Using Delayed Commitment in Locking Protocols for Real-Time Databases*

D. Agrawal A. El Abbadi R. Jeffers
Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

In this paper, we propose locking protocols that are useful for real-time databases. Our approach is motivated from two main observations. First, locking protocols are widely accepted and used in most database systems. Second, in real-time databases it has been shown that the blocking behavior of transactions in locking protocols results in performance degradation. We use a new relationship between locks called ordered sharing to eliminate blocking that arises in the traditional locking protocols. Ordered sharing eliminates blocking of read and write operations but may result in delayed commitment. Since in real-time databases, timeliness and not response time is the crucial factor, our protocols exploit this delay to allow transactions to execute within the slacks of delayed transactions. We compare the performance of the proposed protocols with the two phase locking protocol for real-time databases. Our experiments indicate that the proposed protocols significantly reduce the percentage of missed deadlines in the system for a variety of workloads.

1 Introduction

Databases are being increasingly used for a wide spectrum of applications and many of these applications impose different and often conflicting demands on the underlying system. One such example involves using databases for real-time applications, referred to as real-time database systems. Transactions in real-time databases have a timing constraint associated with them. In general, this constraint is expressed in the form of a *deadline* which indicates that transactions must be completed before a certain time in the future. In contrast to traditional databases where the primary goal is to minimize the response time of user transactions and maximize throughput, the main objective in real-time databases is to ensure that transactions meet their deadlines and to minimize the percentage of transactions that miss deadlines in the system. In databases, there are two aspects to the scheduling of transactions: concurrency control for serializing the execution of transactions and CPU and I/O scheduling for the execution of read and write operations. In this paper, we concentrate on the transaction scheduling aspects for concurrency control in real-time databases. The issue of physical resource (CPU and I/O) scheduling has been dealt with extensively elsewhere [1, 9, 16, 2].

Most commercial database systems use the two phase locking protocol [11] for concurrency control. The two phase locking protocol is preferred over other methods for concurrency control due to its simplicity and ease of implementation. Another factor that influences this choice is the well-understood integration of locking with a wide variety of recovery mechanisms [13, 20]. Unfortunately, the blocking behaviour of locking protocols can greatly degrade the performance of real-time database systems. Sha *et al.* [21] proposed a locking-based protocol that avoids the blocking of high priority transactions for at most the duration of a single embedded transaction. Lin and Son [19] have proposed an optimistic priority-based locking mechanism that dynamically adjusts the serialization order of active transactions in order to reduce blocking. Recent performance studies [15, 14] have shown that some variants of the optimistic protocol [17] outperform two phase locking in real-time databases where transactions have *firm* deadlines, i.e., transactions have no value past their deadlines [1, 15]. The authors point out that transaction blocking in the two phase locking protocol results in unpredictable delays causing transactions to miss their deadlines. In contrast, transactions in the proposed optimistic protocols neither block

*This research is supported by the NSF under grant numbers IRI-9004998 and IRI-9117904.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
1992 ACM SIGMOD - 6/92/CA, USA
© 1992 ACM 0-89791-522-4/92/0005/0104...$1.50

nor suffer from wasted restarts. In [14], the authors analyze several variants of the optimistic approach and present simulation results that indicate that under conditions of low data contention, delaying the validation of low priority transactions results in improved performance. On the other hand, under conditions of high data contention, they use a simple 50% rule that requires a validating transaction to wait only while half or more of its conflict set consists of higher priority transactions. Huang *et al.* [16] developed locking variants of the optimistic concurrency control protocol and compared its performance with the class of two phase locking protocols for real-time databases. Some of their results do not completely agree with the simulation results in [15, 14] which may be due to the differences in the simulation models and in the physical implementation schemes. However, both studies indicate that transaction blocking is the main disadvantage in adapting two phase locking to real-time databases.

The research presented in this paper is greatly influenced by the above mentioned studies and is directly motivated by the following two facts: the popularity of the locking approach in most database systems, and the potential of attaining superior performance with optimistic or non-blocking concurrency control protocols in real-time databases. We propose to use a new variant of the locking approach, referred to as ordered sharing [3], in real-time databases with firm deadlines. Ordered sharing can be used to eliminate blocking of read and write operations. However, transactions may be subject to delay at commitment. In traditional databases, this delay could potentially result in poor response time for transactions. However in real-time databases, timeliness in meeting a transaction's deadline, and not response time, is the crucial factor. We can exploit the slack of a delayed transaction to complete the execution of any transactions causing the delay. In order to commit, a delayed transaction that reaches its deadline may have to abort a lower priority transaction that has not yet completed. To summarize, our approach is to exploit any available slack in a transaction to improve the overall performance of the system by decreasing the number of transactions that miss their deadlines.

2 Locking Protocols for Real-time Databases

A *database* is a collection of *objects*. Users interact with the database by invoking *transactions*. A transaction is a sequence of read and write operations that are executed atomically on the objects. The execution of a transaction must be *atomic*, i.e., a transaction either *commits* or *aborts*. Finally, a transaction is guaranteed to be *correct*, i.e., it maps the database from one consistent state to another consistent state. The execution of a set of transactions is modeled by a structure called a *history*. A history is *correct* if it is serializable [8]. All protocols considered in this paper ensure serializability. In order to ensure that aborting a transaction does not result in cascading aborts we require transactions to read only committed values. Executions that satisfy this requirement are said to *avoid cascading aborts* (ACA) [12].

