Basic Timestamp, Multiple Version Timestamp, and Two-Phase Locking

Wen-Te K. Lin
Jerry Nolte

Computer Corporation of America
Four Cambridge Center
Cambridge, Massachusetts 02142

Abstract

Using simulation, we compare the performance of the Basic Timestamp, the Multiple Version Timestamp, and the Two-Phase Locking concurrency control algorithms. We find that in every system configuration we have simulated the Multiple Version Timestamp algorithm performs only marginally better than the Basic Timestamp algorithm. In addition, we find that when the average transaction size is small, both timestamp algorithms outperform the Two-Phase Locking algorithm. But when the average transaction size is large, the Two-Phase Locking algorithm outperforms both timestamp algorithms.

This research is supported by the Rome Air Development Center (TSR) of the Department of the Air Force under Contract Number F30602-81-C-0028. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Rome Air Development Center of the Department of the Air Force or the U.S. Government.

1. Introduction

Many distributed concurrency control algorithms have been proposed (Bad[1], BG[1], CO[1], Ell[1], Gar[1], GW[1], KR[1], Lin[3], Ros[1], SK[1], SR[1], Sto[1], Tho[1]). But how well do they perform.

A few researchers have attempted to compare the performance of different algorithms (Gar[1], Gra[1], GS[1], KP[1], Lin[1], LN[1], LN[2], LN[3], Mun[1], Pap[1], Rie[1], Sev[1], Tha[1]), and only one researcher has studied the performance of timestamp algorithms (Lin[1]).

These performance studies are very difficult to compare, and it is almost impossible to integrate their results. They not only compare different algorithms, but they also make different assumptions about system and application environments, and they employ different measures for system performance. Therefore we began a major project that compared the principal distinct distributed concurrency control and reliability algorithms, and we used the same model, assumptions, performance (output) parameters, and system and application (input) parameters. Some results of the project concerning the Two-Phase Locking algorithm have been reported in Lin[2], LN[1], LN[2], LN[3], and LN[4]. This paper reports some of the results of this project that concern timestamp algorithms and the comparison between the timestamp algorithms and the Two-Phase Locking algorithm.

In particular, this paper reports our findings about the performance of the Basic Timestamp and the Multiple Version Timestamp algorithms (BG[1]), and about the comparison of their performance with the performance of the Two-Phase Locking algorithm. We found that, contrary to our intuition, the Multiple Version Timestamp algorithm did not significantly increase the throughput of read-only transactions over the Basic Timestamp algorithm. Neither did it improve the throughput of write transactions. We also found that both timestamp algorithms performed much better than the Two-Phase Locking algorithm when the average transaction size was small. But when the average transaction size was large, the Two-Phase Locking algorithm outperformed both timestamp algorithms.

This paper is organized as follows. Section 2 describes the simulation model. Section 3 compares the performance of the Basic Timestamp with the Multiple Version Timestamp algorithms. Section 4 discusses the simulation results of a modified model in which the ratio of read-only transactions to write transactions is fixed inside the system, instead of in the incoming transaction stream. Section 5 compares the performance of the Two-Phase Locking algorithm with that of the Basic Timestamp algorithm, and Section 6 concludes the study. Section 7 contains the references.

2. The Simulation Model

The simulation model has several input parameters. The first input parameter is the database size (DZ), which is the total number of data granules in the database. We assume that every data granule has the same probability of being accessed by a request from a transaction, because in a previous study (LN[1]) we found that other database access patterns can be modeled by the random access pattern with a heavier system load.

The second input parameter is the transaction size (TZ), which is the average number of data granules requested by each transaction. The transaction size for each transaction is randomly drawn from a geometric distribution that has a mean value equal to TZ. (A geometric distribution is the discrete version of an exponential distribution.) We simulate two kinds of transactions: read-only and update (write) transactions. We simulate each read-only transaction as a sequence of read-only requests, each reading a data granule (each read-only transaction sequentially reads its data granules). We model each write transaction as a sequence of read requests, each reading a data granule, followed by a two-phase commit. Thus, each write transaction has a read phase and a commit (write) phase. During the read phase a write transaction reads sequentially its data granules, but during the write phase it issues write requests (one request per data granule) to update in parallel all the data granules that it has read.

