Chapter 3

Performance Analysis: Background

3.0 Introduction

As mentioned earlier, a large number of concurrency control algorithms proposed in the literature for synchronizing the transactions. In fact, in a recent paper on concurrency control on DDBS [Bern81], Bernstein and Goodman have shown 48 principles methods. Also, there are a number of algorithms proposed by others which do not fall in the above 48 cases. Some of the algorithms were even incorrect. To bring some order to this confusion, the concept of serializability was introduced to prove the correctness of the algorithms. This helped to eliminate some incorrect algorithms. But in general, the performance for most of these algorithms is not fully understood. Hence, there is a need to know how well these algorithms perform in terms of cost, response time, throughput, and utilization.

Concurrency control in centralized DBS is well understood and two phase locking has been accepted as a standard solution [Eswa76]. But in the case of DDBS, there is no single standard solution [Bern81].

In section 3.1 we give an introduction to performance analysis. In section 3.2 the performance criteria (the most common performance metrics) is discussed. In section 3.3 we survey prior studies of the performance of concurrency control algorithms. These studies include simulation, analytical techniques and a combination of both on centralized and distributed databases. Among the analytical models the most important so far for this thesis concerned is the Mean Value Analysis model for centralized database system by Tay et.al. The data contention model developed in this thesis based on the mean value analysis, which is an extension of Tay et.al's model. This chapter ends with a summary and conclusions given in section 3.4.
3.1 Performance Analysis

Performance evaluation has played an important role in the development of computer systems. The need for performance evaluation and prediction exists from the initial conception of a system’s architectural design to its daily operation after installation. It helps the designer to find out how the system behaves under different load conditions and also helps in identifying potential system bottlenecks.

CCAs effect the performance of the DDBS. The database designer would like to know how the concurrency control effects the performance of the system. Also, there are many algorithms present in the literature. Database designers are faced with the difficult task of choosing the most appropriate CCAs for their needs.

Performance studies and comparisons can be done by Qualitative analysis and Quantitative analysis. In the qualitative methods one mainly compares the message overhead added by the CCAs. Though this yields insight into the behaviour of the algorithm it will not give a complete picture. As a result it is difficult to draw conclusive results from these studies. Quantitative analysis is the main approach used for comparison. There has been extensive work done in this area for both centralized and distributed database CCAs. Performance studies have been carried out by both simulation and analytical work. The major advantage of simulation methods over analytic methods is that they can be used to model the system with great detail involving complex flow control of the system. But their major disadvantage is that they require an enormous amount of effort and resource to gain comparable confidence in the results. This problem is quite acute in concurrency control studies because of the large number of parameters involved and parametric studies of the entire parameter set are not practical. Analytical methods have the advantage of requiring much less time to obtain the solution and also it often yields greater insight into system’s behaviour. But, the major disadvantage of this methods is modeling is more difficult and the solution techniques may become intractable.
3.2 Performance Criteria

CCAs can be judged on the basis of several performance measures. These measures can be divided into two classes: external and internal.

External Measures

These are the measures that can be observed directly by the users of a database. They are indicators of quality of the service provided by a database. The most common external measures are transaction response time (time interval between the instant when the user submitted the transaction at a site and the instant when transaction is completely executed at that site) and the system throughput (rate of transaction completions in a system).

Internal Measures

These are the measures that cannot be directly observed by the users of a database. These measures are indicators of some mechanisms, or conditions that are internal to concurrency control algorithm. They are of great interest to designers of CCAs, because they may help in identifying weak points of an algorithm, or provide a plausible explanation of certain behaviour of an algorithm. They may help in tuning the performance of a system. Some examples of the internal measures are utilization of the system resources (CPU device, I/O device) (fraction of time the system resource is busy), and the number of messages (the average number of messages exchanged to execute a transaction).

Note that external and internal measures strongly influence each other. For example, response time of a transaction depends upon probability of conflict and wait due to the conflicts. Similarly, probability of conflict is a function of response time, because the larger the response time the longer an transaction holds data items resulting in higher probability of conflict.
These performance measures are functions of a number of system parameters like the number of nodes (sites), size of the database (number of logical data items in the entire database), transaction size (number of logical data items accessed by a transaction), communication characteristics (topology, delays) and computer system characteristics (CPU service time, I/O service time, scheduling). Also, performance depends on the database access methods, data distribution etc.

For an accurate assessment of the performance, all these parameters must be considered in the model. Moreover, for a comprehensive study of the performance, each performance measure must be evaluated over the entire space of these parameters.

3.3 Survey

Given the many proposed distributed concurrency control algorithms, a number of researchers have undertaken studies of their performance. Performance analysis has been done using simulation, analytical techniques, and a combination of simulation and analysis on both centralized and distributed databases. We will consider only distributed databases in our performance analysis. However certain findings from centralized database studies apply to distributed databases and some of these studies are included in the survey.

