PERFORMANCE EVALUATION OF A TWO-PHASE COMMIT BASED PROTOCOL FOR DDBs

Daniel A. Menascé
Departamento de Informática
Pontifícia Universidade Católica,
22453 Rio de Janeiro, Brazil

Tatuo Nakanishi
Escola Federal de Engenharia de Itajubá
Minas Gerais, Brazil

Abstract

Many concurrency control algorithms for distributed database management systems have been proposed in the last few years, but little has been done to analyse their performance. This paper presents the specification of a concurrency control algorithm based on the two-phase commit protocol for DDBs. The results of a complete performance analysis based on an analytic model are presented and discussed.

1. Introduction

A database which has its data elements stored at several sites of a computer network is called a distributed data base (DDB). When a DDB is subject to concurrent access from many users submitting their database requests (transactions) at the same or at different sites, the semantic integrity of the database may be violated. Therefore, concurrency control mechanisms have to be incorporated in the design of the distributed DBMS (DDBMS). Many different solutions have been proposed in the last few years in the literature [1,2,3,4,5,6,7,8 and 18].

A nice and comprehensive study of concurrency control (CC) mechanisms was given by Bernstein and Goodman in [9 and 10]. Their work classify concurrency control mechanisms into timestamp based and locking based ones. An important contribution of the work in [9] is the recognition that CC mechanisms can be obtained by combining a read-write synchronization technique with a write-write synchronization technique. For instance, for timestamp based CC mechanisms they considered three different techniques for read write synchronization and four techniques for write-write synchronization, generating a total of 12 principal CC mechanisms. By introducing some modifications to the basic techniques, over 50 different CC mechanisms can be generated.

In the presence of a such a multitude of CC mechanisms one is faced with the problem of comparing their performance. Among the few works on performance of CC mechanisms we have [11,12,13,19 and 14]. This paper presents a specification and a performance evaluation of a concurrency control mechanism based on the two-phase commit protocol.

Section two of this paper presents very briefly some of the background needed to read this paper. We assume that the reader is familiar with aspects of concurrency control in distributed data bases. Section three presents the algorithm in an informal but easy to understand manner. The algorithm uses timestamp ordering to achieve write-write synchronization and uses the two-phase commit protocol to achieve atomic updates. Section four introduces the analytic model used in the analysis, and section five lists the results obtained in the analysis. Section six concludes with the presentation of several curves to illustrate the behavior of the algorithm under several circumstances. Simulation was used to verify the accuracy of the analytic model. A comparison between the simulation and the analytic results also appears in section 6.

2. Basic Concepts and Definitions

A distributed database (DDB) is a database with its data elements stored at several sites of a computer network. Certain data items may be redundantly stored at more than one site, for reliability reasons. If a copy of all data items of the database exists at every site then the DB is said to be fully replicated. Otherwise it is said to be partially replicated. In the former case, queries can be completely evaluated at a single site, while in the latter, several query processing alternatives exist. The selection of an optimum query evaluation strategy which minimizes response time has been studied by several authors [15]. We are not interested in studying this effect, but rather the impact of the degree of conflicts between transactions in their response time. Therefore, we assume that the DB is fully replicated since conflicts between transactions will occur irrespective of the degree of replication of the DB.

Users interact with the database through transactions which may be of two types: read transactions and update transactions. A read transaction is composed of one or more read actions plus any processing that may be necessary. Since the DDB is fully replicated the read actions can be performed at a single site. An update transaction, besides reading and processing, will update the DB. In this case, all copies must be updated.

The execution of an update transaction can be divided into three phases: a read and processing phase, a pre-commit phase and a commit phase. During the pre-commit phase all information necessary to update the database is stored in stable storage in all sites. This information is called an intentions list [16 and 17]. During the commit phase the database is actually updated through the execution of the intentions list. A detailed explanation of the implementation of intentions lists can be found in [17].

For each transaction there is a process, called transaction controller (TC), that manages the execution of the transaction. The TC is assumed to reside at the site where the transaction is introduced in the system (the site of origin of the transaction). Upon arrival at the system, a transaction is assigned a unique timestamp.

3. The Concurrency Control Algorithm

The algorithm presented here is rather simple and is based on well known ideas such as the use of:

i) two-phase commit protocol for atomic implementation of update transactions

ii) locking for conflict detection between transactions

iii) timestamps, delays and restarts for conflict resolution between transactions.

The execution of a read transaction is totally accomplished at the site of origin of the transaction. The read and computation actions of an update transaction are executed at its site of origin while the update actions have to be executed at all sites. During the first phase of the two-phase commit, update information is broadcast to all sites and during the second phase the updates are actually committed to the database.

Locks are used by transactions to indicate their intention to read or update the DB. (A detailed description of the types of locks considered here is given later). In this way, a transaction can be prevented from accessing a database resource if it is locked in an incompatible way by another transaction. Instead of waiting in a queue for the desired resource, transactions retry after a certain delay.