The third input parameter is the multiprogramming level (MP), which is the number of transactions that run concurrently. There are always exactly MP transactions running in the simulation system. A new transaction is initiated as soon as an existing transaction completes. However, when an existing transaction is aborted, a new transaction that has the same transaction size but that may access different data granules is started; this action is paraphrased as "aborted and restarted" in the rest of this paper.

The fourth input parameter is the R/W ratio, which is the ratio of read-only to write transactions. Two different interpretations of R/W ratio are simulated. In the first interpretation, the R/W is fixed in the stream of incoming transactions (i.e., a newly initiated transaction is designated as a read-only or write transaction, according to the R/W ratio using a random number generator). In the second interpretation, the R/W ratio is fixed within the simulation system. Within the MP concurrently running transactions, the ratio of read-only to write transactions is always fixed at R/W. Thus, a completed read-only (write) transaction is always replaced by a new read-only (write) transaction.

The fifth input parameter is the communication delay. Communication delay is drawn from a probabilistic distribution, which varies among different simulated runs. The distributions simulated include discrete versions of hypoexponential and hyperexponential distributions, each having a different standard deviation but the same mean value of one simulation time unit. To explain communication delay, we now describe our simulation model in more detail. In the model, unless specified otherwise, processing of data requests and transmission of messages incur no delay.

There is a database manager (DM) in the simulation model. All transaction requests for data granules (either read-only or write requests) are sent to the DM by their transactions. When the DM receives a request, it delays a certain period of time before processing the request. This delay is the communication delay. After the communication delay, the DM then processes the request without incurring any more delay. Thus the communication delay encapsulates communication delay, cpu processing delay, and IO delay. The DM processes each request differently, depending on the concurrency control algorithm being simulated.

In the timestamp based algorithms, each transaction is assigned a unique timestamp when the transaction is started. Moreover, when the transaction is executed, the timestamp is attached to every data request of the transaction. If the transaction is aborted and restarted, it is assigned a new timestamp. In the case of the Basic Timestamp algorithm, the DM maintains a read timestamp and a write timestamp for each data granule. The read timestamp and write timestamp record the timestamps of the last transaction that reads and writes respectively the data granule. When the DM processes a read request from a write transaction, the DM compares the timestamp of the write transaction (which is attached to the request) with the read and write timestamps of the data granule requested. If the timestamp of the write transaction is smaller than the read timestamp of the data granule, the write transaction is restarted immediately to avoid aborting it later when it tries to commit. If the timestamp of the write transaction is smaller than the write timestamp of the data granule, then the update transaction also is restarted immediately because it tries to read the data granule after a transaction that has a greater timestamp has updated the data granule. If the timestamp of the write transaction is larger than both the read and write timestamps of the data granule, the former timestamp replaces both the latter timestamps and the request is granted by the DM. After the request is granted by the DM, the transaction immediately issues the next request.

When the DM processes a write request from a write transaction, the DM again compares the timestamp of the write transaction with the read and write timestamps of the data granule requested. If the timestamp of the transaction is smaller than the read timestamp, the DM again immediately restarts the transaction. If the timestamp of the transaction is larger than the read timestamp, but smaller than the write timestamp, the DM ignores the write request. If the timestamp of the write transaction is larger than both the read and write timestamps of the data granule, the former timestamp replaces the write timestamp of the data granule, and the request is granted. The write transaction commits only after all write requests (issued in parallel) are granted; otherwise it aborts.

Let T(t) represents the timestamp of transaction t, and R(x) and W(x) the read timestamp and write timestamp of data granule x. The above protocol used by the DM can be summarized as follows. During the read phase of a write transaction t,

```
for each x read by t,

 if T(t)<R(x) --> restart t;
 if T(t)<W(x) --> restart t;
 if T(t)>R(x) & T(t)>W(x) --> replace R by T,
                              request is granted.
```

And during the write phase of a write transaction t,

```
 if T(t)<R(x) for any x updated by t --> restart t;
 else for each x updated by t,
 if T(t)<W(x) --> update to x is ignored,
 if T(t)>W(x) --> replace W(x) by T(t),
                  update to x is granted.
```

We next describe how the DM processes a read-only transaction in the Basic Timestamp algorithm. When the DM processes a read-only request from a read-only transaction, the DM compares the timestamp of the read-only transaction with the write timestamp of the data granule requested. If the write timestamp of the data granule is larger than the transaction timestamp, the DM immediately restarts the read-only transaction; otherwise the DM approves the read-only request, and the timestamp of the read-only transaction replaces the read timestamp of the data granule if the former timestamp is greater than the latter. The read-only transaction issues the next request as soon as the previous request is approved. In summary, if read-only transaction t requests data granule x, and if:

```
 T(t)<W(x) --> restart t;
 T(t)>W(x) --> replace R(t) by T(t) if R(t)<T(t),
               request is approved.
```

Notice that, as stated previously, all the processings by the DM incurs no delay in our simulation model, except for the communication delay discussed earlier.

