warp protocol.

1. High Dependency Scenario. We assume for the sake of simplicity (both in modeling and in actual protocol implementations) that all of the writes are value dependent, and that no conditionally dependent reads need to be performed. Thus, a transaction in its write phase will only need to be backed up to the state preceding the write phase. If the transaction is still in its read phase, the situation is even better. The transaction simply needs to do its $i^{th}$ read again. For example, if transaction $T_9$ has issued the following read operations and is currently working on the last one,

$$R_9(O_{10})\,R_9(O_{15})\,R_9(O_{20})\,R_9(O_{25})$$

then $i = 4$. However, assuming transaction $T_8$ now writes a new version of object $O_{20}$, $R_9(O_{20})$ will need to be redone, so as soon as $R_9(O_{25})$ finishes†, transaction $T_9$ does not go on to the next read ($i = 5$), rather it remains in the current Markov state and performs the read of object $O_{20}$ again.

2. Low Dependency Scenario. More optimistic scenarios are also possible. For example, one or a few writes could be dependent on the read-in-question, while the other operations are independent.

In this paper we present results for the high dependency scenario (see [29] for results assuming low dependency). The state transition diagram shown in Figure 4 illustrates the behavior of a typical transaction under scenario 1. The states in the diagram have the following meanings: State 0 is the *start state*, states 1 to $r$ are *read states*, states $r+1$ to $k$ are *write states*, state $k+1$ is the *pre-commit state*, state $k+2$ is the *commit state*, while state $k+3$ is the *rollback state*. In this diagram $p$ = probability a read should have read a different version. The other parameters are $q = 1 - rp$, $c' = 2/c$, and $u' = 1/u$. Once a formula for $p$ has been developed,

---

† Whether to interrupt $R_9(O_{25})$, let the two reads proceed in parallel, or initiate $R_9(O_{20})$ later, is operation and system dependent (see [23]).

the balance equations may be solved in the same fashion that they were for the Realistic Recovery protocol. (Note that the transitions denoted by self loops do not affect the balance equations, since they are not transitions out of the state. They are simply there to convey extra information about what can happen while a transaction is in a read state.)