An update transaction must successfully lock all needed resources at its site of origin before reading the database. Locks are held until the end of transaction. During the execution of an update transaction, no other conflicting transaction will be able to lock the resources it needs. These transactions will have to retry some time later.

Once an update transaction locks its resources at its site of origin it must try to lock these resources at the remaining sites. As it is not possible to instantaneously lock the needed resources at all sites, deadlocks may arise. In order to deal with this problem, timestamp are used to establish priorities between transactions. A timestamp is assigned to a transaction by its controller. This timestamp indicates the instant when the transaction succeeded in locking all needed resources. Transactions with smaller timestamps are given higher priority. During the precommit phase an update transaction may be rejected by a transaction of higher priority. The rejected transaction must free all acquired resources and be resubmitted.

An update transaction may start its read actions at a given site even if there are other conflicting read transactions but no other conflicting update transaction at that site. However, in order to start the execution of its update actions an update transaction must wait until all conflicting read transactions finish.

Propagation of lock requests and update information to any site is accomplished in a single message. Let us now describe the different types of messages and lock modes used in our concurrency control algorithm.

The following messages are involved.

a. PRECOMMIT(t)	:	this message is broadcast from the transaction controller of transaction t to all other sites, requesting that they store their intentions lists in stable storage.
b. ACK(t)	:	this message is sent by a site to the TC of t to indicate that the PRECOMMIT message was accepted and that the intentions list of t was already stored in stable storage.
c. REJECT(t)	:	this message is broadcast by a site to all other sites to indicate the rejection of the PRECOMMIT of transaction t.
d. COMMIT(t)	:	this message is broadcast by the transaction controller of t to tell all other sites to execute their intentions lists.

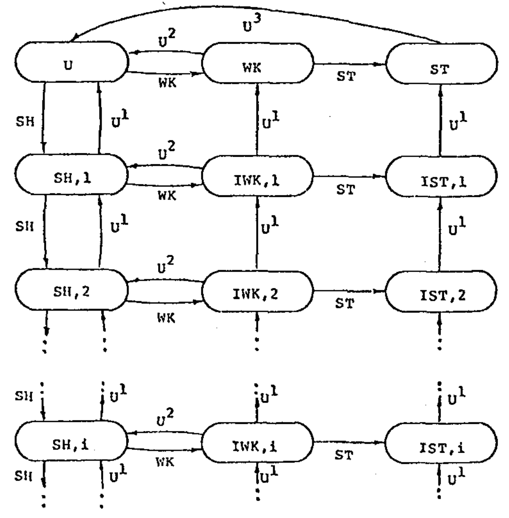
Any database item may be in six different states:

1. U	:	unlocked
2. (SH,i)	:	locked in share mode by i read transactions. This lock mode is used for read-only transactions.
3. WK	:	locked in weak mode. An item is put into this state to indicate that an update transaction intends to update it. This type of lock may be lifted by other transactions under certain circumstances as will be seen later.
4. ST	:	locked in strong mode. This indicates that the item is being update. This type of lock may not be lifted by another transaction.
5. (IWK,i)	:	locked in intention WK mode for one update transaction and locked in SH mode by i(i>0) read transactions. Whenever a WK lock would be placed on an item locked in SH mode, an IWK lock is used instead. When all read transactions release the SH lock on the item, the IWK lock is converted to WK lock.

6. (IST,i): locked in intention ST mode by an update transaction and locked in SH mode by i (i>0) read transactions.

Figure 1 illustrates the possible transitions for the state of a database resource at a given site. The labels on the arrows indicate the kind of lock request which caused the transition. The labels SH, WK, ST and U indicate a request to place the resource in SH, WK, ST and U mode respectively. A superscript is used to distinguish different situations when resources are freed.

An update transaction is only allowed to request to place a resource in ST or IST mode if it is in WK or IWK for the same transaction.



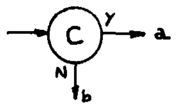
U^1 : the resource is freed by a read transaction.

U^2 : the resource is freed by an update transaction rejected during the precommit phase.

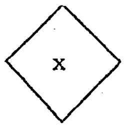
U^3 : the resource is freed by an update transaction which completes.

Figure 1 - Transitions Between Lock Modes

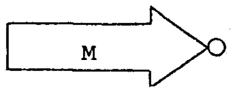
In order to describe the algorithm a graphical representation will be used to help the reader to visualize what is really going on. The following graphical conventions are used:



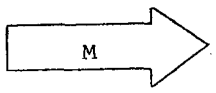
if condition C is true then go to a else go to b.



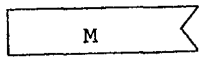
place all items of a transaction in x mode. The lock mode which results from this request can be found by examining the diagram of figure 1.