We next discuss how the DM processes transaction requests for the Multiple Version Timestamp Algorithm. The Multiple Version Timestamp model is very similar to the Basic Timestamp model. Conflicts between requests and data granule timestamps are dealt with in the same way. However, in the Multiple Version Timestamp model, we kept four read and four write timestamps for each data granule; the first one is the smallest and the fourth one is the largest. Thus, a read-only transaction can access earlier versions of a data granule if the timestamp of the read-only transaction is smaller than the largest write timestamp of the data granule to be accessed. But because we require an update transaction to read first what it writes, an update transaction can read only the latest version; thus, if all transactions are write transactions, this model degenerates to the Basic Timestamp model. For this reason, we did not simulate the Multiple Version Timestamp algorithm that has R/W ratio equal to 0.

Moreover, we simulated only the Multiple Version Timestamp algorithm that had four versions of each data granule, because the probability of restart for read-only transactions is already small in the single version Basic Timestamp algorithm, and because the number of versions does not affect significantly the probability of restart for update transactions.

Performance measures (output parameters) of the simulation model for the Basic Timestamp algorithm and Four Version Timestamp algorithm include system throughput (number of requests completed per time unit), and the probability of restart of read-only requests and read requests of update transactions -- the number of aborted requests divided by the number of processed requests. Since a write transaction may progress to the write phase and then conflict and abort, we also include the probability of restart of transactions (not requests) during the write phase.

We next discuss how the DM processes transaction requests for the Two-Phase Locking algorithm. When the DM receives a transaction request for a data granule, it delays a period of time before it processes the request. This delay is the communication delay discussed earlier which encapsulates communication delay, cpu delay, and IO delay.

If the request is a read-only request from a read-only transaction the DM tries to set a read lock on the data granule requested. If the data granule is already locked by a write transaction, the requesting transaction waits until it is unlocked; otherwise the DM grants the request, and the read-only transaction immediately issues the next request. If the request is a read request from a write transaction, the DM attempts to set a write lock on the data granule requested. If the granule is already locked by another transaction (either read-only or write), the requesting transaction waits until the lock is released; otherwise the request is granted. Deadlocks can occur, and they are detected periodically. When a deadlock is detected, the transaction in the deadlock cycle that holds the least number of locks is aborted. A write transaction commits its updates in parallel after it successfully completes its read phase.

The output parameters (performance measures) of the Two-Phase Locking model included in this report consist of only the system throughputs. The other output parameters and the details of the Two-Phase Locking model can be found in LN[2] and LN[3].

3. Basic Timestamp vs. Multiple Version Timestamp

We compare the simulation results of both the Basic and the Multiple Version Timestamp algorithms in this section. We simulate the communication delay by using different hypoexponential and hyperexponential distributions, but we report results only from one distribution that has a standard deviation of 0.528. Results for other distributions exhibit similar behavior, and they can be found in LN[4].

We first examine the probability that a read request (both a read-only request and a read request of update transactions during the read phase) will conflict, resulting in the restart of its transaction. Figure 1 and Figure 2 respectively show these probabilities for the Basic and the Multiple Version Timestamp algorithms. We note that because read-only transactions never restarted in the Multiple Version Timestamp model, Figure 2 contains only data for update transactions during the read phase. We note also that, for some of the heavy load cases, the system thrashed and never stabilized; therefore the data are not reliable. However, they do qualitatively indicate what is happening. Comparing these two figures, we find very little difference between the Basic Timestamp and the Multiple Version Timestamp algorithms in the probability of restart during the read phase.

We next examine the probability of restart of update transactions during the write phase. Figure 3 and Figure 4 show the results for the Basic Timestamp and the Multiple Version Timestamp algorithms, and the difference between the two figures is also very small.