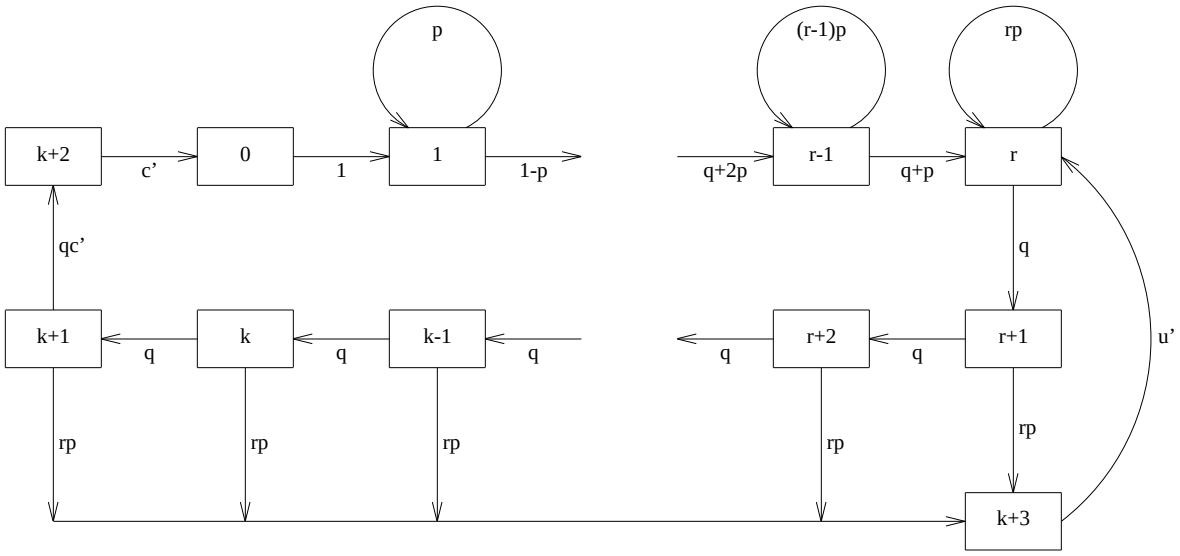


**Figure 4: Time Warp Protocol.**

## 8. Simulation Results

With *D* fixed at 500 active objects and the object access time distributed uniformly from 0 to 20 milliseconds (expected value of 10 milliseconds), four experiments were conducted. Each experiment generated simulation results for the Time Warp protocol (anti-transaction variant under scenario 1), the Multiversion Timestamp Ordering protocol and the Realistic Recovery protocol. These results are summarized in Tables 1-4. These tables show the throughput, and the number of rollbacks, anti-transactions, aborts and blocks per 100 committed transactions. (The method of batch means was used to control the simulation; 5 batches each representing 100 committed transactions were produced; the performance measures were computed on this basis.)

Table 1: $k = 12,\ r = 6,\ c = 3.5,\ a = 2.5,$ and $u = 2$
Table 2: $k = 12,\ r = 7,\ c = 3.5,\ a = 2.5,$ and $u = 2$
Table 3: $k = 12,\ r = 8,\ c = 3.5,\ a = 2.5,$ and $u = 2$
Table 4: $k = 12,\ r = 9,\ c = 3.5,\ a = 2.5,$ and $u = 2$

Note, $c = 3.5$ means that the expected time to commit is 35 milliseconds (it is realistic for commits to take a few times longer than a write). The results of the first and third experiment are also displayed graphically in Figure 5 and Figure 6, respectively. These figures display the *throughput* in committed transactions per second (tps). In particular, the throughput is plotted versus the level of concurrency (M), which ranges from 10 (low concurrency) to 70 (high concurrency). In the first experiment reads and writes were equally likely (ReadProb = 0.50, $k = 12$, $r = 6$), while in the third experiment reads were more common (ReadProb = 0.67, $k = 12$, $r = 8$).

In previous studies [15], the Multiversion Timestamp Ordering (MVTO) protocol was found to perform significantly better than the standard two-phase locking protocol, if the system had the resources necessary for high concurrency. From the results given in this paper, it appears that the Time Warp (TW) protocol's overall performance is even better. For the equiprobable (ReadProb = 0.50) case, the peak throughput improves from 94 tps for the MVTO protocol to 119 tps for the TW protocol. This is a 27% improvement in peak throughput.

Even more dramatic is the case in which reads are more common (ReadProb = 0.67) than writes. In this case, the peak throughput improves from 104 tps for the MVTO protocol to 146 tps for the TW protocol. This is a substantial 40% improvement in peak throughput. (Note, the Realistic Recovery protocol exhibits much greater throughput, peaking above 300 tps. This is expected since recoverability is a much easier condition to enforce than serializability.)

Although the number of rollbacks performed by the TW protocol is usually greater than the sum of the aborts and blocks (blocks are caused by the embedded Realistic Recovery protocol) performed by the MVTO protocol, TW exhibits better performance since rollbacks are less costly operations. Not only is $u$ less than $a$, but an abort will necessitate the resubmission of an entire transaction, while a rollback will allow the transaction to continue. On the negative side, however, when the concurrency level is very high, the rate of rolling back begins to explode leading to a sharp downturn in throughput. The problem of cascaded rollbacks escalates to the point where more work is done in reversing time than in progressing time. Finally, as anti-transactions are relatively costly, it is fortunate that their frequency is much lower than that of

rollbacks.

## 9. Conclusions

In the field of simulation, the Time Warp protocol has been shown to be one of the better protocols for parallel and distributed simulation. In this paper, we have used simulation models to demonstrate that adaptations of the Time Warp protocol can provide superior database transaction throughput. Overall, we believe that so long as the probability of conflict (as evidenced by the rate of rollback) does not get too high, the Time Warp protocol has an excellent potential for high performance.