Many widely used concurrency control protocols use *locking* as a basic primitive for synchronization. Traditionally, there are two types of relationships between locks: shared and non-shared. Transactions in real-time databases have *deadlines* associated with them which can be used to assign priorities to transactions. In this paper, we study concurrency control protocols for real-time databases with firm deadlines. A transaction with an earlier deadline is considered to have a higher priority than a transaction with a later deadline. Strict two-phase locking [11] is the most widely accepted concurrency control protocol. However, in real-time databases, this protocol is augmented with a high priority conflict resolution scheme to ensure that high priority transactions are not blocked by low priority transactions and is referred to as 2PL-HP [1, 15]. The protocol can summarized as follows:

1. A transaction T_i must obtain read (write) locks before executing read (write) operations. If T_i's lock has a non-shared relationship with locks held by any transaction T_j and if all such transactions have a lower priority than T_i, they are aborted and T_i can acquire its lock. Otherwise, T_i is blocked until the locks are released by the higher priority transactions.
2. Transactions release all their locks at commitment.

When locks with two types of relationships, shared and non-shared, are used there are three types of blocking that can occur in the system: *read-write blocking* occurs when a transaction holds a read lock on an object and a lower priority transaction requests a write lock on the same object; *write-read blocking* and *write-write blocking* can be defined similarly. In 2PL-HP, a blocked transaction has to wait until all higher priority transactions holding conflicting locks commit.

Recently in [3], a new locking primitive was introduced that allows a new relationship, referred to as *ordered sharing*. Ordered sharing can be used with two phase locking to eliminate the three types of blocking described above. For example, in order to eliminate read-write blocking, a transaction T_j can be granted a write lock on an object even if a transaction T_i holds a read lock on the same object. We say that there is an *ordered shared relationship* from T_i's read lock to T_j's write lock. To ensure serializability strict two phase locking with ordered sharing must observe the following rule:

Ordered Sharing Rule : If T_j acquires a lock with an ordered shared relationship with respect to a lock held by another transaction T_i, the corresponding opera-

tion of T_j must be executed after that of T_i. Furthermore, T_j cannot commit until T_i terminates (i.e., commits or aborts).

If ordered sharing is used to eliminate blocking, transactions may be delayed at commit. However, this delay does not block other transactions from executing read and write operations.

We now describe the two phase locking protocol with ordered sharing (2PL-OS) adapted for real-time databases. In particular, we assume that we have shared relationships between read locks and ordered shared relationships between the remaining three types of conflicts between locks. Also, the *update-in-place* policy is used to execute operations [13]. The protocol can be summarized as follows:

1. Transactions acquire locks before executing operations and release all their locks at commit (abort).
2. When a transaction T is ready to commit it waits until either the ordered sharing rule is satisfied or it reaches its deadline. In the former case it commits whereas in the latter case it tries to commit by aborting all preceding transactions with which it has an ordered shared relationship.
3. When a transaction T aborts, it releases all its locks and causes the abort of all transactions which read values written by T.

This protocol is useful for real-time databases since it does not block any read and write operations from executing. There is a possibility of delay when transactions commit but this does not result in performance degradation, since in real-time databases timeliness of transactions is of greater value than the response time of transactions. Furthermore, this delay can be exploited to execute other transactions within a delayed transaction's slack. Unfortunately, 2PL-OS suffers from the undesirable phenomenon of cascading aborts. We have argued elsewhere that locking protocols that suffer from cascading aborts have poor performance [5]. In the following we develop two variants of 2PL-OS that do not suffer from the problem of cascading aborts.

In the first variant, cascading aborts are avoided by retaining write-read blocking, i.e., ordered sharing is not allowed from write locks to read locks. Hence, a non-shared relationship from write locks to read locks is used in this protocol which is referred to ACA 2PL-OS. ACA 2PL-OS is a hybrid of 2PL-HP when locks with non-shared relationships are used and 2PL-OS when locks with ordered shared relationships are used. In particular, when a transaction tries to acquire a read lock on an object and all writers have lower priorities, they are aborted and the transaction can acquire a read lock; otherwise, the transaction is blocked. For the other two types of conflicts, i.e., read-write and write-write, locks with ordered sharing are used and transactions must adhere to the ordered sharing rule. Since write locks are held until commit, transactions cannot read uncommitted data. Hence all executions resulting from this protocol avoid cascading aborts.

The second variant avoids cascading aborts in 2PL-OS by exploiting the before-images of objects and is referred to as the two phase locking protocol with ordered sharing and before-images, 2PL-OS/BI. Our approach is similar to the one used by the multi-version two phase locking protocols [7, 22]. When a transaction T_i executes a write operation on an object x, it creates a new (uncommitted) value for x. The original (committed) value of x is referred to as the *before-image* of x. When T_i commits, the before-image is discarded and the new value becomes the committed value of x. If T_i aborts, the before-image of x is used to restore x to its prior state. Since the protocol allows multiple writers to execute concurrently, the state of the object is represented by a single committed version and several uncommitted versions corresponding to the different values written by each uncommitted transaction. When a transaction reads an object, instead of reading the value written by an uncommitted transaction, it always reads the current committed version of the object, thus avoiding the possibility of cascading aborts. Hence, there is a reversal of the ordered shared relationship between the two transactions. In particular, write-read blocking is eliminated by allowing the reader to read committed data and requiring an ordered shared relationship from the read lock to the write lock. Hence, the reader must commit before the writer as mandated by the ordered sharing rule. This non-restrictive 2PL-OS protocol uses ordered sharing to eliminate write-read blocking and uses before-images to avoid cascading aborts. 2PL-OS/BI has all the desirable properties of 2PL-OS especially the property of allowing other transactions to execute within the slack of a committing transaction.