For the Basic and the Multiple Version Timestamp algorithms, Figure 5 and Figure 6 show the system throughput, which is the number of completed (excluding those aborted) data requests per time unit. Notice that the average communication delay is always one and that there are always MP transactions running in the system; therefore if there is no transaction abortion, the throughput must equal MP, which is the maximum possible throughput. Combined read-only and update throughputs for system configurations that have average transaction size equal to 4 are within 10% of the possible maximum. But combined throughputs of system configurations that have average transaction size (TZ) larger than 16 are less than 30% of the maximal throughput.

These two figures show system thrashing when the average transaction size is large or when the system load is heavy. If the system is in equilibrium, write throughput should be very nearly 1/3 of the read throughput, since incoming transactions occur in that ratio. However, this is not true for TZ=32, or for TZ=16, MP=32, and 64. In these cases, the system thrashed and was jammed with long update transactions that never finished. These observations show that both timestamp algorithms performed extremely poorly during long transactions or while bearing heavy loads.

When we compare Figure 5 with Figure 6, we find little difference between these two algorithms in throughput except when the transaction size (TZ) or the system load (TZxMP/DZ) is large, in which case the throughputs are extremely low and the statistics are not reliable anyway.

From the observations made in this section, we can conclude that both algorithms perform poorly when the average transaction size is large or when the system load is very heavy. In addition, there is no significant difference in performance between the Basic Timestamp and the Multiple Version Timestamp algorithms. Additional versions of data do not improve significantly the throughput of read-only transactions. When the load is light, the probability of conflict for read-only transactions is very small; therefore additional versions of data do not increase the read-only transaction throughput. When the load is heavy, the system is jammed with long update transactions that never finish, thus locking out read-only transactions; therefore additional versions of data do not help in this case either.

One may argue that if we do not allow the system to be saturated with long update transactions, the Multiple Version Timestamp algorithm should perform better than the Basic Timestamp algorithm. We will test this argument in the next section.

4. Results of a Modified Model

In the last section, we concluded that there is no significant difference between basic timestamp and multiple-version timestamp protocols in performance, including the throughput of read-only transactions. One may argue that this conclusion is not valid because the simulation model should not have allowed update transactions to jam the system, thus locking-out read-only transactions.

To test this argument, we impose the R/W ratio limitation inside the system, instead of in the incoming transaction stream: that is, the ratio of the number of running read-only transactions to the number of running update transactions is always fixed at R/W. All other parameters of the model remain unchanged. The results are shown in Figures 7 and 8 for the basic and multiple-version timestamp protocols respectively. We include in the figures data from the previous model for comparison. These data are marked by *.

Comparing the data of the modified model with the data of the previous model, we find that by fixing the R/W ratio inside the system, instead of in the incoming transaction stream, the throughputs of read-only transactions increase tremendously when the average transaction size (TZ) is large. The reason is that when the R/W ratio is fixed inside the system, the system no longer can be saturated with long update transactions that never finish. But when the average transaction size is small, fixing the R/W ratio inside the system does not increase significantly the throughputs of read-only transactions. The reason is that the system is never saturated with long update transactions in the first place.

When we compare Figure 7 with Figure 8, we find no significant difference between the performance of the basic timestamp protocol and the multiple-version timestamp protocol. This contradicts the earlier argument that if the R/W is fixed inside the system instead of in the incoming transaction stream, the multiple-version timestamp protocol should have higher read-only transaction throughputs than the basic timestamp protocol.

The reason for this surprising result is that both timestamp protocols favor read-only transactions. Whenever there is a conflict between an active read-only transaction and an active update transaction, both protocols abort the update transaction. In both protocols, an active read-only transaction is aborted only if it conflicts with a completed update transaction that has a later timestamp, and this occurs rarely because update transactions take much longer to complete. Since read-only transactions rarely get aborted in the basic timestamp protocol, more versions of data make little difference in read-only transaction throughput.

5. Timestamp vs. Locking

In this section we compare the performance of the basic timestamp protocol with the performance of the two-phase locking protocol.

We show part of the simulation results, specifically the throughput, in Figure 9; the rest can be found in LN[2] and LN[3]. The throughput is the number of requests completed per time unit, excluding requests aborted.

Comparing Figure 9 with Figure 5, we find that when the average transaction size (TZ) is small, the basic timestamp protocol outperforms the two-phase locking protocol. But when the average transaction size is relatively large (TZ larger than 16) the two-phase locking outperforms the basic timestamp protocol.