A qualitative study that discussed performance issues for a number of distributed locking and timestamp algorithms was presented in [Bern80,81]. The size and complexity of distributed database systems makes experimental comparison of CCAs almost infeasible, only two experimental studies have been reported in [Kohl85, Noe87]. In [kohl85] the performance of locking was compared with optimistic algorithm. The results were obtained using a lightly loaded two-site testbed system, and were strongly influenced by the fact that data and log records were stored on the same disk. In [Noe87] an implementation of Bayer’s Time Interval concurrency control method and its comparison to the performance of conventional timestamp method was reported. The implementation
was done on the Eden experimental local area network.

Our approach in the succeeding paragraphs will be to discuss the studies that have been done using pure simulation, followed by those of analysis and the ones using a combination and mention whether the study was done on distributed databases, centralized databases or both.

It should be noted that in many of these studies the results are contradictory rather than supportive. As will be noted later, the main reasons for contradiction are the differences in the models themselves.

3.3.1 Simulation studies

Ries

Ries [Ries77,79,79a] developed a simulation model to study concurrency control algorithms that used both static and dynamic locking on both centralized and distributed databases. The model is a closed queueing system with each site having a CPU and an I/O unit. His model considered a large number of input parameters (about twenty) that described the database, the transactions, the site, and the network. The main output parameter was utilization of CPU and I/O.

In [Ries77,79] effects of locking granularity (number of data items in each lock) on performance were reported. These two papers were on centralized database systems and the results indicated that coarse granularity is better if the transaction size is large. But, fine granularity is better if the transactions are small and the lock placement is either random placement or worst case placement. Their conclusions for centralized databases were that dynamic locking allows more throughput than static locking when transactions are short and less throughput when transactions are long. This conclusion has been verified by several others. They assumed that deadlock is not an important factor.

In [Ries79a] Ries compared four concurrency control algorithms, two based on centralized control and two based on distributed control. The centralized methods were
variations of centralized two phase locking. In the first, transactions queued for access to the data items they needed (when locking conflicts occurred), while in the other, transactions were restarted whenever the data items they needed were not available (whenever lock requests were denied). The restart method was found to allow more throughput in almost all cases investigated, particularly when transactions access a large number of data items and required extensive communication among sites. The distributed control methods differed in the way that deadlock was handled. In one, a deadlock detection algorithm was invoked periodically. In the other (which was variation of the "Wound-wait" protocol [Rose78]), deadlock was prevented by ensuring that an "older" transaction would never queue for data items held by a "younger" transaction. Instead the younger transaction would be restarted after releasing all its locks. The deadlock prevention method performed better in most cases. Only when transactions involved many data items and the communication delays were large did deadlock detection perform better.

In the higher level comparison of centralized and distributed control, the simulations done by Ries indicated that distributed control leads to better performance whenever most transaction can be handled locally. If most transactions involve multiple sites, however, centralized control does better unless transactions involve few data items and the communications delays are quite small.

Ries and stonebraker attempted to determine the performance of these algorithms without separating the effects of data contention from resource contention. In efforts conducted by several researches [Lin82,82a, Tay84] it was found that hardware contention clouded the issue of performance due to data contention. Therefore it is important to separate these effects and combine them later so that the performance of the algorithm can be fully evaluated.
Lin and Nolte

Lin and Nolte have described results obtained from a single simulation model in a series of Papers [Lin82,82a, 82b,83]. The concurrency control methods studied in their experiments include a two phase locking method, a time-stamp order method, and a method based on multiversions. As mentioned earlier the number of factors effecting the performance is very large and as a result including all of them is too expensive for simulation. In order to simplify the simulation the system and transactions were modeled at a high functional level. In their model, CPU processing time, I/O processing time and message delay are all effectively combined into one delay which is termed communication delay. This delay was assumed to be either hyper or hypo exponentially distributed. The system considered was a closed multiprogramming system, i.e. the number of transactions running concurrently remains at a constant level. This assumption is equivalent to assuming that there is a constant backlog of transactions a waiting activation, or that the system is being subjected to a load beyond that which it is capable of handling. Consequently, it is not surprising that, in many simulation experiments the system became clogged with restarted update transactions. In simulating two phase locking, whenever a transaction conflicts with other transactions (during lock requests) the transaction is put in a FCFS queue and it will be started when the lock becomes free. Deadlock detection and restarts are assumed to be instantaneous.

Lin and Nolte’s model represents a more abstract model of the system than Ries. Consequently, the range of questions that can be answered accurately with their model is more restricted.