2PL-OS/BI minimizes the problem of *wasted* restrarts and eliminates the problem of *mutual* restarts [14]. A wasted restart occurs when an executing transaction is aborted by another transaction that later misses its deadline. In 2PL-OS/BI, restarts may occur due to deadlocks or due to the aborts caused by the forced commitment. Since only committing transactions can cause restarts of other transactions in 2PL-OS/BI all such restarts are useful. In [4], it is shown that the wasted restarts in 2PL-OS/BI due to deadlocks are insignificant. ACA 2PL-OS, on the other hand, may suffer from wasted restarts since a reader may abort a lower priority writer and later the reader itself aborts. Since priorities are assigned when transactions are created and do not change during their lifetimes, our protocols do not suffer from the problem of mutual restarts.

3 The Simulation Model

In order to evaluate the performance of the proposed locking protocols for real-time databases, a database simulation model was developed based on [10, 6, 15]. This simulation model was developed using the SIMSCRIPT II.5 language [18] and implements a centralized database. It is divided into three main components: a Transaction Manager (TM), a Concurrency Control Agent (CCA), and a Data Manager (DM). The TM is responsible for issuing lock requests, the CCA schedules these requests according to the specifications of the protocol, and the DM is responsible for granting access to the physical data objects.

The logical simulation model, shown in Figure 1(a), represents a closed queuing model of a single-site database system. Each transaction has an associated deadline that is calculated as [15]:

$$deadline = txn_start_time +$$

$$(slack_factor * estimated_total_txn_time)$$

Slack_factor is an input parameter that controls the tightness or looseness of the deadlines and *estimated_total_txn_time* is the estimated total service time for the transaction.

The CCA processes lock requests received through the *cc_agent_queue*. Delayed lock requests will be rescheduled once the conflicting operations have released their locks. When a lock request can be granted, an acknowledgement is sent back to the TM which then forwards the database operation to the DM. The use of deadlines to schedule operations prevents the formation of deadlocks in 2PL-HP. Unfortunately, deadlocks may form in both ACA 2PL-OS and 2PL-OS/BI. For this reason, we use a deadlock detection strategy based on wait-for graphs. Whenever a deadlock is detected, the transaction with the latest deadline is chosen as the victim and aborted. We use this approach since the transaction with the latest deadline has the lowest priority among the deadlocked transactions.

The physical queuing model, shown in Figure 1(b), is very similar to the one used in [10, 6] in which the parameters *num_cpus* and *num_disks* specify the number of CPU servers and the number of I/O servers. CPU and I/O scheduling is based on transaction priorities. The parameters *cpu_time* and *io_time* represent the amount of CPU and I/O time associated with reading or writing a data object. Both of these parameters are modeled as uniform distributions. The parameter *cc_req_time* represents the amount of CPU time associated with servicing a concurrency control request (lock request), which is assumed to be a constant.

A form of the batch means method was used for the statistical analysis. Each simulation consisted of a minimum of 3 repetitions, each consisting of 2000 seconds of simulation time. The first 200 seconds of each repetition were discarded in order to let the system stabilize after initial transient conditions. In general we achieved 90 percent confidence intervals for our results[1]. Table 1 summarizes the parameters and their values used in the simulation model and experiments.

4 Experiment Results

In this section, we present and analyze the results of the simulation experiments for protocols 2PL-HP, ACA 2PL-OS, and 2PL-OS/BI. Our analysis is similar to the one reported by Haritsa, Carey, and Livny [15]. We feel that this approach simplifies understanding empirical results reported in different simulation studies. In each of the following experiments, the number of terminals, *num_terms*, is varied to include multiprogramming levels that we consider reasonable for actual database systems. This provides a wide range of operating conditions with respect to data contention (lock conflict) and resource contention (waiting for CPUs and disks). In order to evaluate the effect that resources have on the system, one CPU resource and two disk resources were chosen to represent one *resource unit* [6]. Resource related experiments were performed by varying the number of resource units rather than by individually varying the number of CPUs or the number of disks. By using this combination of CPUs and disks, resource utilization in the system turns out to be slightly I/O bound (disk utilization is slightly higher than CPU utilization) [6].

4.1 Effect of Multiprogramming Level

We first evaluate the effect of varying the multiprogramming level on the performance characteristics of the three locking protocols for real-time databases. The experiment is based on a system with four resource units where transactions have a slack factor of three. We assume that transactions are only aware of their deadlines but do not know of their exact requirements in terms of CPU and I/O time, i.e., the *not tardy* approach is used for executing transactions. The *not tardy* policy is in effect when the TM can only determine missed deadlines when they expire [1]. We refer to these settings as the *baseline* experiment.

Figure 2 illustrates the throughput, miss percentage, number of restarts and disk utilization in the three protocols. We also plot the average arrival rate of transactions in the throughput graph of Figure 2. The throughput graph illustrates that 2PL-OS/BI, which does not incur any blocking, has the best performance among all three protocols. In particular, the maximum throughput for 2PL-OS/BI is 6.75 in comparison to 4.6 of 2PL-HP. This represents a 47% improvement over 2PL-HP. The maximum throughput for

[1] We did not generate the excessive number of repetitions that would have been necessary to get 90 percent confidence intervals for very small values (e.g. miss percentages below 3%).