Much work remains to be done in the future. The most important next step is to deal with the situation in which the use of anti-transactions is not feasible. We plan to study the performance of the deferred commitment variant and the Hybrid variant of the Time Warp protocol to handle this situation. The performance of other dependency scenarios also needs to be investigated. As the degree of dependency within transactions is reduced, the cost of aborting a transaction becomes much greater than the cost of partially rolling back a transaction, thereby giving Time Warp an even bigger advantage. Additional improvements in the Time Warp variants discussed in this paper might be found by incorporating some optimizations such as lazy cancellation or lazy rollback [52]. Finally, other simulation protocols, such as Chandy-Misra [11, 12, 40] and Space-Time [11, 12, 53] need to be considered for adaptation to the problem of database consistency.

**Acknowledgements**: The authors would like to express their sincere appreciation to Jane Cameron, K.C. Lee, Sanjai Narain, Abel Weinrib, and Nilesh Parate for their thoughtful comments and suggestions.

**Table 1: Summary of Simulation Results for k = 12 and r = 6.**

| | Time Warp | | | Multi Timestamp | | | Realistic | |
|---|---|---|---|---|---|---|---|---|
| *M* | *rbacks* | *antis* | *thruput* | *aborts* | *blocks* | *thruput* | *blocks* | *thruput* |
| 10 | 74.40 | 1.20 | 55.31 | 29.00 | 16.40 | 51.24 | 13.00 | 62.21 |
| 20 | 195.20 | 4.40 | 92.84 | 72.40 | 38.60 | 78.82 | 25.80 | 116.60 |
| 30 | 308.60 | 10.80 | 119.47 | 122.20 | 72.60 | 93.42 | 40.80 | 164.83 |
| 40 | 692.20 | 25.20 | 113.58 | 209.00 | 105.80 | 94.27 | 53.20 | 215.15 |
| 50 | 1476.60 | 75.20 | 86.72 | 321.40 | 161.60 | 87.11 | 61.80 | 255.39 |
| 60 | 2647.60 | 128.60 | 70.53 | 465.00 | 244.40 | 83.61 | 70.60 | 298.86 |
| 70 | 8400.33 | 441.83 | 60.28 | 618.20 | 293.60 | 76.34 | 79.60 | 340.54 |

**Table 2: Summary of Simulation Results for k = 12 and r = 7.**

| | Time Warp | | | Multi Timestamp | | | Realistic | |
|---|---|---|---|---|---|---|---|---|
| *M* | *rbacks* | *antis* | *thruput* | *aborts* | *blocks* | *thruput* | *blocks* | *thruput* |
| 10 | 63.00 | 0.80 | 56.04 | 30.40 | 12.80 | 51.23 | 8.00 | 62.89 |
| 20 | 154.80 | 5.40 | 96.00 | 69.40 | 33.60 | 79.28 | 25.40 | 116.71 |
| 30 | 290.40 | 10.40 | 119.60 | 126.80 | 56.40 | 91.81 | 36.80 | 168.79 |
| 40 | 469.60 | 21.60 | 122.31 | 200.20 | 87.40 | 96.10 | 46.80 | 215.24 |
| 50 | 1120.80 | 67.80 | 96.96 | 327.80 | 124.00 | 84.98 | 63.60 | 261.52 |
| 60 | 1578.00 | 81.20 | 90.16 | 405.20 | 164.40 | 88.95 | 69.60 | 305.90 |
| 70 | 4743.40 | 282.20 | 48.42 | 576.80 | 220.40 | 77.87 | 78.00 | 347.19 |