Results from their study [Lin82] indicates that nonuniform accessing of locks is quite similar to uniform access under heavy load. Throughput first increased and then decreased as the multiprogramming level was increased. In their comparative study [Lin83], they found that two phase locking out performed timestamp algorithms for small
transactions whereas the reverse was true for large transactions also they found that no significant increase in performance can be obtained in using multiversion method. In [Lin82b] their results indicate that under the two-phase locking both probability of conflict between transactions and the probability of deadlock increased with the mean communication delay (but were insensitive to the variance of communication delay).

In [Lin82a] the effect of the ratio of read-only to update transactions on performance of distributed databases with Two-phase locking algorithms are discussed. Results indicate that the ratio of read to update transactions has little effect on the probability of conflict, deadlock, and the blocking delay of update lock requests, whereas it has significant effect on the read-only transactions. Some of the results obtained were counterintuitive and the authors attribute these to the system getting saturated with update transactions regardless of ratio of read to update transactions.

Their conclusions contradicted the results of other studies [Care83]. In their results the basic timestamp algorithm outperforms locking, which contradicts the results of [Care83]. As has been pointed out by carey, the main reason is because of the assumption that service times for transactions do not depend on the number of other transactions in the system. Also, in their results they showed that the number of sites need not be a parameter, which reflects the shortcoming of their model rather than a fundamental result. As pointed out by sevcik in [Sevc83], the calculation of communication delay to account for CPU, I/O and message delay, the number of sites, the degree of replication of data and the concurrency method is difficult and they have omitted these factors from their model.

Lin and Nolte [Lin82,82a] extended the work done by Ries and Stonebraker, but did not consider resource contention.
Galler

Galler in his thesis [Gall82] characterized ten concurrency control methods for distributed databases and provided a subjective rating of them with respect to twelve characteristics that describe the system and its workload. He also used simulation to compare a locking method (basic two-phase locking with primary copy locking for updates) and a timestamp method (basic time-stamp order with the Thomas write rule). His results showed that the timestamp method permitted significantly higher throughput than the locking method for a wide range of system description parameter. In particular, as the number of sites increased, the performance advantage of the timestamp method became larger. Overall, the timestamp method was found to be superior whenever the probability of conflict between transactions was not excessive. His conclusions were based on the fact that the probability of deadlock among transactions was not very high. It appeared that the timestamp method did better because it did not occupy data items for long periods of time by locking them.

Galler's simulation model used ten input parameters, including number of sites and number of data items (or granules) at each site. Rather than having a fixed number of transactions continuously active at all sites, Galler specified a fixed population of interactive users, each of which would cycle between a "think time" (a delay sampled from a negative exponential distribution with a specified mean) and an active period in which a transaction they generate is processed. Thus, by varying the mean think time from zero upwards, Galler's model represents system congestion ranging from very heavy (frequent conflicts and restarts) to very light (almost no conflict). In Galler's study the model of resource contention was artificial and the study assumed fully replicated data, extremely small transactions, and very coarse concurrency control granularity.

Thanos, Carlesi and Bertino

Thanos, Carlesi and Bertino used simulation to investigate several algorithms, both
centralized and decentralized, based on two-phase looking in DDBSs [Than81]. They distinguish read locks (which are sharable) and write locks (which are exclusive). Read locks are released as soon as all precommits are acknowledged, while write locks are released implicitly by the commit. This is an important consideration since it is widely held that many databases have a preponderance of reads over updates. Furthermore read locks can be shared by more than one transaction at once thus increasing the level of concurrency.

The parameters of their model include the ratio of queries to updates, degree of replication of the data, the average read-set and write-set sizes, and the probabilities of conflict (overlap between the readset or writeset of one transaction with the writeset of another) between transactions from two classes. Both static and dynamic acquisition of locks were considered. They found that, if dynamic acquisition of locks was used, then decentralized control led to better performance than did centralized control (except when the degree of replication was very low). They made a point of avoiding some assumptions that have been made in other studies for convenience even though they are unrealistic. Some such assumptions not made in this model were (1) all transactions do updates, (2) the database is fully replicated, and (3) all transactions have similar patterns of database read and writes.

In their paper [Than82,83,88] a performance analysis of four CCAs based on two-phase locking has been carried out. Algorithms considered for comparison were decentralized algorithm, primary copy algorithm, majority consensus algorithm and centralized algorithm. They have studied the impact of data replication, lock predeclaration, the time-out parameter, local processing time, the degree of locality of reference, interarrival time and network parameters on the performance of these algorithms. Their simulation model is described in [Than88].
Carey

Carey [Care83, 83a, 84] extended also the simulation model developed by Ries and Stonebraker and used it to compare the performance of different variations of locking, timestamp and serial validation algorithms for centralized database systems. They concluded that if conflicts are rare then all the algorithms perform equally well. But if conflicts are frequent then algorithms that use blocking (queueing transactions until its locks are available) perform better than the algorithms which use restarts, which contradicts the findings of Ries. However, according to Tay [Tay84], a close examination of Cary’s simulations indicate that serious resource contention was present, and there was no apparent attempt to account for that effect. Their results on granularity were similar to that of Ries. Cary also showed that preallocation is superior or equal to two phase locking irrespective of workload or granularity. This conclusion cannot be substantiated without separating the effects of hardware and data contention. Also showed, hierarchical locking improve the performance especially if the transactions were large.