To learn why the timestamp protocol outperforms the two-phase locking when the average transaction size is small, we examined our previous simulation results on the two-phase locking protocol ([Lin2], [NL1], [NL2]). We found that when the average transaction size is small, the probability is very small in both algorithms that two transactions will conflict with each other. However, when a conflict does occur, a transaction is more likely to conflict with a long transaction than with a short transaction. Thus, if blocking is used to resolve the conflict, as is done in the two-phase locking algorithm, the blocked transaction tends to have a long wait because long transactions take long periods of time to complete. On the other hand, a basic timestamp algorithm resolves the conflict by aborting one of the conflicting transactions which is likely to be short. Thus, when the average transaction size is small, restarting transactions by the basic timestamp protocol is better than blocking transactions by the two-phase locking protocol.

However, when the average transaction size is large, the two-phase locking algorithm performs better than a basic timestamp algorithm. Because the probability of two transactions conflicting with each other is high in this environment, conflict resolution using transaction abortion causes many transactions to be constantly aborted and restarted. Thus in this environment it is better to wait then to abort. However, we must emphasize that both algorithms perform badly, even though the two-phase locking algorithm is preferable.

We must caution that this result must be taken in the context of our simulation model assumption. In our model, we do not simulate queueing for CPU, IO devices, and communication lines. Queueing for these devices is captured in a single model parameter, the communication delay, which has an erlangian distribution. To validate the conclusions in a more realistic model, we are currently modeling explicit queueing for these devices. Our preliminary results from this model seem to reaffirm our conclusions.

Also notice that our results differ slightly from the results of Gra[1]. The variance is due to the difference in models. In Gra[1], all transactions have the same size and all transactions have the same probability of conflict and deadlock. Our simulation results show that longer transactions have higher probability of conflict and deadlock [LN4]. In addition, in Gra[1] it is assumed that the number of locks outstanding is equal to 50% of the combined size of transactions running in the system. Our simulation results show that the average number of locks outstanding decreases substantially below 50% of that size as the average transaction size gets larger [LN4]. Moreover, the basic timestamp algorithm is not studied in Gra[1].

6. Conclusions

We come to three major conclusions concerning the performance of timestamp and two-phase locking concurrency control algorithms.

First, over a wide range of system conditions, the multiple version timestamp method performs only marginally better than the basic timestamp method.

Second, when the average transaction size (TZ) is small, the basic timestamp protocol outperforms two-phase locking protocol. However, when the average transaction size is relatively large, the two-phase locking protocol outperforms the basic timestamp protocol.

Third, when the average transaction size is small, fixing the ratio of read-only transaction to update transactions inside the system does not improve system performance. But when the average transaction size is relatively large, fixing the R/W ratio inside the system significantly improves the throughput of the system, because this prevents the system from being saturated by long update transactions.

But we caution that these conclusions be taken in the context of the simulation model assumptions. Currently we are altering some of the assumptions to see whether these conclusions remain true. In particular, we are simulating the communication delay in more detail. We are breaking down the communication delay into IO processing delay, CPU processing delay, message communication delay, and data communication delay. Preliminary results from the detailed model seem to indicate that these conclusions remain true.

7. References

Bad[1] Badal, D.Z., et al., "A proposal for Distributed Concurrency Control for Partially Redundant Distributed Database System," 3rd Berkeley Workshop on Distributed Data Management and Computer Networks, 1978.

BG[1] Bernstein, P., N. Goodman, "Concurrency Control in Distributed Database Systems," ACM Computing Survey, Vol. 13, No. 2, June 1981.

CO[1] Ceri, Stefano, & Susan Owicki, "On the Use of Optimistic Methods for Concurrency Control in Distributed Databases," 6th Berkeley Workshop on Distributed Data Management & Computer Network, Feb. 16-19, 1982, Asilomar, CA.

Ell[1] Ellis, C.A., "A Robust Algorithm for Updating Duplicate Databases," 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977.

Gal[1] Galler, B.I., Ph.D. Thesis, University of Toronto, 1982.

Gar[1] Garcia-Molina, H., "Performance of Update Algorithms For Replicated Data in a Distributed Database," Ph.D. Thesis, Dept. of Computer Science, Stanford University, June 1979.

GS[1] Gelembe, E., & K. Sevcik, "Analysis of Update Synchronization for Multiple Copy Databases," 3rd Berkeley Workshop on Distributed Databases & Computer Network, 1978.

GW[1], Garcia-Molina, H., & Gio Wiederhold, "Read-Only Transactions in a Distributed Database," ACM TODS, Vol. 7, No. 2, June, 1982.