**Table 3: Summary of Simulation Results for k = 12 and r = 8.**

| | Time Warp | | | Multi Timestamp | | | Realistic | |
|---|---|---|---|---|---|---|---|---|
| *M* | *rbacks* | *antis* | *thruput* | *aborts* | *blocks* | *thruput* | *blocks* | *thruput* |
| 10 | 38.20 | 0.60 | 56.80 | 25.60 | 8.80 | 52.89 | 11.20 | 62.75 |
| 20 | 96.20 | 3.00 | 100.82 | 60.80 | 25.40 | 81.59 | 16.80 | 120.23 |
| 30 | 193.40 | 9.40 | 125.05 | 119.20 | 41.80 | 93.30 | 32.80 | 171.04 |
| 40 | 332.60 | 14.40 | 143.29 | 167.20 | 57.00 | 102.70 | 40.40 | 218.44 |
| 50 | 498.40 | 27.80 | 145.90 | 239.60 | 77.20 | 104.28 | 51.40 | 267.46 |
| 60 | 999.40 | 69.20 | 112.11 | 324.20 | 109.20 | 100.45 | 61.60 | 312.06 |
| 70 | 1583.60 | 116.80 | 93.96 | 443.20 | 141.20 | 94.51 | 65.60 | 354.69 |

**Table 4: Summary of Simulation Results for k = 12 and r = 9.**

| | Time Warp | | | Multi Timestamp | | | Realistic | |
|---|---|---|---|---|---|---|---|---|
| *M* | *rbacks* | *antis* | *thruput* | *aborts* | *blocks* | *thruput* | *blocks* | *thruput* |
| 10 | 25.60 | 0.60 | 58.31 | 21.60 | 6.40 | 54.62 | 6.00 | 63.44 |
| 20 | 70.80 | 3.00 | 103.85 | 50.40 | 14.20 | 89.24 | 14.40 | 120.41 |
| 30 | 138.40 | 7.00 | 136.94 | 78.20 | 26.80 | 109.51 | 28.00 | 172.41 |
| 40 | 168.60 | 8.40 | 169.35 | 125.80 | 34.20 | 119.22 | 35.00 | 226.01 |
| 50 | 257.60 | 15.40 | 182.91 | 172.80 | 51.60 | 122.60 | 44.80 | 271.19 |
| 60 | 403.80 | 32.20 | 175.01 | 228.00 | 56.00 | 125.77 | 45.60 | 322.09 |
| 70 | 546.00 | 40.40 | 177.91 | 342.00 | 78.40 | 111.76 | 54.80 | 378.00 |

Figure 5

Figure 6

## 10. References

[1] J.A. Miller and O.R. Weyrich, Jr., "Query Driven Simulation Using SIMODULA," *Proceedings of the 22nd Annual Simulation Symposium*, Tampa, FL, March 1989, pp. 167-181.

[2] J.A. Miller, W.D. Potter, K.J. Kochut and O.R. Weyrich, Jr., "Model Instantiation for Query Driven Simulation in Active KDL," *Proceedings of the 23rd Annual Simulation Symposium*, Nashville, TN, April 1990, pp. 15-32.

[3] W.D. Potter, J.A. Miller, K.J. Kochut and S.W. Wood, "Supporting an Intelligent Simulation/Modeling Environment Using the Active KDL Object-Oriented Database Programming Language," *Proceedings of the Twenty-First Annual Pittsburgh Conference on Modeling and Simulation*, Pittsburgh, PA, May 1990, pp. 1895-1900.

[4] J.A. Miller and N.D. Griffeth, "Performance Modeling of Database and Simulation Protocols: Design Choices for Query Driven Simulation," *Proceedings of the 24th Annual Simulation Symposium*, New Orleans, LA, April 1991, pp. 205-216.

[5] K.J. Kochut, J.A. Miller and W.D. Potter, "Design of a CLOS Version of Active KDL: A Knowledge/Data Base System Capable of Query Driven Simulation," *Proceedings of the 1991 AI and Simulation Conference*, New Orleans, LA, April 1991, pp. 139-145.

[6] J.A. Miller, K.J. Kochut, W.D. Potter, E. Ucar and A.A. Keskin, "Query Driven Simulation Using Active KDL: A Functional Object-Oriented Database System," *International Journal in Computer Simulation*, vol. 1, no. 1, 1991, pp. 1-30.

[7] J.A. Miller, W.D. Potter, K.J. Kochut, A.A. Keskin and E. Ucar, "The Active KDL Object-Oriented Database System and Its Application to Simulation Support," *Journal of Object-Oriented Programming*, Special Issue on Databases, July-August 1991, pp. 30-45.