Cary and Livny in their paper [Care88] a simulation analysis is presented for four concurrency control protocols for distributed database systems-distributed 2PL [Gary79], wound-wait [Rose78], basic timestamps [Bern80, 80a], and a distributed optimistic algorithm [Sinh85] - using a detailed model of DDBMS. Each protocol is used in a series of experiments that vary the amount of time required to process a message and the number of active files per site. Within each experiment, the workload and the amount of replication are varied. The simulation model is thorough. The paper mentions 20 parameters for such items as the number of sites, the number of files, and the CPU time required to send or receive a message. They assumed local area network in their simulation experiments, where the actual time on the wire for messages is negligible. The simulation shows that 2PL was superior performer due to its avoidance of transaction restarts. "Optimistic locking" where transactions lock remote copies of data only as they
enter into the commit protocol (at the risk of end-of-transaction deadlocks) was the best performer in replicated databases where messages are costly. Also they concluded that increasing the number of copies had negative effect on the performance due to update costs. In their study to distribution and parallelism, they examined two cases: One where transactions executed serially but sometimes nonlocally, and the other where transactions executed in parallel at several sites. In the nonlocal case, only minor differences were observed compared to strictly local execution. In the parallel case, they observed that parallelism is only beneficial under light load, especially if messages are expensive. In 2PL, an increase in waiting due to lock contention was observed in case of parallel execution.

The simulation has a modular design, allowing different components of the system to be replaced without changing other components. Cary and Livny in their recent paper [Care91] extend their paper [Care88] and compare the same four protocols with optimistic two-phase locking, a protocol designed by them for replicated data. The simulation shows that their protocol performs best in most situations.

**Pun and Belford**

Pun and Belford [Pun86] studied the optimal granularity and degree of multiprogramming for distributed data base system. They used a simulation to model a fully replicated database. Each node in the system was modeled by a CPU, with PS (processor sharing) scheduling and I/O with FCFS scheduling. The communication delay between the nodes were assumed to be constant. The system was modeled as a closed system with a constant degree of multiprogramming. They simulated two phase locking algorithm and studied the optimal granule assignment for "random placement", "worst placement "and "best placement". For static locking policy, they found that for both random placement and worst placement, fine granularity (each data item is a lock) is optimal for small transactions, and coarse granularity is optimal for large transactions. The
choice of the granularity did not depend on the communication delays. They also considered dynamic locking, and in this case it was found that communication delay did effect the choice of granularity. For well placed locking scheme it was found that the optimal always occurred when the granularity was equal to transaction size.

Garcia-Molina

In [Garc77] the performance of a centralized locking algorithm is compared with the distributed voting algorithm. Event driven simulator were used to obtain performance comparisons. At each node, the model consists of two FIFO queues, an I/O server and a CPU server serving these queues. The performance metrics were response time, I/O utilization, and the number of messages. The results indicated that centralized locking algorithms performed considerably better than voting algorithms except in the case of very high I/O utilization.

Devor and Carlson in their paper [Devo82] presented a framework for the design of a locking mechanism, which requires a lock specification policy (defines the granularity(ies) of locable resources, the access modes (e.g.share,exclusive)) and the compatibility of concurrent (mode/resource/requests), a lock acquisition policy (defines when locks are requested (e.g.static, dynamic)) and lock release policy (defines when the locks acquired by a transaction are released). From their simulation they concluded that granularity in general is not an independent variable in lock mechanism design but is driven by the system and resource structure, a dynamic lock acquisition protocol is generally necessary (due to performance improvements which results from fine granularity) and even when releasing shared locks before the end of transaction [LRP2] is more expensive than releasing all locks at end of transaction [LRP3], system reformance may still improve when transaction use LRP2.
Bhargava in [Bhar82] has compared the performance of an optimistic approach with that of a distributed locking algorithm with the wound-wait technique. Messages costs were high, and restart costs were biased by buffering assumptions.

Balter and Decitre in [Balt82] have studied several alternative schemes for handling or preventing deadlock in distributed locking algorithms.