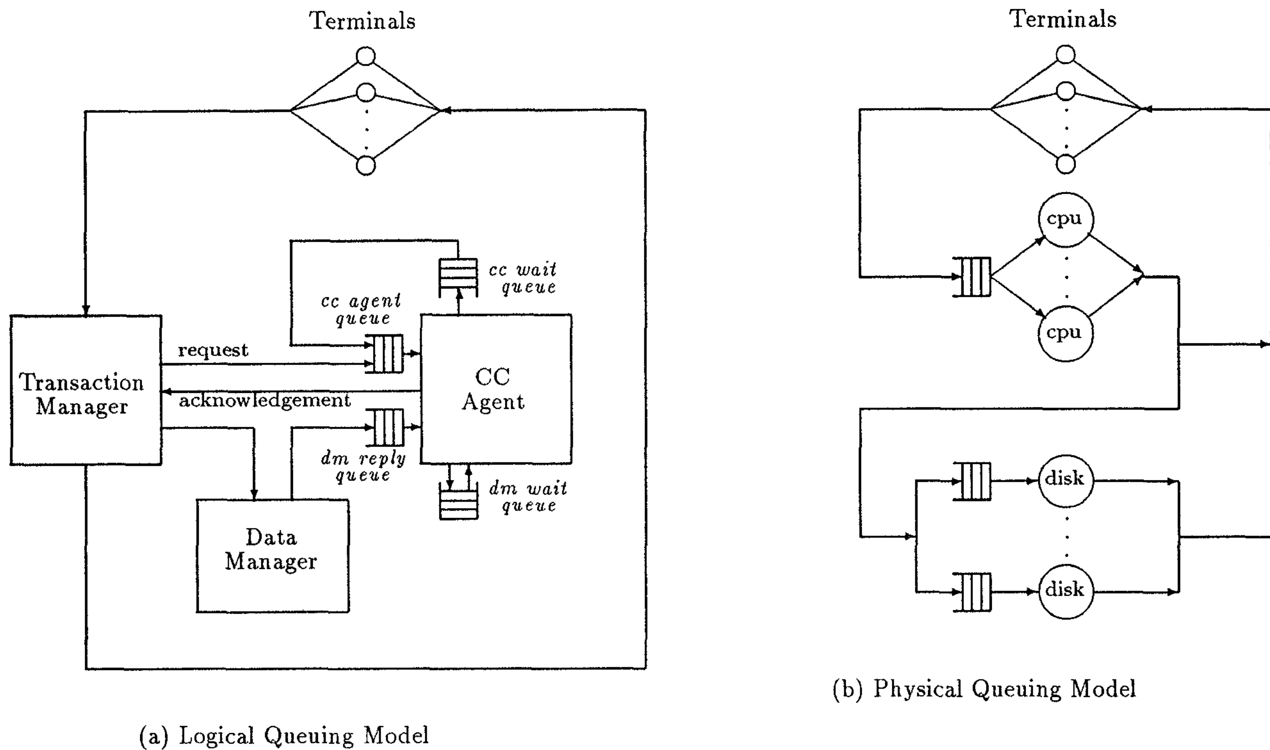
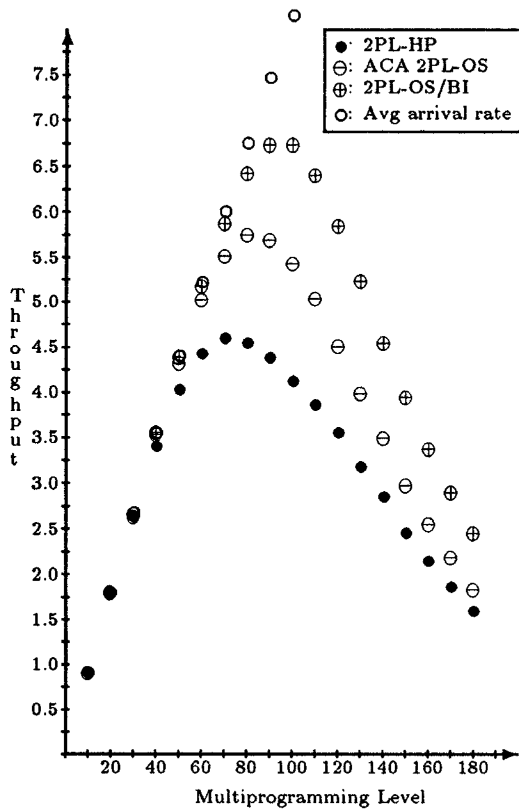
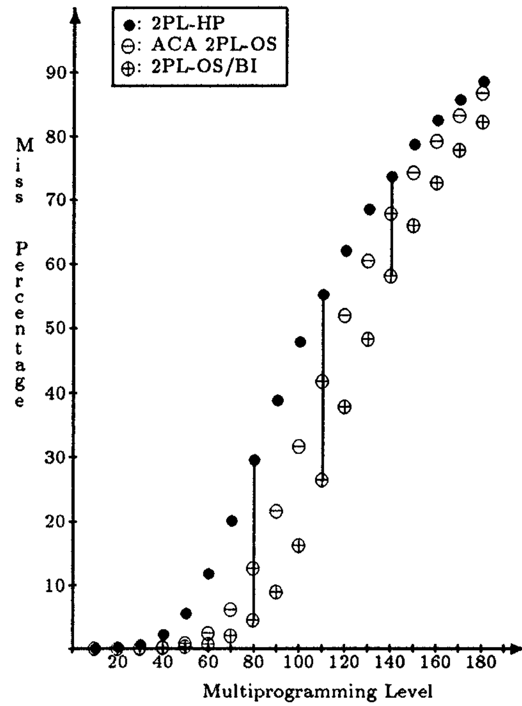