[8] J.A. Miller, O.R. Weyrich, Jr., W.D. Potter and V.C. Kessler, "The SIMODULA/OBJECTR Query Driven Simulation Support Environment," in *Progress in Simulation*, vol. 3, J. Leonard and G. Zobrist (Eds.), 1992. (to appear)

[9] W.D. Potter, T.A. Byrd, J.A. Miller and K.J. Kochut, "Extending Decision Support Systems: The Integration of Data, Knowledge, and Model Management," *Annals of Operations Research*, 1992. (to appear)

[10] J.A. Miller, J. Arnold, K.J. Kochut, A.J. Cuticchia and W.D. Potter, "Query Driven Simulation as a Tool for Genetic Engineers," *Proceedings of the International Conference on Simulation in Engineering Education*, Newport Beach, CA, January 1992, pp. 67-72.

[11] R.M. Fujimoto, "Optimistic Approaches to Parallel Discrete Event Simulation," *Transactions of the Society for Computer Simulation*, vol. 7, no. 2, 1990, pp. 153-191.

[12] R.M. Fujimoto, "Parallel Discrete Event Simulation," *Communications of the ACM*, vol. 33, no. 10, October 1990, pp 30-53.

[13] N.D. Griffeth and J.A. Miller, "Performance Modeling of Database Recovery Protocols," *Proceedings of the 4th Symposium on Reliability in Distributed Software and Database Systems*, Silver Spring, MD, October 1984, pp. 75-83.

[14] N.D. Griffeth and J.A. Miller, "Performance Modeling of Database Recovery Protocols," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 6, June 1985, pp. 564-572.

[15] J.A. Miller, *Markovian Analysis and Optimization of Database Recovery Protocols*, Ph.D. Thesis, Georgia Tech, August 1986.

[16] W. Kim, *Introduction to Object-Oriented Databases*, The MIT Press, Cambridge, MA, 1990.

[17] X. Liu, *Simulation of Multiversion Concurrency Control Protocols*, M.S. Thesis, November 1991.

[18] X. Liu, J.A. Miller and N.R. Parate, "Transaction Management for Object-Oriented Databases: Performance Advantages of Using Multiple Versions," *Proceedings of the 25th Annual Simulation Symposium*, Orlando, FL, April 1992, pp. 222-231.

[19] A. Biliris, "A Data Model for Engineering Design Objects," *Proceedings of the IEEE Second International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, Gaithersburg, MD, October 1989, pp. 49-58.

[20] D.R. Jefferson, "Virtual Time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, July 1985, pp. 404-425.

[21] D.R. Jefferson and A. Motro, "The Time Warp Mechanism for Database Concurrency Control," *Proceedings of the Second International Conference on Data Engineering*, Los Angeles, CA, February 1986, pp. 474-481.

[22] D.P. Reed, *Naming and Synchronization in a Decentralized Computer System*, Ph.D. Thesis, MIT, September 1978.

[23] J.A. Miller and N.R. Parate, "On the Correctness of Hybrids and Variants of Time Warp: New Protocols for Database Transaction Management," *ACM Transactions on Database Systems*, 1992. (in review)

[24] N.R. Parate, *Object-Oriented Databases: The Time Warp Protocols for Concurrency Control*, M.S. Thesis, June 1992.

[25] Y.C. Tay, *A Mean Value Performance Model for Locking in Databases*, Ph.D. Thesis, Harvard University, February 1984.

[26] Y.C. Tay, N. Goodman and R. Suri, "Locking Performance in Centralized Databases," *ACM Transactions on Database Systems*, vol. 10, no. 4, December 1985, pp. 415-462.

[27] Y.C. Tay, *Locking Performance in Centralized Databases*, Academic Press, Inc., Boston, MA, 1987.