Kiessling and Landherr in [Kies83] have tested the improvement in performance of creating a new version of data when there is an update and keeping the old version for read requests that are already in the system (a variation of multiversion methods.). Their results show that there is less conflict among transactions when two versions are used and seemed to indicate a performance improvement. However their conclusion were contradicted by Peinl and Reuter [Pein83] who studies the same methods. Peinl and Reuter determined that one version method is as good as the two version method. One possible explanation is the choice of performance measures in determining the best method. Kiessling and Landherr based their conclusions on the ratio of total number of conflicts to restarts, whereas Peinl and Reuter used the ratio of executing transactions to their number of restarts. Neither study measured throughput, so that results can not be compared.

OZSU [OZSU85] used a simulation methodology based on Petri nets to analysis two locking-based concurrency control algorithms for DDBSs. The first one is the centralized locking algorithm of Garcia-Molina [Garc79] and the second one is the distributed locking algorithm due to Gardarin and Chu [Gard80]. The results they obtained indicate that these algorithms perform quit similarly under most circumstances with the system model and the assumptions they have employed. Also they obtained that under the circumstances they have simulated, the additional cost of synchronizing actions among sites may not be warranted.
Borges [Borg86] presented a framework of a model for database concurrency control analysis. Through a set of simulation experiments he carried out he concluded that simplification on the model for examples assuming single time delay as in Lin and Nolte [Lin82,83], unlimited CPU and I/O resources and a constant transaction size, may lead to incorrect conclusions over the mechanism performance.

Many of different studies were informative but they contradicted the results of other studies. Agrawal and Carey [Agra87] considered the different assumptions made by different researchers and indicated that assumptions like infinite serves (constant processing time) are not realistic and lead to results which contradict results obtained from studies with dissimilar assumptions. They performed a number of concurrency control performance studies using a queueing network simulator.

Hunag and Chin [Huna88] have proposed a non-two phase and deadlock-free concurrency control method for centralized DBSs, called Ascending order locking protocol (AOLP) Based on the property of ordering granules and compared its performance with that of two methods, 2PL [Eswa76] and tree protocol [Kede79]. They concluded that AOLP method has better performance than 2PL method and the tree protocol method on the evaluated factors of throughput, conflict ratio, and response time. Also, they find that a coarse granule will yield higher throughput when the size of a transaction is large and the distribution of accessed data items is random. This result confirmed the results obtained by Ries [Ries79].

KUMAR and HSU [Kuma91] Evaluated the performance of a new concurrency control method for centralized DBSs, called cautious waiting (CW). Cautious waiting is a nonpreemptive (an active transaction, i.e., a transaction waiting for CPU or I/O is never rolled-back) and deadlock-free Algorithm. Similar to other 2PL algorithms, CW uses transaction rollback and blocking to resolve conflicts among concurrent transactions.
However, its approach differs from others in the sense that in resolving conflicts it takes into considerations the states of conflicting transactions rather than their age. They compared CW algorithm with General waiting (GW) algorithm [Tay85] wound-wait (WW), and wait-die (WD) algorithms [Rose78]. They showed that CW mechanism performs consistently better under a wide range of parameter values than the other mechanisms investigated. They also showed that it is more robust, in the sense that it is less sensitive to load increases and other changes in parameter settings. In contrast, they showed that GW mechanism is not as robust and efficient under stress situations. They showed also, that CW mechanism is flexible in choosing transactions to be aborted. The mechanism tends to regulate the length of the wait-for graph, pruning out a large portion of the wait-for graphs that might develop into long queues, without resorting to excessive aborting. This Result congruent with what was advocated in [Balt82], where it was suggested that wait-for graphs should be limited at most to a length of one.

Rusinkiewicz and Leiss [Rusi91] defined a simulation method of a DDBS and used it to analyze the behaviour of four concurrency control algorithms based on timestamping method. The CCAs are basic timestamp ordering (TSO) using "do nothing" approach, basic TSO with primary copies, the Thomas write rule combined with basic TSO for read-write synchronization, and multiversion TSO. Two-phase commitment was incorporated into all algorithms.

3.3.2 Analytical Studies

In recent years a number of analytic methods have been proposed for studying the performance of concurrency control algorithms. Many of these are based on queueing or Markovian models, other approaches like mean value analysis are also being used. Sevcik [Sevc83] has given a short background on some of these approaches and suggested some general directions in analytical modeling. In this section some of these analysis are discussed.
3.3.2.1 Markovian Models

Shum and Spirakis [Shum81] provide an approach to analysis using Markov Processes. In the paper they analyze the no-waiting method of dynamic locking (transactions that cannot obtain their locks immediately are recycled). However, they assume that the transactions come back as completely different transactions. This is similar to assuming that the particular transaction is lost to the system.