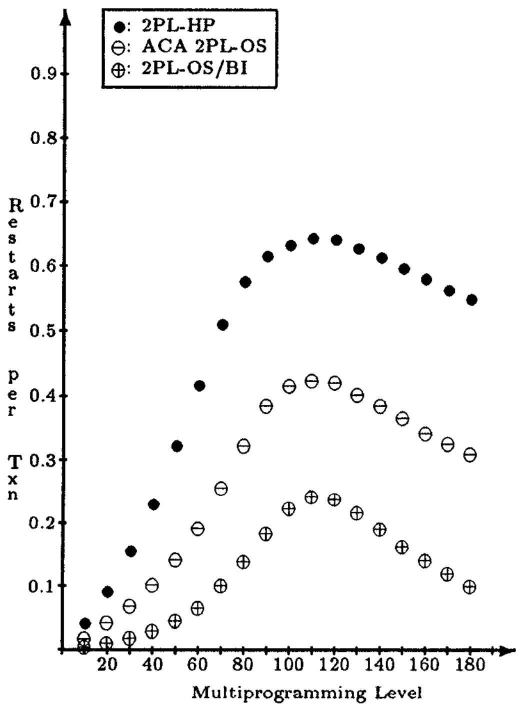
Figure 1: The Simulation Model

Parameter	Description	Value
db_size	Number of objects in database	1000 objects
num_terms	Number of terminals	10 to 180 terminals
num_cpus	Number of cpus	k CPUS
num_disks	Number of disks	$2k$ Disks
txn_size	Mean transaction size	20 operations
update_txn_pct	Update transaction percentage	60%
write_op_pct	Write operation percentage	50%
inter_arrival_delay	Transaction inter-arrival delay	10 seconds
cpu_time	CPU time for accessing an object	12 milliseconds
io_time	I/O time for accessing an object	35 milliseconds
cc_req_time	CPU time for servicing a cc req.	3 milliseconds
slack_factor	Slack Factor	1-9

Table 1: Simulation Model Parameter Definitions and Values







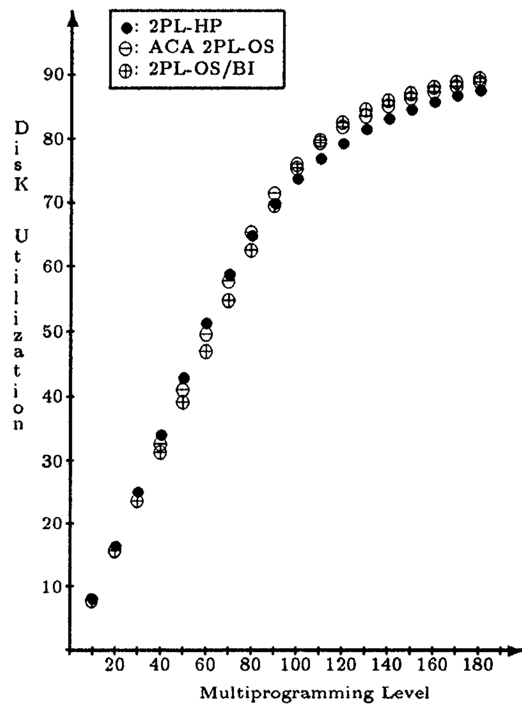


Figure 2: Baseline Case: slack=3, resource units=4, not tardy policy

ACA 2PL-OS is approximately 5.75 (a 25% improvement). The thrash points for 2PL-HP, ACA 2PL-OS, and 2PL-OS/BI are 75, 83, and 95, respectively. After the thrash points, as the multiprogramming level is increased the three protocols converge indicating that in our protocols aborts due to deadlocks dominate the system and the effect of "blocking" versus "non-blocking" is marginalized.

The differences between the three protocols become more obvious when we examine the miss percentage graph in Figure 2. In particular, the rate of increase of transactions missing their deadlines is initially much sharper for 2PL-HP than it is for 2PL-OS/BI. In particular, at the point when 2PL-HP exhibits its maximum throughput (75 terminals), 25% of the transactions are already missing their deadlines. In contrast, only 3% of the transactions miss their deadlines in 2PL-OS/BI and 9% in ACA 2PL-OS. At the thrash point of 2PL-OS/BI (95 terminals), the percentage of transactions that miss their deadlines is 12.5% for 2PL-OS/BI, 27% for ACA 2PL-OS, and 44% for 2PL-HP. The miss percentages are indistinguishable in the three protocols at low multiprogramming level or equivalently at low data contention (10 to 30 terminals). However, at medium to high data contention (more that 40 terminals), 2PL-OS/BI misses significantly fewer transaction deadlines than 2PL-HP. At very high multiprogramming levels, the vast majority of transactions miss their deadlines under any protocol.

Note that similar to OPT-BC (optimistic broadcast [15]), only a committing transaction can generate restarts in 2PL-OS/BI. In 2PL-HP and ACA 2PL-OS a transaction causing a restart may later abort. Thus, there is an increased likelihood of wasted restarts in protocols with blocking and this explains the superior performance of 2PL-OS/BI. In particular, Figure 2 illustrates the number of restarts per transaction in the three protocols and demonstrates that in general 2PL-OS/BI has fewer restarts than the other two protocols. This can be explained by noting that 2PL-OS/BI, and to a lesser extent ACA 2PL-OS, restarts transactions only when it is absolutely necessary, i.e., transactions allow others to run in their slack. Finally, the disk utilization for the three protocols is illustrated in Figure 2. All three protocols have similar disk utilizations. In particular, the disk utilization in 2PL-HP remains high in spite of transaction blocking since the high priority rule restarts transactions resulting in continuous disk activity. However, as pointed out by [15], since 2PL-HP has wasted restarts much of the disk utilization is useless in a real-time system with 2PL-HP when data contention is high.