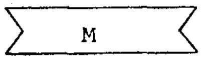
broadcast message M.



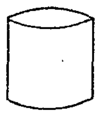
send message M.



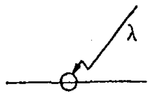
arrival of message M.



save message M.



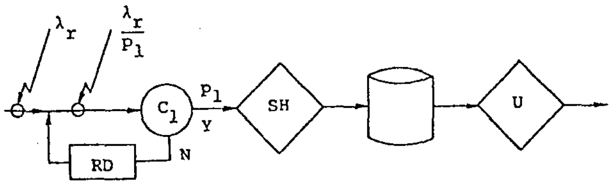
access the database.



the rate at which transactions cross the circled path is λ.

waiting for a message to arrive.

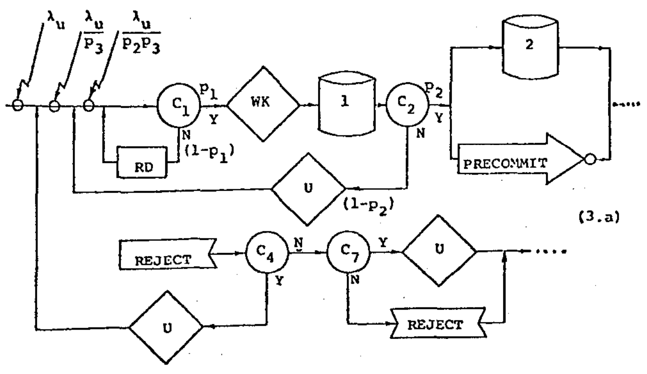
We are now ready to describe the algorithm a correctness proof of which can be found in [20]. The graphical description that follows illustrates the algorithm as executed at a single site. All sites implement the same algorithm. Let us first consider read transactions (see figure 2). When read transactions arrive they perform the test indicated in the figure. If the answer is no (N exit) then the test is repeated after a certain delay. Otherwise, the necessary items are locked in share mode, the database is read and the resources are



C_1 : are all the resources needed in U or SH mode?

Figure 2 - Read Transaction

unlocked. The algorithm for update transactions is shown in figure 3.



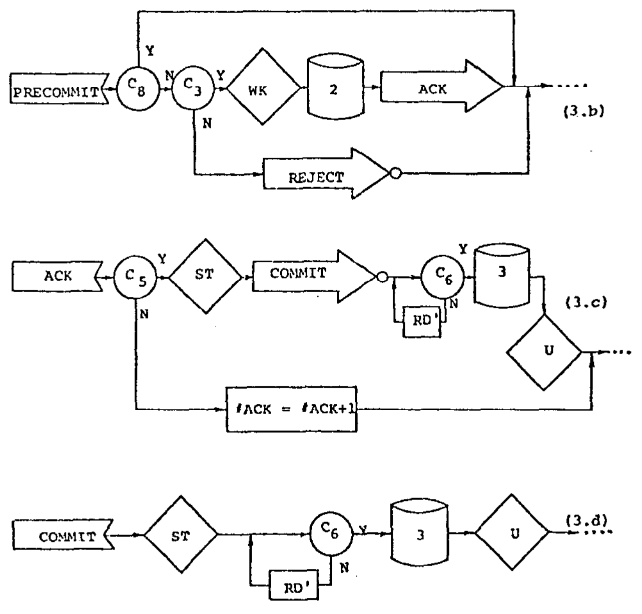


Figure 3 - Update Transaction

The conditions for the tests in figure 3 are

C_1 : are all needed resources in U or SH mode?

C_2 : are all needed resources still locked in WK or IWK mode for the transaction?

C_3 : are the needed resources free, locked in SH mode or locked in WK or IWK mode for transactions with greater timestamp?

C_4 : is the node the TC of the transaction?

C_5 : is the number of received ACKs (#ACK) equal to N - 2?

C_6 : are all the needed resources in ST mode?

C_7 : has the site already received a PRECOMMIT message for this transaction?

C_8 : is there a saved REJECT for the transaction?

Figure 3.a represents the arrival of an update transaction at a given node. The transaction must initially read the database (see database access box labelled 1), and then, in parallel, start to store its intentions list in stable storage(box labelled 2) and broadcast the PRECOMMIT messages. The arrival of a REJECT message at the site of the TC causes the transaction to be restarted. Figure 3.b shows that when a PRECOMMIT message is received it can be either accepted or rejected. Notice that a PRECOMMIT message may lift locks placed in WK or IWK mode by transactions of greater timestamp. This is the reason why the test of condition C_2 in figure 3.a is necessary. Figure 3.c shows that when all ACKs are received by the TC of the transaction, a COMMIT message is broadcast to all other sites. The database access box labelled 3 represents the execution of the intentions list. Finally, figure 3.d shows what happens when a COMMIT message is received.