A model based on Markovian processes was proposed by Chesnais, Geleneble and Mitrani in [Ches83]. The vector Markovian Chain representing all the transactions was difficult to solve (computationally intractable). So an approximation was proposed in which processing of a single transaction was represented by a Markov chain and the interaction with other transactions was brought into effect while calculating the probability of conflict. They analyzed the no-waiting case of dynamic locking by making a transaction repeat a conflicting request a fixed number of L times at random intervals. If the lock request was still not fulfilled after L+1 tries, then the transaction was restarted. They achieved the best results when L=0 indicating that pure no-waiting was best. They also generated a new transaction when it was recycled like Shum and Spirakis. The main disadvantage of the model is the complexity of the solution grows as the database size increases and the model itself is rather restrictive.

Mitra and Weinberger in [Mitr84] proposed a detailed model of static locking using Markov processes. They were forced to make a number of simplifying assumptions due to the complexity of the model. The most crucial of these assumptions was to consider transactions that conflicted over data to be lost to the system. While this may be acceptable for telephone traffic analysis, its use for database systems is not clear. In order to relax this assumption they were forced to make a number of approximations.

Griffeth and Miller [Grif85] used a Markovian chain to model four different algorithms,
two of them based on locking, one based on the optimistic approach and the fourth one based on an immediate restart mechanism. Their model is similar to the one proposed by chesnais et.al. Transactions were considered to access a fixed number of granules (r reads and s writes). The major difference was that they considered transactions getting blocked by other transactions. In extending the chesnais model they assumed that the transaction waiting state is a Markov state, which is inconsistent with their own results in their study. The amount of time spent by a process in a Markov state is exponentially distributed. But, their derivation of the waiting time distribution shows that it is not exponentially distributed.

Singhal [Sing91] used a Markovian chain to model basic timestamp ordering algorithm. Singhal has shown that the exact performance model of basic timestamp ordering CCA is so complicated that is impossible to find its closed-from solution. To reduce the complexity of the analysis he analyzed a single transaction in isolation rather than analyzing the whole system and reflected the presence of other transactions on the isolated transaction by the probability that a conflicting access to a data object has been made. This was the main drawback of his analysis.

The above five models are for centralized database systems.

3.3.2.2 Queueing Network Models

A number of studies using queueing network models for database systems have been reported in recent years.

Irani and Lin [Iran79] Proposed a queueing network model (QNM) for analyzing dynamic locking in centralized database systems. The model consisted of a CPU and I/O system with a constant multiprogramming level. Transactions were assumed to request locking uniformly over the entire processing in the system. When a lock request for a
transaction conflicts, then it enters a logical server known as Lock-Wait station. The delay in this station is the sum of the expected total loading for each transaction. It was assumed to be independent of the number of items locked currently and the database size. As pointed out by Thomasian this is one of the main weakness of this model.

Garcia Molina [Garc79a] used both simulation and analysis to investigate the relative performance of centralized and decentralized control for DDBSs. The centralized methods that he investigated were variants of centralized locking (that differed in when updates could be applied once they were received at a site). The decentralized methods were a majority consensus method (resembling the one described by Thomas [Thom 79]) an a ring method (based on that of Ellis [Ell77]). Neither decentralized method exploited the possibility of having one site communicates with many others in parallel.

Garcia-Molina used a simple model of transactions in which readsets and writesets are predeclared, and each transaction consisted only a read phase, a processing phase, and a write phase. He assumed full replication of the data at all the sites. Of the ten (or 50) parameters of the model, performance was found to be most sensitive to the transaction arrival rate, the readset and the writeset sizes, the number of sites, the communication delay between sites, and the time to read a lock or data item.

Garcia-Molina’s results indicated that the centralized methods were generally better than decentralized ones that he investigated (as long as the centralized site was sufficiently powerful to handle its workload). With a large number of sites the decentralized method became impractical since the data was fully replicated and a transaction was processed at only one site at a time.

Garcia-Molina’s analytic technique was iterative in nature. Starting with an assumption of no conflicts, he estimated the time that each transaction would be active. From this he obtained an estimate of the frequency of conflicts between transactions, and an improved estimate of how long a transaction would be active, including delays due to
conflicts. This iterative process converged to mutually consistent estimates of the
transaction active times and the frequency of conflicts. The results from this analysis
agreed well with the corresponding simulation results as long as no site were heavily
loaded.

Potier and Leblanc [Poti80] proposed a model for analyzing static locking for centralized
DBSs. They applied the hierarchical decomposition approach from queueing theory to the
problem of transactions started from on-line terminals. They separated data contention and
hardware resource contention and then used combinatorial analysis for one part (data
contention) and queueing analysis for the resource contention portion. Langer and Shum
[Lang82] contend that they underestimated the probability of conflict of transactions
competing for data, but since no simulation were conducted it is difficult to verify who
is right. Morris and Wong [Morr85] used basically the same model proposed by Potier
and Leblanc and compared their results to simulation with a fairly accuracy. However
their objectives were different. Morris and Wong were comparing the performance of
static locking to optimistic concurrency control. They showed that static locking has
higher throughput, but essentially did not resolve the issue of who was correct in the
Potier-Leblanc vs Langer-Shum dispute.