[28] J.A. Miller, "Simulation of Database Transaction Management Protocols: Hybrids and Variants of Time Warp," *1992 Winter Simulation Conference Proceedings*, Arlington, VA, December 1992. (to appear)

[29] J.A. Miller, "Evaluation of Hybrids and Variants of Time Warp Protocols for Database Transaction Management," *IEEE Transactions on Knowledge and Data Engineering*, 1992. (in review)

[30] J.A. Miller, O.R. Weyrich, Jr., and D. Suen, "A Software Engineering Oriented Comparison of Simulation Languages," *Proceedings of the 1988 Eastern Simulation Conferences: Tools for the Simulationists*, Orlando, FL, April 1988, pp. 97-104.

[31] G. Agha, *Actors: A Model of Concurrent Computation in Distributed Systems*, The MIT Press, Cambridge, MA, 1986.

[32] R. Agrawal and D.J. DeWitt, "Integrated Concurrency Control and Recovery Mechanisms: Design and Performance Evaluation," *ACM Transactions on Database Systems*, vol. 10, no. 4, December 1985, pp. 529-564.

[33] P.A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," *ACM Computing Surveys*, vol. 13, no. 2, June 1981, pp. 185-222.

[34] P.A. Bernstein, N. Goodman and V. Hadzilacos, "Recovery Algorithms for Database Systems," Technical Report-10-83, Harvard University, March 1983.

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

[36] W. Cellary, E. Gelenbe and T. Morzy, *Concurrency Control in Distributed Database Systems*, North-Holland, Amsterdam, 1988.

[37] J.N. Gray, et al., "The Recovery Manager of the System R Database Manager," *ACM Computing Surveys*, vol. 13, no. 2, June 1981, pp. 223-242.

[38] N.D. Griffeth, "Reliability Issues in Data Engineering," Technical Report, Georgia Tech, 1986.

[39] V. Hadzilacos, "An Operational Model for Database System Reliability," *Proceedings of the 1983 Conference on Principles of Distributed Computing*, 1983, pp. 244-257.

[40] K.M. Chandy and J. Misra, "Distributed Simulation: A Case Study in the Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering*, vol. SE-5, no. 5, May 1979, pp. 440-452.

[41] J. Misra, "Distributed Discrete-Event Simulation," *ACM Computing Surveys*, vol. 18, no. 1, March 1986, pp. 39-65.

[42] C.H. Papadimitriou, P.A. Bernstein and J.B. Rothnie, "Computational Problems Related to Database Concurrency Control," *Proceedings of the Conference on Theoretical Computer Science*, Waterloo, Canada, 1978, pp. 275-282.

[43] J.D. Ullman, *Principles of Database Systems*, Computer Science Press, Rockville, MD, 1982.

[44] H. Garcia-Molina, "Using Semantic Knowledge for Transaction Processing in a Distributed System," *ACM Transactions on Database Systems*, vol. 8, no. 2, June 1983, pp. 186-213.

[45] J.E. Allchin, *An Architecture for Reliable Decentralized Systems*, Ph.D. Thesis, Georgia Tech, September 1983.

[46] F. Baskett, K.M. Chandy, R.R. Muntz and F.G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *Journal of the ACM*, vol. 22, no. 2, April 1975, pp. 248-260.

[47] E.D. Lazowska, J. Zahorjan, G.S. Graham and K.C. Sevcik, *Quantitative System Performance -- Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

[48] C.H. Sauer and K.M. Chandy, *Computer Systems Performance Modeling*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.

[49] L. Kleinrock, *Queuing Systems, Vol I: Theory*, John Wiley & Sons, Inc., NY, 1975.

[50] S.M. Ross, *Stochastic Processes*, John Wiley & Sons, Inc., NY, 1983.

[51] M.H. Graham, N.D. Griffeth and B. Smith-Thomas, "Reliable Scheduling of Transactions on Unreliable Systems," *Proceedings of the 1984 Conference on Principles of Database System*, Waterloo, Canada, 1984, pp. 300-310.

[52] D.R. Jefferson and P. Reiher, "Supercritical Speedup," *Proceedings of the 24th Annual Simulation Symposium*, New Orleans, LA, April 1991, pp. 159-168.

[53] K.M. Chandy and R. Sherman, "Space, Time, and Simulation," Proceedings of the SCS Multiconference on Distributed Simulation, March 1989, pp. 53-57.