As it can be seen from the test of condition C_3, a precommit of a transaction with a smaller timestamp has higher priority over one of greater timestamp. In other words, precommits of conflicting transactions are accepted in timestamp order. This is how write-write synchronization is achieved (see Bernstein 80a). In [20] a proof that any execution generated by this algorithm is serializable, and that read transactions always see consistent data, can be found.

Let us comment on the necessity of the test of condition C_6 before actually executing the intentions list of a transaction. Whenever a node is ready to commit, all resources involved are either in ST or IST mode. If all resources are in ST node then the commit can be done right away. Otherwise, it must be delayed until the IST mode is converted to ST mode. This will happen when read transactions that were reading when the IST mode was installed finish to read. It should be remembered that after the IST mode is granted, no other read transaction is allowed to start. Therefore, it is very likely that no wait will be necessary before executing the commit, since an interval of at least 2T time units (T is the message transmission time) will elapse between the granting of an IST lock and the execution of the commit.

A final comment on the algorithm is that there is no situation in which a transaction has to wait for database resources. This is avoided using delays and retries.

4. Model Definition and Parameters

In this section we present the assumptions and parameters used to construct the model analysed in this paper.

4.1 Transaction Characterization

The set of resources read by a transaction is called its read set and the set of resources updated by it is called its write set. Our first assumption for transactions is:

T1 : The size of the read set is the same as the size of the write set and is the same (constant) for every transaction.

The next assumption is concerned with the arrival process of transactions.

T2 : Read and update transactions arrive at the system according to a Poisson process. The total arrival rate of read or update transactions is equally divided among all sites, i.e. the load is balanced.

The important measure of interest for transactions is their response time. The response time of a transaction is measured as the time interval between the instant the transaction is submitted to the system and the instant when the user is notified of the completion of the transaction. Update transactions are considered to be completed when all sites have written their intentions list into stable storage, since from this point on, the modifications generated by the transaction will be reflected into the database no matter how many failures occur. For read transactions the completion instant is when all read and processing actions have been executed.

4.2 Network and DDB Assumptions

The assumptions regarding the database are the following:

D1 : The database is fully replicated

D2 : The total number of items which compose the database is constant, i.e. we disregard insertions and deletions to the database.

The assumptions regarding the network are the following:

N1 : The network does not lose nor duplicates messages. Actually, the transport layer at each node implements this property for messages.

N2 : Messages that go from site A to site B arrive at their destination in the same order they were sent.

N3 : The transmission time between a pair of nodes is assumed to be constant.

4.3 Site Assumptions

This section considers the assumptions about the computer system of a site. Our model takes into account that transactions use both CPU and I/O devices and that they compete for the use of these resources. The model used to analyze this effect is shown in figure 4 and is a network of queues.

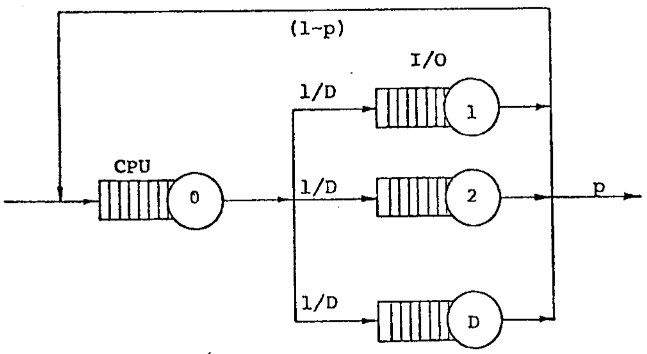


Figure 4 - Model of a Site

As we can see, there are (D + 1) queues. Queue 0 is the CPU queue and the remaining ones are associated with the D existing I/O devices. After being served at the CPU, transactions move to any one of the I/O devices with equal probability. The probability p shown in figure 4 was introduced to add another characteristic to transactions, namely their complexity. Complex transactions will have to go through several cycles before they leave the system (small values of p), while simple transactions will require few cycles (p closer to one). The CPU and all I/O devices are assumed to be exponential servers with FCFS queueing discipline.

4.4 List of Parameters and Performance Measures

A list of all parameters considered in the model is given below:

n = number of resources used by a transaction

λ_r = average arrival rate of read transactions per node.

λ_u = average arrival rate of update transactions per node.

p = probability that a transaction leaves the computer system after being served by an I/O device.

M = number of resources of the database.

D = number of I/O devices per node.

N = number of sites in the network.

T = transmission time of a message between a pair of nodes.

μ_{cpu} = average CPU processing rate.

μ_{io} = average I/O device processing rate.

RD = resubmission delay during the read phase.

The performance measures of interest are

R_r = average response time of read transactions.

R_u = average response time of update transactions.