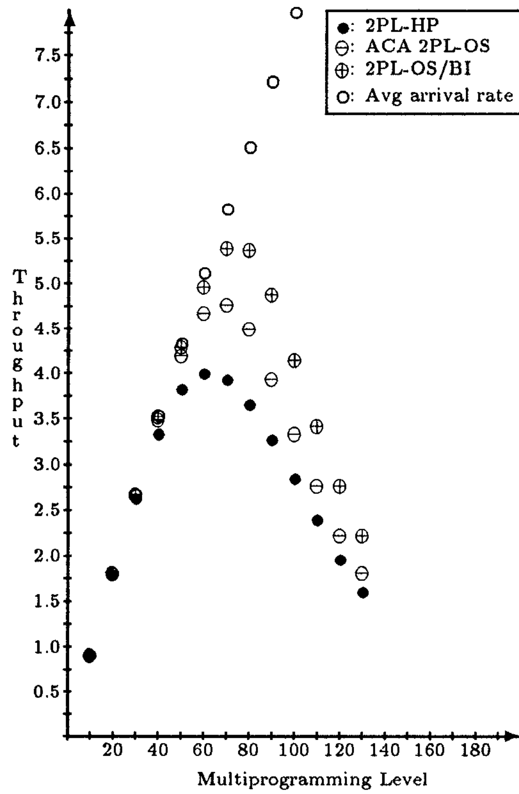
4.2 Effect of Resources

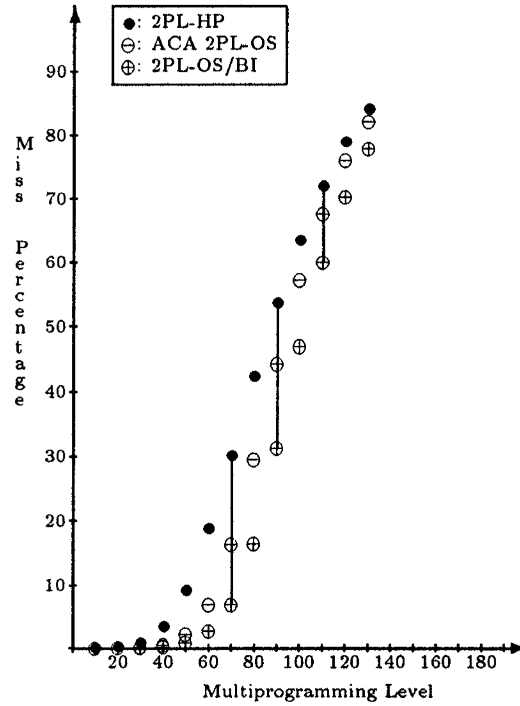
Figure 3 reports the throughputs and miss percentages in the three protocols for a variable number of resources with the remaining parameters the same as in the baseline experiment. The experiments were conducted with three and five resource units. When the number of resources is reduced from four to three, we observe that the relative improvements due to ordered sharing is less pronounced (see Figure 3). This is due to increased resource contention in the system. When the number of resources is increased from four to five, the throughputs and miss percentages in protocols 2PL-OS/BI and ACA 2PL-OS are significantly better than in 2PL-HP. In the case of five resources, 2PL-OS/BI and ACA 2PL-OS are able to meet many more deadlines than 2PL-HP. For example, 2PL-OS/BI does not miss any deadlines up to the multiprogramming level of 60 terminals where 2PL-HP misses 10% of the deadlines. Similarly, ACA 2PL-OS does not miss any deadlines up to 50 terminals. Once again this is due to the desirable property of protocols 2PL-OS/BI and ACA 2PL-OS which eliminates blocking at the expense of possible delays in transaction commitment. Hence, protocols with ordered sharing are particularly useful when there is less contention for resources in the system.

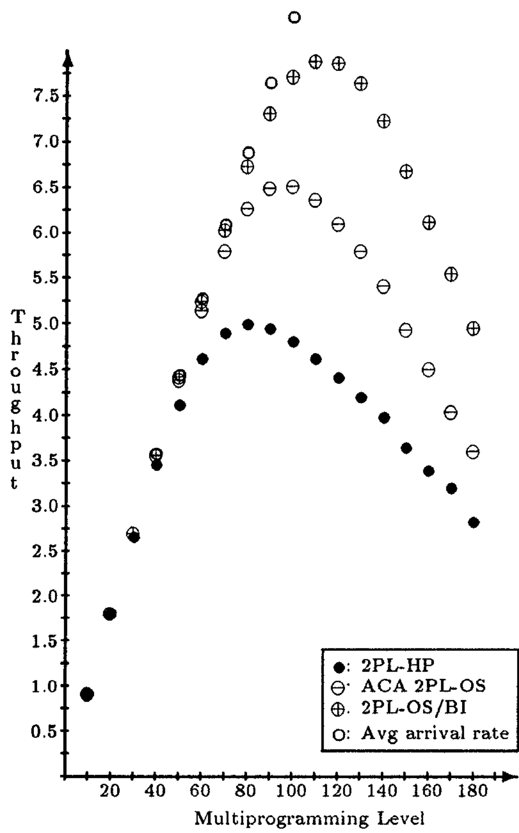
4.3 Effect of Slack

In this experiment, we evaluate the effects of varying the slack factor on the performance of the three protocols. In particular, the slack factor was varied from 1 to 9, and the experiment was conducted with four resource units and at two multiprogramming levels. We chose 75 terminals which is the thrash point for 2PL-HP and 95 terminals which is the thrash point for 2PL-OS/BI. The results of this experiment are illustrated in Figure 4.