Thomasian [Thom82] extended the QNM proposed by Irani and Lin. The constant
waiting time assumption in the Lock-Wait station was eliminated by assuming that it is
equal to half the response time (R/2). In the extended model there are D (number of
database items) Pseudo-Servers to represent waiting for locks. Lock request by
transactions are assumed to be uniformly distributed over the processing time. An iterative
method was used for solving the system. There are a number of shortcomings in this
model also. The blocking time calculation is an underestimate because of the assumption
R/2 and it was shown in [Tay84] that this is worse for short transactions. Also, it does
not take into account the possibility that when a transaction gets blocked it may have to wait for more than one transaction.

**Thomason and Ryu** [Thom83] considered static locking in a centralized system. The system was analyzed using a hierarchical decomposition method, where the highest level model yields the user statistics and the lower level yields various transaction probabilities (conflicts). Their model was again based on QNM, but it is structurally different from their previous model.

**Menase and Nakanishi** [Mena82] compared optimistic and (pessimistic) static locking algorithms for centralized systems. The model for the optimistic algorithm was a two level model and an iterative procedure was used for obtaining the desired results, whereas simulation was used for the locking algorithm. In the first level the probability of conflict was assumed constant and the computer system was modeled as a QNM to obtain average time spent by a transaction. In the second phase the computer system was replaced by a set of parallel exponential servers and a Markov chain model was used to derive expressions for the probability of conflict. Locking schemes were found to perform better than optimistic algorithm.

Menasce and Nakanishi [Mena84] did a thorough study of a timestamp based concurrency control method for distributed database systems. They formulated a set of graphical primitives for representing concurrency control methods, and used them to describe a particular timestamp method precisely. From this representation they identified many relationships among model parameters and various model performance measures. This established a set of simultaneous non-linear equations that could be solved iteratively (as was done by Garcia-Molina) Essentially arbitrary estimates were made initially for such quantities as the probability of conflicts between transactions and the utilizations of various system resources. Empirically, the convergence of the iterative solution was not
sensitive to the choice of initial values.

Menasce and Nakanishi's model assumed fixed size of readsets and writesets, full data replication, and constant communication delays between sites. Each site was represented as a central server queueing network model [Lazo84] with both processor and input/output resources.

The model was used to estimate the effect on the performance of changes to the transaction arrival rate, the ratio of queries to updates and the size of readsets and writesets.

Hac [Hac86] proposed a queueing network model for a distributed system with dynamic locking. In this model, the deadlock problem is neglected, and a decomposition method is used.

Ryu and Thomasian [Ryu90] adopted Irani and Lin model and did an excellent job of showing how hardware resource and data contention can be separated in a queueing model. They used standard queueing network decomposition to divide the model into various levels. Their model reaches three major conclusions.

1. Prior to experiencing system thrashing (where throughput declines due to severe contention for resources), performance is affected most by time lost in waiting for locks.

2. Thrashing happens when workload, defined by the following expression is

\[
\text{Workload} = \frac{(\text{Number of lock requests})^2 \times (\text{Number of transaction processing})}{\text{Total Number of data objects}}
\]

3. Granularity affects throughput based on the size of the database. Performance is poor when the database is small and the number of data objects per transactions is large, but tends to improve as the database gets larger until it reaches a certain
size where it begins to decrease again. These results have been independently
verified by Tay and others in [Tay84].

The authors give a detailed description of a Markov chain procedure for
calculation of probabilities at the highest level of the response time model. They describe
extensive validation and compare the results to those of the simulation approach.

Ryu and Thomasian in [Ryu90a] presents analytical models to evaluate two
restart-oriented concurrency control protocols. The model used in evaluating their
performance assumes a closed system with a fixed number of transactions. The system
is solved by formulating a set of equations and solving them iteratively. Simulation is used
to validate the approximate solutions obtained by the analytical models. While the overall
contribution of the paper is quite good, some shortcomings are noticeable. The validation
of the models is not robust; simulation tests were performed for relatively small databases
with moderate degrees of concurrency. The speed of convergence of the iterative
algorithms needs to be addressed, especially at high concurrency levels. The reported
deviation between the analytical solution and simulation at heavy data contention should
also be reported at high concurrency levels and for larger databases.

Both of the above two papers considered dynamic locking in centralized database
system.