5. Performance Evaluation Results

This section presents the results obtained in our analysis. All derivations can be found in [20].

Before we present the results, we would like to comment on some assumptions we made. The first strong assumption used throughout the analysis is that the arrival process of any kind of message is a Poisson process. This approach was taken because otherwise the analytic model would become extremely hard to manage. In order to verify how good an approximation this is, we built a simulation program in SIMSCRIPT II. The simulation results agreed very well with the analitic ones, as one can see from table 1.

The other assumption that we would like to comment on is assumption T1, namely that all transactions use a constant number, n, of resources. This choice was adopted because it simplified the analysis and also because we did not have actual data that could serve as an indication of what would be a realistic distribution of the size of the read or write sets. Our assumption implies that the probability of conflict between two transactions, denoted by PC is

$$PC = \begin{cases} 1 - \dfrac{\binom{M-n}{n}}{\binom{M}{n}} & \text{if } n \le M/2 \\ 1 & \text{otherwise} \end{cases} \qquad (1)$$

We assume that the average time spent at the computer system to execute the read actions of a transaction and the average time to store its intentions list in stable storage are the same. Let this time be denoted by $\bar{t}$. Then we have the following result.

Result 1:

$$\bar{t} = \frac{1}{\beta}\left(\frac{\rho_{cpu}}{1-\rho_{cpu}} + \frac{D\rho_{io}}{1-\rho_{io}}\right) \qquad (2)$$

where $\rho_{cpu} = \frac{\beta}{p\mu_{cpu}}$

$$\rho_{io} = \frac{\beta}{pD\mu_{io}}$$

$$\beta = \lambda_r + \frac{\lambda_u}{p_3}\left[1 + \frac{1}{p_2} + (N-1)p_3^{\frac{1}{N-1}}\right]$$

and

$p_2 \overset{\Delta}{=} P_r$ [an update transaction is not rejected after read-computation phase but before entering its precommit phase]

$p_3 \overset{\Delta}{=}$ [an update transaction is not rejected during its precommit phase]

Result 2: The average response time, R_r, of a read transaction is given by

$$R_r = RD\left(\frac{1-p_1}{p_1}\right) + \bar{t} \qquad (3)$$

where,

$p_1 \overset{\Delta}{=} P_r$ [a transaction finds all needed resources free or locked in SH mode at its site of origin]

Result 3: The average response time of an update transaction is given by

$$R_u = \frac{R_r}{p_2 p_3} + \frac{2T}{p_3} + \overline{\max_{N-1}}[\tilde{t}] \qquad (4)$$

where the notation $\overline{\max_j}[\tilde{v}]$ indicates the expected value of a random variable defined as the maximum between j random variables distributed as the random variable $\tilde{v}$. Since $\tilde{t}$ is the random variable which indicates the time a node takes to process an external PRECOMMIT message, $\overline{\max_{N-1}}[\tilde{t}]$ is the average maximum time it takes the PRECOMMIT message to be processed at the (N-1) nodes.

We show , in Result 4, an expression for the average of the maximum of a given number of identically distributed random variables. This derivation is rather simple if we assume that $\tilde{t}$ is exponentially distributed with an average $\bar{t}$.

Result 4: Let $\tilde{y}$ be a random variable defined as $\tilde{y} = \max[\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_k]$ where $\tilde{x}_1, \ldots, \tilde{x}_k$ are identical and independently distributed random variables. Let the $\tilde{x}_i$ r.v.s. be all exponentially distributed with average $\bar{x}$. Then, the expected value of $\tilde{y}$, $\bar{y}$ is given by

$$\bar{y} = \overline{\max_k}[\tilde{x}] = k\,\bar{x}\sum_{j=0}^{k-1}\frac{\binom{k-1}{j}(-1)^j}{(j+1)^2} \qquad (5)$$

Result 5: The probability that a transaction finds all needed resources free or locked in SH mode at its site of origin, p_1, is given by

$$p_1 = \sum_{k=0}^{\lfloor M/n \rfloor}\frac{\binom{M-nk}{n}}{\binom{M}{n}}\frac{\rho^k e^{-\rho}}{k!} \qquad (6)$$

where
$$\rho = \frac{\lambda_u}{p_2 p_3}\{\bar{t} + p_2[2T + p_3\,\overline{\max_{N-1}}[\tilde{t}]]\} + $$
$$+ (N-1)\lambda_u p_3^{\frac{2-N}{N-1}}\{T + p_3^{\frac{N-2}{N-1}}[T + \overline{\max_{N-1}}[\tilde{t}]]\}$$

Result 6: The probability, p_2 that an update transaction is not rejected after its read phase but before it enters its precommit phase is given by

$$p_2 = (\gamma\bar{t} + 1)^{1-N} \qquad (7)$$

where $\gamma = \lambda_u PC\, p_3^{\frac{2-N}{N-1}}$

Result 7: The probability, p_3, that an update transaction is not rejected during its precommit phase is given by

$$p_3 = \left[\frac{e^{-\alpha T}}{1-(\alpha\bar{t})^2}\right]^{N-1} \qquad (8)$$

where $\alpha = \lambda_u PC\, p_3^{\frac{2-N}{N-1}}$

In order to obtain R_r and R_u, the following procedure would be used.