Gra[1] Gray, Jim, Pete Homan, Ron Obermarck, Hank Korth, "A Straw Man Analysis of Probability of Waiting and Deadlock," IBM Research Report, RJ3066 (38112), San Jose, CA, 1981.

KR[1] Kung, H.T., & John T. Robinson, "On Optimistic Methods for Concurrency Control," ACM TODS, Vol. 6, No. 2, June 1981.

KP[1] Kung, H.T., & C.H. Papadimitriou, "An Optimality Theory of Database Concurrency Control," Proc., ACM SIGMOD Conference, 1979.

Lin[1] Lin, W.K., "Performance Evaluation of Two Concurrency Controls Mechanisms in a Distributed Database System," Sigmod-81 International Conference on Management of Data, Ann Arbor, MI, 1981.

Lin[2] Lin, W.K., et al., "Distributed Database Control & Allocation: Semi-Annual Report," Technical Report, Computer Corporation of America, Cambridge, MA, January 8, 1982.

Lin[3] Lin, W.K., "Concurrency Control In a Multiple Copy Distributed Database System," 4th Berkeley Workshop on Distributed Data Management and Computer Networks Aug. 1979.

LN[1] Lin, W.K., J. Nolte, "Performance of Two-Phase Locking," 6th Berkeley Workshop on Distribute Data Management and Computer Networks, Feb. 16-19, 1982, Pacific Grove, CA.

LN[2] Lin, W.K., J. Nolte, "Read Only Transactions and Two-Phase Locking," 2nd Symposium on Reliability in Distributed Software and Database Systems, Pittsburgh, PA. 1982.

LN[3] Lin, W.K., J. Nolte, "Communication Delay and Two-Phase Locking," 3rd International Conference on Distributed Computing Systems, Fort Lauderdale, FL, 1982.

LN[4] Lin, W.K., J. Nolte, "Distributed Database Control and Allocation Project -- Third Semi-annual Technical Report," Computer Corp. of America, Cambridge, MA, 1982.

Pap[1] Papadimitriou, C.H., "On the Power of Locking," Proc. of 1981 ACM SIGMOD Conference.

Rie[1] Ries, D., "The Effect of Concurrency Control on Database Management System Performance," Ph.D. Thesis, Electronics Research Lab, Univ. of Cal., Berkeley, 1979.

Ros[1] Rosenkrantz, D.J., et al., "System Level Concurrency Control for Distributed Database Systems," ACM Trans. on Database System, Vol 3, No. 2, June 1978.

Sev[1] Sevcik, K.C., "Database System Performance Prediction Using an Analytical Model," 7th Conference on Very Large Data Bases, Cannes, France, 1981.

SK[1] Silberschatz, A., & Z.M. Kedem, "A Family of Locking Protocols for Database Systems that Are Modeled by Directed Graphs," IEEE Trans. on Software Engineering, Vol. SE-8, No. 6, November, 1982.

SR[1] Sterns, R.E., D.J. Rosenkrantz, et al., "Distributed Database Concurrency Controls Using Before-Values," Sigmod-81 International Conference on Management of Data, Ann Arbor, MI, 1981.

Sto[1] Stonebraker, M., et al., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," IEEE Trans. on Software Engineering, Vol SE-5, No. 3 May, 1979.

Tha[1] Thanos, C., et al., "Performance Evaluation of Two Concurrency Control Mechanisms in a Distributed Database System," Lecture Notes in Computer Science, Ed. G. Goos & J. Hartmanis, Springer-Verlag, NY, 1981.

Tho[1] Thomas, R.H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Database," ACM Transaction On Database System, Vol 4, No. 2, June 1979.

Read-Only Transaction
DZ = 4096, R/W = 3/1

	TZ		
MP	4	16	32
16	.0016	.0031	.0013
32	.0030	.0026	.0017
64	.0049	.0027	.0024

Read-Only Transaction
DZ = 8192, R/W = 3/1

	TZ		
MP	4	16	32
16	.0011	.0015	.0014
32	.0015	.0021	.0011
64	.0029	.0021	

Update Transaction
DZ = 4096, R/W = 3/1

	TZ		
MP	4	16	32
16	.0063	.0236	.0244
32	.0117	.0329	.0339
64	.0239	.0452	.0456

Update Transaction
DZ = 8192, R/W = 3/1

	TZ		
MP	4	16	32
16	.0033	.0165	.0177
32	.0067	.0244	.0254
64	.0121	.0337	

Update Transaction
DZ = 4096, R/W = 0

	TZ		
MP	4	16	32
16	.0065	.0238	.0248
32	.0127	.0333	.0342
64	.0227	.0455	.0458

Figure 1
Average Probability of Restart at Read Phase
(Basic TS)
Standard Deviation of Processing Delay = 0.528