Figure 4 shows that when deadlines are very tight, that is when the slack factor is close to one, the performance of the three protocols is indistinguishable. As the slack in deadlines is increased, protocols with ordered sharing start demonstrating superior performance. In general, both 2PL-OS/BI and ACA 2PL-OS miss significantly fewer deadlines than 2PL-HP beyond the slack factor of 1.5. For example, with 75 terminals and slack factor 4, 2PL-OS/BI has almost no transactions missing their deadlines while 2PL-HP has about a 16% miss rate. As the slack is increased all protocols asymptotically reach a miss percentage of zero. At the thrash point of 2PL-OS/BI, which is a heavy load for 2PL-HP, the miss percentage in 2PL-OS/BI becomes insignificant at about a slack factor of 6. 2PL-HP is still missing 24% of its transactions under the same conditions. At this multiprogramming level 2PL-HP is saturating the resources in the system. The improvement in performance of protocols with ordered sharing can be explained as being due to the execution of transactions during the potential delay of other transactions' commitment. In particular, when the slack factor is greater than 1.5, we notice that 2PL-OS/BI, and to a lesser extent ACA 2PL-OS, miss significantly fewer deadlines than 2PL-HP.







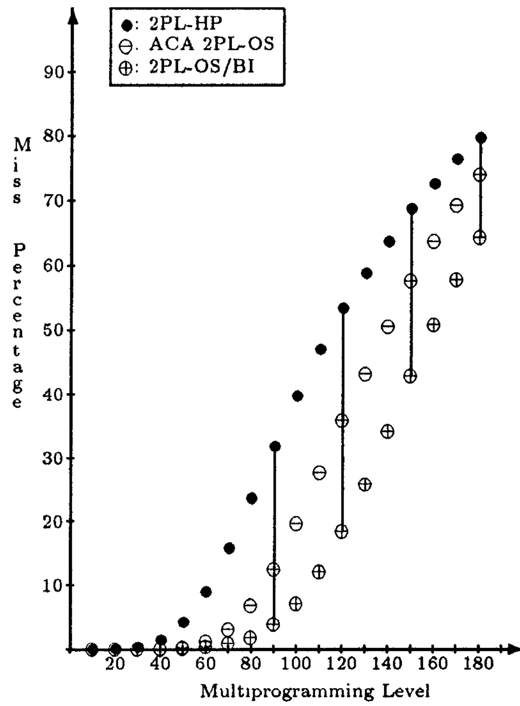


Figure 3: slack=3, resource units =3(top) =5(bottom), not tardy policy

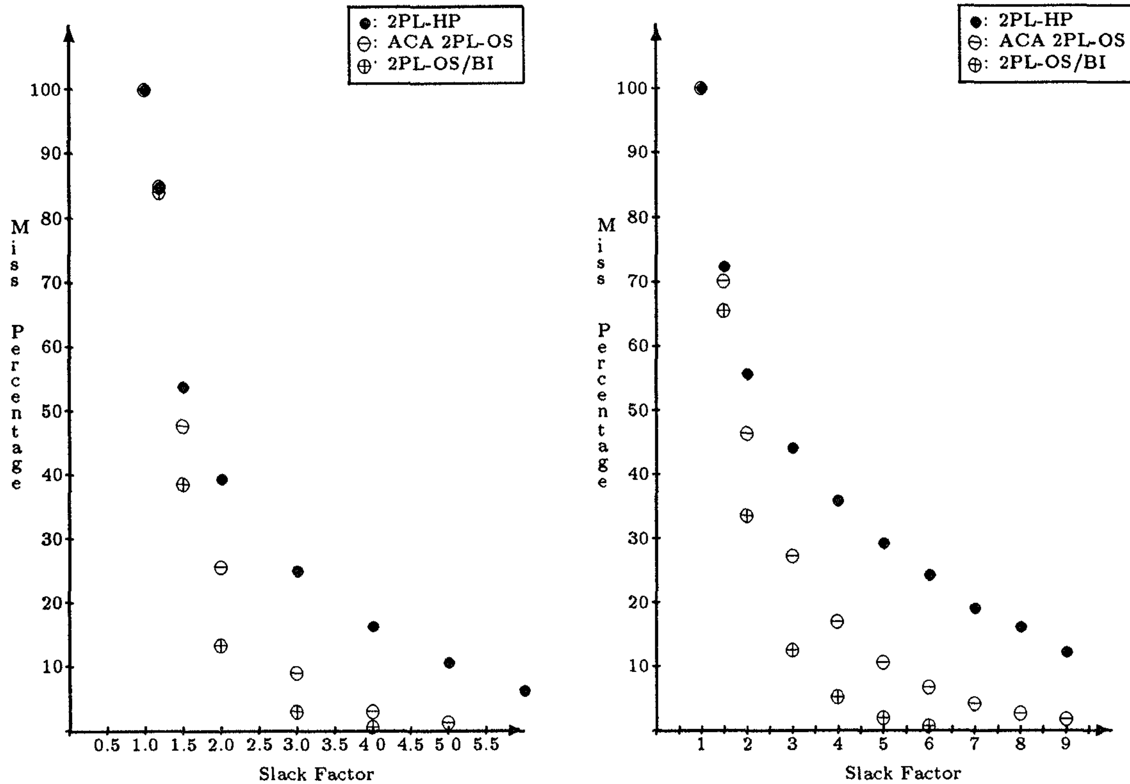


Figure 4: Varying slack factor, 75 terminals(left), 95 terminals(right)

5 Conclusion

In this paper, we described locking based protocols for real-time database systems. Our approach is motivated by two main observations. First, locking protocols are widely accepted and used in most database systems. Second, in real-time databases it has been shown that the blocking behavior of transactions in locking protocols results in performance degradation. We proposed using a new relationship between locks called *ordered sharing* to eliminate blocking. Ordered sharing has the desirable property of eliminating blocking of read and write operations at the expense of a possible delay at transaction commitment. Unlike conventional databases where this delay may degrade response time, in real-time databases we exploit this delay by allowing other transactions to run within the slacks of delayed transactions. Eliminating all types of blocking through ordered sharing may, however, result in cascading aborts. We overcome this problem by using before-images, which are generally maintained for recovery purposes. Based on these ideas, we described two protocols ACA 2PL-OS and 2PL-OS/BI that are suitable for real-time databases.