1. Choose initial values $p_2^{(0)}$ and $p_3^{(0)}$ for the probabilities p_2 and p_3. Recommended values are 0.9999. Set i=0.
2. Compute $\bar{t}$ using $p_2^{(i)}$ and $p_3^{(i)}$ according to Result 1.
3. Compute $p_3^{(i+1)}$ using $p_3^{(i)}$ and $\bar{t}$ according to Result 7.
4. Compute $p_2^{(i+1)}$ using $p_3^{(i+1)}$ and $\bar{t}$ according to Result 6.
5. If $\max\{|(p_2^{(i+1)} - p_2^{(i)})/p_2^{(i+1)}|, |(p_3^{(i+1)} - p_3^{(i)})/p_3^{(i+1)}|\}$ is greater than a given tolerance, then set i=i+1 and go to step 2.

6. Compute $\bar{t}$ using $p_2^{(i+1)}$ and $p_3^{(i+1)}$ according to Result 1.

7. Compute p_1 using $p_3^{(i+1)}$, $p_2^{(i+1)}$ and $\bar{t}$ according to Result 5.

8. Compute R_r using p_1 and $\bar{t}$ according to Result 2.

9. Compute R_u using R_r, $p_2^{(i+1)}$, $p_3^{(i+1)}$ and $\bar{t}$ according to Result 3.

6. Conclusions

Many concurrency control algorithms for DDBs have been proposed in the last few years. In the vast majority of cases, their performance analysis has been limited to counting the number of messages or the number of bits transmitted. In this paper we presented a concurrency control algorithm and the results of a performance evaluation which yielded results such as average response time of read and update transactions and utilization of CPU and I/O devices, as a function of several parameters and as a function of the degree of conflicts between transactions.

The results obtained in the previous section allows us to draw many interesting curves, but due to space limitations, we will concentrate only on four of them.

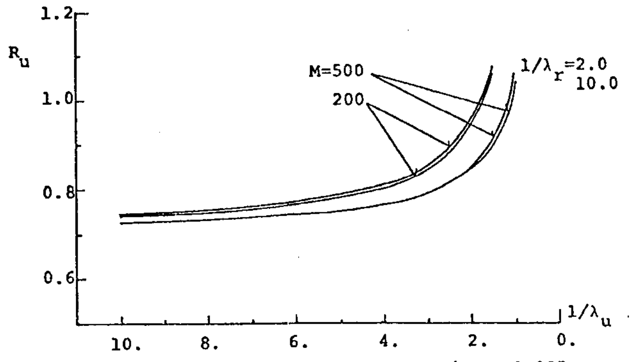
Figure 5 displays the variation of the average response time of update transactions, R_u, as a function of their average arrival rate. Two sets of curves are shown: one for M = 200 and one for M = 500. For each value of M, two values of $1/\lambda_r$ are considered. As it can be seen from the figure, read transactions have little impact on update transactions. Both sets of curves exhibit a similar behavior. In the beginning R_u grows very modestly since the average arrival rate of transactions is sufficiently small so that conflicting transactions interfere very little with one another. After a certain point, the average arrival rate of update transactions is high enough so that the effect of the degree of conflicts between them starts to impact considerably their average response time. This effect is superimposed with that of the increased contention for CPU and IO resources in the computer system of each site. The isolation of these two effects can be seen in Figures 6 and 8 discussed below.

Figure 6 shows the interesting effect of the complexity of update transactions ,p, on their average response time R_u (p is the probability that a transaction leaves the computer system after being served by an I/O device). A decrease in p represents an increase in the numbers of CPU-IO cycles that will have to be performed by each transaction. In other words for a given arrival rate of update transactions, smaller values of p imply higher contention for computer system resources. As one can see, this increase in load has a heavy impact on R_u. Also indicated in Figure 6 is the average response time, R_u^e, of an update transaction which finds the system empty. These values can be interpreted as a lower bound on R_u, since no interference between transactions is considered.

Each one of the three curves of Figure 8 shows the variation of R_u as a function of the size of the transaction, n, for a given value of $1/\lambda_u$. As the size of a transaction increases, the probability of conflicts between transactions also increases. However, this effect can be better observed for higher values of λ_u. For instance, the curve for $1/\lambda_u$= 3.0 grows much faster than that for $1/\lambda_u$= 20.0. It is worth noticing the dramatic effect that the number of DB resources referenced by a transaction has on performance. Consider the curve for $1/\lambda_u$=3.0. For n=10, R_u is 59% higher than its value for n=2, although the number of resources referenced by a transaction grew from 1% to 5% of the total DB resources.
This can be explained by the fact that, for n=2,the probability of conflict, PC, is equal to 0.0199 while for n=10, PC = 0.4085.