Update Transaction
DZ = 4096, R/W = 3/1

	TZ		
MP	4	16	32
16	.0063	.0240	.0247
32	.0121	.0339	.0343
64	.0232	.0453	.0459

Update Transaction
DZ = 8192, R/W = 3/1

	TZ		
MP	4	16	32
16	.0040	.0165	.0179
32	.0077	.0242	.0255
64	.0126	.0338	.0346

Update Transaction
DZ = 4096, R/W = 0

	TZ		
MP	4	16	32
16	.0065	.0238	.0248
32	.0127	.0333	.0342
64	.0227	.0455	.0458

Figure 2
Average Probability of Restart at Read Phase
(Multiple version TS)
Standard Deviation of Communications Delay = 0.528

Update Transaction
DZ = 4096, R/W = 3/1

	TZ		
MP	4	16	32
16	.033	.270	.578
32	.050	.459	.785
64	.079	.672	.886

Update Transaction
DZ = 8192, R/W = 3/1

	TZ		
MP	4	16	32
16	.016	.190	.4702
32	.027	.138	.6149
64	.049	.476	

Update Transaction
DZ = 4096, R/W = 0

	TZ		
MP	4	16	32
16	.031	.296	.64
32	.049	.462	.91
64	.083	.834	.94

Figure 3
Average Probability of Restart at Write Phase
(Basic TS)
Standard Deviation of Processing Delay = 0.528

Update Transaction
DZ = 4096, R/W = 3/1

	TZ		
MP	4	16	32
16	.029	.267	.476
32	.048	.510	.695
64	.080	.684	.873

Update Transaction
DZ = 8192, R/W = 3/1

	TZ		
MP	4	16	32
16	.014	.165	.433
32	.031	.310	.629
64	.048	.523	.768

Update Transaction
DZ = 4096, R/W = 0

	TZ		
MP	4	16	32
16	.031	.296	.64
32	.049	.462	.91
64	.083	.834	.94

Figure 4
Average Probability of Restart at Write Phase
(Multiple Version TS)
Standard Deviation of Communications Delay = 0.528

Read-Only Transaction
DZ = 4096, R/W = 3/1

MP	TZ			
	4	8	16	32
16	11.60	7.30	3.67	0.90
32	23.21	11.80	3.29	0.43
64	42.82	14.90	1.61	0.15

Read-Only Transaction
DZ = 8192, R/W = 3/1

MP	TZ			
	4	8	16	32
16	11.84	8.6	5.89	1.65
32	23.04	14.4	6.17	1.06
64	45.50	24.5	5.12	

Update Transaction
DZ = 4096, R/W = 3/1

MP	TZ			
	4	8	16	32
16	3.72	2.34	1.25	0.23
32	7.57	3.85	0.93	0.10
64	13.80	4.80	0.42	0.03

Update Transaction
DZ = 8192, R/W = 3/1

MP	TZ			
	4	8	16	32
16	3.96	2.88	1.98	0.44
32	7.64	4.78	1.96	0.29
64	15.29	8.23	1.50	

Update Transaction
DZ = 4096, R/W = 0

MP	TZ		
	4	16	32
16	14.15	1.14	0.13
32	26.24	0.97	0.03
64	44.19	0.09	0.01

Figure 5
Through-put in Requests per Time Unit
(Basic TS)
Standard Deviation of Processing Delay = 0.528

Read-Only Transaction
DZ = 4096, R/W = 3/1

MP	TZ		
	4	16	32
16	11.8	3.6	1.3
32	23.2	1.8	0.4
64	43.5	1.6	0.3

Read-Only Transaction
DZ = 8192, R/W = 3/1

MP	TZ		
	4	16	32
16	11.6	5.7	1.6
32	23.2	6.6	1.2
64	46.0	4.5	1.0

Update Transaction
DZ = 4096, R/W = 3/1

MP	TZ		
	4	16	32
16	4.0	1.14	0.33
32	7.8	0.56	0.08
64	14.4	0.39	0.07

Update Transaction
DZ = 8192, R/W = 3/1

MP	TZ		
	4	16	32
16	4.0	1.93	0.53
32	7.5	2.19	0.31
64	15.2	1.28	0.21

Update Transaction
DZ = 4096, R/W = 0

MP	TZ		
	4	16	32
16	14.15	1.14	0.13
32	26.24	0.97	0.03
64	44.19	0.09	0.01

Figure 6
Through-put in Requests per Time Unit
(Multiple Version TS)
Standard Deviation of Communications Delay = 0.528