Other Models

Queueing analysis of the Le Lann's ticketing algorithm and ordering issues in
distributed database were studied by Kaumon and Kleinoff [Kaum81]. The main focus
of this work was to identify the effect of network delays on the order of updates. The
communication network was modeled as a single queueing system with an infinite server
system with exponential service time. The customers were updates with poisson arrivals,
from all sites. The database manager services the updates in increasing order of ticket
numbers. The goal of developing models for determining the effects of CC algorithms on DDBS performance was not met. The model developed is suited mainly for Le Lann’s algorithm and could not easily be extended to study and compare other methods.

**Cheng and Belford** Compared a reliable centralized scheme (with one backup) with distributed timestamp ordering scheme [Chen80]. The sites were modeled as M/G/1 systems but still considered to behave as a Jackson’s Network. Also, the communication delay was assumed to be constant which will make the results questionable.

Effects of network load and topology on the performance of two phase locking algorithms was considered by **Singhal and et.al.** [Sing 84]. The channels are modeled as M/M/1 queues (with FCFS scheduling). Even though the service rates for the messages were different (multi-class traffic), the network was modeled as a Jackson Network. This assumption results in over estimation of the communication delays. Also, the database modeling part and evaluation of probability of conflict is too simplistic. They concluded that an assumption of constant delay is not appropriate for long haul network.

### 3.3.2.3 Mean Value Analysis

**Tay et.al.** [Tay84,a,b,c,85] Proposed a mean value analysis for centralized database for locking schemes. Tay pointed out some of the difficulties in modeling database using Markovian or queueing models. To avoid these problems they proposed a new scheme. In their scheme the entire analysis is done by using steady state average values and using a flow diagram. A set of algebraic equations can be easily written from the flow diagram using conservation principles. Using their simple model they showed that the model is general enough to model multiple transaction classes, nonuniform access, sharable as well as exclusive locks, static and dynamic locking schemes, transaction with fixed length as well as transactions with variable lengths. In their model, they differentiated between data contention and resource contention (which was not done in earlier works). The model,
though simple, enable many questions that arose in some simulation studies to be answered. It is also showed that data contention can lead to thrashing similar to resource contention in conventional systems.

Though there are many advantages with the above model there are many shortcomings. In their model resource contention was not modeled explicitly Agrawal and Cary [Agra87] showed that neglecting resource sharing of the physical devices in concurrency control models lead to erroneous results.

3.4 Summary and Conclusions

In the performance studies both simulation and analytical studies were reviewed. It was seen that in analytical modeling the problem does not simplify if we model the database as a server, similar to the physical resources of a QNM Almost all the work on distributed database assumed communication delay (i.e. time taken by a massage to travel from one site to another site) to be constant (except [Sing84]), [this assumption is justified in a lightly loaded communication medium. This assumption is supported by a simulation study done by Lin and Nolte [Lin82b] which shows that the standard deviation of communication medium delay has insignificant effect on the performance of concurrency control algorithms], or modeled a communication channel as an infinite server, i.e., the communication channel serves all intersite communication in parallel. As pointed out by cary in [Care83] that such an assumption leads to erroneous conclusions. Analytical work in centralized database systems is more mature than distributed database systems. It is this area, performance of distributes database, that part of our work concentrates. The approach taken by Tay seems to be promising because of its simplicity and versatility. The analytical model developed in this thesis is based on Mean Value Analysis and Queueing network models.

While the distributed concurrency control performance studies to data have been informative, a number of important questions remain unanswered. These include:
(1) How do the performance characteristics of the various basic algorithm classes compare under alternative assumptions about the nature of the database, the work-load, and the computational environment?

(2) How does the distributed nature of transactions affect the behaviour of the various classes of concurrency control algorithms?

(3) How much of a performance penalty must be incurred for synchronization and updates when data is replicated for availability or query performance reasons?

The first of these questions remains unanswered due to shortcomings of past studies that have examined multiple algorithm classes. The most comprehensive of these studies, [Lin83] and [Balt82], suffer from unrealistic modeling assumptions. In particular, contention for physical resources such as CPUs and disks was not captured in their models. As pointed by Agrawal and Cary [Agra87] neglecting to model resources can drastically change the conclusions reached.

The second question above remains open since a number of previous studies have modeled transactions as executing at a single site, making remote data access requests as needed (e.g. [Balt82, Gall82, Lin83]; few studies have carefully considered distributed transaction structure. Finally, the third question remains open since previous studies have commonly assumed either no replication (as in [Lin83, Balt82]) or full replication (as in [Garc79, 79a, Gall82, Mena84, Pun86]), and in some of these studies their simplified models of transaction execution have often ignored important related overheads such as that of the commit protocol.

In our study we will address only a subset of the open questions, we feel that our results constitute an important step towards understanding distributed concurrency control algorithms.
Finally this survey lends credence to the claim by many that no single unified approach to analyzing the performance of various concurrency control methods exists. To adequately compare all algorithms it is likely that a collection of techniques will have to be used.