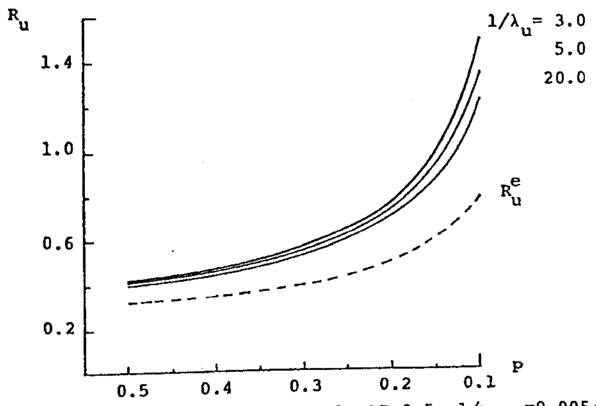
Finally Figure 7 shows the variation of the utilization of the CPU and IO devices with λ_u and λ_r.

In order to render our analysis more manageable we made two simplifying assumptions, namely that the arrival process of any type of message at a given site is Poisson and that the computation time at each node is exponentially distributed. A simulation model was developed to verify the validity of these assumptions. As one can see from table 1 the maximum observed error is of the order of 10% even when the system is heavily loaded. In other words, the analytic model is remarkably accurate.



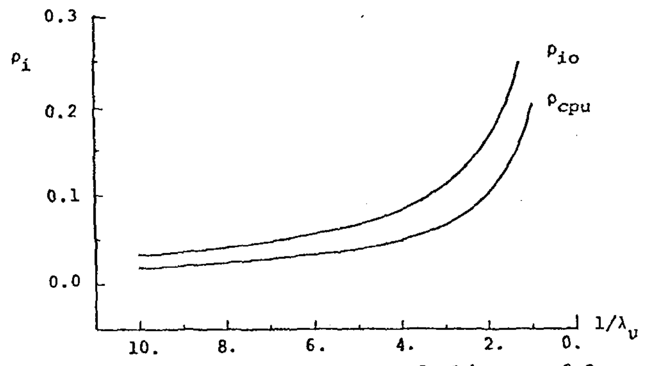
N=6; p=0.2; n=5; D=3; T=0.1; RD=0.5; $1/\mu_{cpu}$=0.005; $1/\mu_{io}$=0.025;

Figure 5 - R_u vs aver.inter arrival time of update trans.



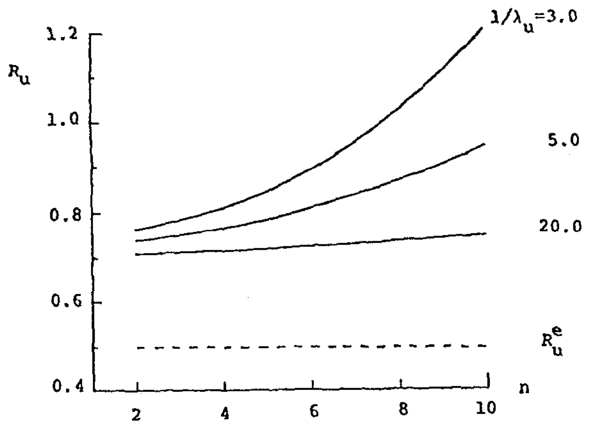
N=6; M=500; n=5; D=3; T=0.1; RD=0.5; $1/\mu_{cpu}$=0.005; $1/\mu_{io}$=0.025; $1/\lambda_r$=5.0

Figure 6 - R_u vs probability p



N=6; M=500; D=3; n=5; T=0.1; RD=0.5; $1/\mu_{cpu}$; p=0.2; $1/\mu_{io}$; $1/\lambda_r=1/\lambda_u$

Figure 7 - Utilizations vs average inter arrival times



N=6; M=200; D=3; T=0.1; RD=0.5; p=0.2; $1/\mu_{cpu}=0.005$; $1/\mu_{io}=0.025$; $1/\lambda_r=5.0$