Through-Put (Read-Only)
DZ=8192, R/W=3/1

	TZ		
MP	4	16	32
16	10.7 (11.8)	10.3 (5.9)	x
32	x	x	22.9 (1.1)
64	47.0 (45.5)	x	x

Through-Put (Update)
DZ=8192, R/W=3/1

	TZ		
MP	4	16	32
16	4.8 (4.0)	1.2 (2.0)	x
32	x	x	.044 (.290)
64	13.3 (15.3)	x	x

Probability of Restart
(Read-Only)
DZ=8192, R/W=3/1

	TZ		
MP	4	16	32
16	.0007 (.0011)	.0008 (.0015)	x
32	x	x	.0001 (.0011)
64	.0021 (.0029)	x	x

Probability of Restart
(Update in Read Phase)
DZ=8192, R/W=3/1

	TZ		
MP	4	16	32
16	.0040 (.0033)	.038 (.017)	x
32	x	x	.025 (.025)
64	.0147 (.0123)	x	x

Probability of Restart
(Update in Write Phase)
DZ=8192, R/W=3/1

	TZ		
MP	4	16	32
16	.016 (.016)	.19 (.19)	x
32	x	x	.81 (.61)
64	.049 (.049)	x	x

Figure 7
Basic Timestamp Model with
R/W Fixed Inside the System

(Figures within parenthesis are results of the model having the R/W ratio fixed in the input stream. "x" means not available.)

Through-Put (Read-Only)
DZ=8192, R/W=3/1

	TZ		
MP	4	16	32
16	10.9 (11.6)	10.9 (5.7)	x
32	x	x	22.9 (1.2)
64	46.6 (46.0)	x	x

Through-Put (Update)
DZ=8192, R/W=3/1

	TZ		
MP	4	16	32
16	4.6 (4.0)	1.6 (1.9)	x
32	x	x	1.69 (0.31)
64	13.4 (15.2)	x	x

Probability of Restart
(Read-Only)
DZ=8192, R/W=3/1

	TZ		
MP	4	16	32
16	0 (0)	0 (0)	x
32	x	x	0 (0)
64	0 (0)	x	x

Probability of Restart
(Update in Read Phase)
DZ=8192, R/W=3/1

	TZ		
MP	4	16	32
16	.004 (.004)	.015 (.013)	x
32	x	x	.024 (.026)
64	.013 (.013)	x	x

Probability of Restart
(Update in Write Phase)
DZ=8192, R/W=3/1

	TZ		
MP	4	16	32
16	.015 (.014)	.19 (.17)	x
32	x	x	.54 (.63)
64	.049 (.048)	x	x

Figure 8
Multiple Version Timestamp
with R/W Fixed Inside the System

(Figures within parenthesis are results of the model having the R/W ratio fixed in the input stream. "x" means not available.)

Read-Only Through-Put DZ=4096, R/W=3/1			
	TZ		
MP	4	16	32
16	8.90	5.01	3.04
32	16.18	5.18	2.04
64	26.50	3.62	1.35

Read-Only Through-Put DZ=8192, R/W=3/1			
	TZ		
MP	4	16	32
16	9.46	8.04	4.29
32	17.67	9.26	3.56
64	33.18	6.64	2.45

Update Through-Put DZ=4096, R/W=3/1			
	TZ		
MP	4	16	32
16	2.89	1.73	0.92
32	5.34	1.71	0.63
64	8.82	1.23	0.48

Update Through-Put DZ=8192, R/W=3/1			
	TZ		
MP	4	16	32
16	3.04	2.65	1.43
32	5.80	3.05	1.19
64	11.02	2.13	0.87

Update Through-Put DZ=8192, R/W=0			
	TZ		
MP	4	16	32
16	11.79	7.07	3.61
32	21.51	6.69	2.98
64	36.30	5.14	2.04

Figure 9
Through-Put of Two Phase Locking