Figure 8 - R_u vs number of transaction resources

Case	$1/\lambda_r=1/\lambda_u$	R_r sim.	R_r anal.	%Dif.	R_u sim.	R_u anal.	%Dif.
M = 500	10.	.154	.163	-5.8	.689	.718	-4.2
	8.	.156	.167	-7.0	.706	.725	-2.7
	6.	.177	.173	2.3	.739	.737	0.3
	4.	.189	.186	1.6	.774	.761	1.7
	3.	.206	.199	3.4	.807	.788	2.4
	2.	.244	.228	6.6	.869	.845	2.8
	1.5	.280	.262	6.4	.966	.913	5.5
	1.	.377	.348	7.7	1.215	1.087	10.5
M = 200	10.	.173	.176	-1.7	.714	.734	-2.8
	8.	.180	.183	-1.7	.725	.745	-2.8
	6.	.192	.195	-1.6	.767	.764	0.4
	4.	.232	.221	4.7	.843	.805	4.5
	3.	.257	.249	3.1	.895	.850	5.0
	2.	.335	.312	6.9	1.071	.954	10.9

N=6; n=5; D=3; T=0.1; RD=0.5; p=0.2; $1/\mu_{cpu}=0.005$; $1/\mu_{io}=0.025$
Table 1 - Simulation versus Analytic Results.

Acknowledgement

The authors would like to thank Marco Antonio Casanova for the helpful discussions they had and for his comments on a draft of this paper. This research was supported by FINEP. Support from CNPq, Brasil and from IBM do Brasil is gratefully acknowledged.

References

[1] - Ellis, C.A., "A Robust Algorithm for Updating Duplicated Databases", Proceedings 1977, Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, University of California, Berkeley California, May, 1977.

[2] - Thomas, R.H., "A Majority Concensus Approach to Concurrency Control for Multiple Copy Databases", ACM Transactions on Database Systems 5, 2, June 1979.

[3] - Rosenkrantz, D.J., Stearns, R.E. and Lewis P.M. "System Level Concurrency Control for Distributed Database Systems", ACM Transactions on Database Systems, 3, 2, June 1978.

[4] - Alsberg, P.A., Belford, G.G., Day, J.D. and Grapa, E., "Multi-Copy Resiliency Techniques", Center for Advanced Computation, AC Document Nº 202, University of Illinois, May 1976.

[5] - Menascé, D.A., Popek, G.J. and Muntz, R.R., "A Locking Protocol for Resource Coordination in Distributed Databases", ACM TODS, June 1980.

[6] - Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES, IEEE Transactions on Software Engineering, SE-5, 3, May 1979.

[7] - Bernstein, P.A., Shipman, D.W. and Rothnie Jr, J.B., "Concurrency Control in a System for Distributed Database (SDD-1)", ACM Transactions on Database Systems 5,1, Mar 1980.

[8] - Reed, D.P., "Naming and Synchronization a Decentralized Computer System", Ph.D. Thesis, M.I.T. Department of Electrical Engineering, Sep 1978.

[9] - Bernstein, P.A., and Goodman, N., "Timestamp-Based Algorithms for Concurrency Control in Distributed Database Systems", Proc. 6th VLDB Conference, Montreal, Canada, Oct 1980.

[10] - Bernstein, P.A., and Goodman, N., "Fundamental Algorithms for Concurrency Control in Distributed Database Systems", Tech. Rep., Computer Corporation of America, Feb 1980.

[11] - Garcia Molina, H., "Performance of Update Algorithms for Replicated Data in a Distributed Database" Ph.D. Dissertation, Computer Science Department, Stanford Univer-

sity, Jun 1979.

[12] - Ries, D., "The Effects of Concurrency Control on the Performance of a Distributed Data Management System", Proc. 4th Berkeley Conference on Distributed Data Management & Computer Networks, Aug 1979.

[13] - Dantas, J.E.R., "Performance Analysis of Distributed Database Systems" Ph.D. Dissertation, Computer Science Department, University of California, Los Angeles, 1980.

[14] - Menascé, D.A. and Nakanishi, T., "Optimistic versus Pessimistic Concurrency Control Mechanisms in Database Management Systems", Information Systems, 7, 1.

[15] - Hevner, R. and Bing Yao, S., "Query Processing in Distributed Database Systems", IEEE Trans. on Software Engineering SE-5, 3, May 1979.

[16] - Lampson, B. and Sturgis, H., "Crash Recovery in a Distributed Data Storage System", Tech. Report, Computer Science Laboratory, Xerox Palo Alto Research Center, Palo Alto, California 1976.

[17] - Menascé, D.A., and Landes, O., "On the Design of a Reliable Storage Component for Distributed Database Systems", ibid [9].

[18] - Gardarin, G. and Chu W.W., "A Distributed Control Algorithm for Reliably and Consistency Updating Replicated Databases", IEEE Transaction on Computers, C-29, 12, Dec. 1980.

[19] - Lee, H., "Queueing Analysis of Global Locking Synchronization Schemes for Multicopy Databases", IEEE Transaction on Computers, C-29, 5, May 1980.

[20] - Nakanishi, T. and Daniel A. Menascé, "Correctness and Performance Evaluation of a Two-Phase Commit Based Protocol for DDBs, Technical Report, Departamento de Informática, PUC/RJ, Brazil.