We compared the performance of the two proposed protocols with 2PL-HP, the two phase locking protocol for real-time databases. Our performance results clearly establish the superiority of the two proposed protocols over 2PL-HP. In general, 2PL-OS/BI has the best performance followed by ACA 2PL-OS and finally 2PL-HP. In the region of low data contention or low workload all three protocols exhibit comparable performance. However, at higher load or data contention 2PL-OS/BI misses significantly fewer deadlines than 2PL-HP. Once again at very high data contention the three protocol miss almost all deadlines. Our resource related experiments indicate that protocols with ordered sharing will benefit significantly from an abundance of resources. More experiments and details justifying our conclusions appear in [4].

Acknowledgements

We would like to thank Alan Lang and Hemant Madan for building the database simulation model. We would also like to thank Bharat Kalayanpur who did the initial design of the simulation model for real-time databases. Finally, we would like to acknowledge the cooperation of various faculty members in our department who gave us access to their machines.

References

[1] R. Abbott and H. Garcia-Molina. Scheduling Real-time Transactions: A Performance Evaluation. In *Pro-*

ceedings of the 14th International Conference on Very Large Data Bases, pages 1–12, August 1988.

[2] R. Abbott and H. Garcia-Molina. Scheduling Real-Time Transactions with Disk Resident Data. In *Proceedings of the 15th International Conference on Very Large Data Bases*, August 1989.

[3] D. Agrawal and A. El Abbadi. Locks with Constrained Sharing. In *Proceedings of the Ninth ACM Symposium on Principles of Database Systems*, pages 85–93, April 1990. An expanded version of this paper appears as technical report TRCS 90-14, Department of Computer Science, University of California, Santa Barbara.

[4] D. Agrawal, A. El Abbadi, and R. Jeffers. Using Delayed Commitment in Locking Protocols for Real-Time Databases. Technical report, Department of Computer Science, University of California at Santa Barbara, 1992.

[5] D. Agrawal, A. El Abbadi, and A. E. Lang. Performance Characteristics of Protocols with Ordered Shared Locks. In *Proceedings of the Seventh IEEE International Conference on Data Engineering*, April 1991. To appear in IEEE Transactions on Knowledge and Data Engineering.

[6] R. Agrawal, M. J. Carey, and M. Livny. Concurrency Control Performance Modeling: Alternatives and Implications. *ACM Transactions on Database Systems*, 12(4):609–654, December 1987.

[7] R. Bayer, H. Heller, and A. Reiser. Parallelism and Recovery in Database Systems. *ACM Transactions on Database Systems*, 5(2):139–156, June 1980.

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

[9] A. P. Buchmann, D. R. McCarthy, M. Hsu, and U. Dayal. Time-Critical Database Scheduling: A Framework for Integrating Real-Time Scheduling and Concurrency Control. In *Proceedings of the Fifth IEEE International Conference on Data Engineering*, pages 470–480, January 1989.

[10] M. J. Carey. *Modeling and Evaluation of Database Concurrency Control Algorithms.* PhD thesis, Electronics Research Library, College of Engineering, University of California, Berkeley, September 1983.

[11] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The Notion of Consistency and Predicate Locks in Database System. *Communications of the ACM*, 19(11):624-633, November 1976.

[12] V. Hadzilacos. A Theory of Reliability in Database Systems. *Journal of the ACM*, 35(1):121–145, January 1988.

[13] T. Härder and A. Reuter. Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys*, 15(4):287–317, December 1983.

[14] J. Haritsa, M. J. Carey, and M. Livny. Dynamic Real-Time Optimistic Concurrency Control. In *Proceedings of the 1990 IEEE 11th Real-Time Systems Symposium*, 1990.

[15] J. Haritsa, M. J. Carey, and M. Livny. On Being Optimistic about Real-Time Constraints. In *Proceedings of the Ninth ACM Symposium on Principles of Database Systems*, pages 331–343, April 1990.

[16] J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley. Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 35–46, August 1991.

[17] H. T. Kung and J. T. Robinson. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems*, 6(2):213–226, June 1981.

[18] A. Law and C. Larmey. *An Introduction to Simulation Using SIMSCRIPT II.5.* CACI Inc., Los Angeles, 1984.

[19] Y. Lin and S. H. Son. Concurrency Control in Real-Time Databases by Dynamic Adjustment of Serialization Order. In *Proceedings of the Eleventh IEEE Real-Time Systems Symposium*, pages 104–112, December 1990.

[20] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: A Transaction Recovery Method supporting fine-granularity Locking and Partial Rollbacks using Write-ahead Logging. Technical Report RJ 6649R, IBM Almaden Research Center, 1989. To appear in the ACM Transactions on Database Systems.

[21] L. Sha, R. Rajkumar, S. H. Son, and C. Chang. A Real-Time Locking Protocol. *IEEE Transactions on Computers*, 40(7):793–799, July 1991.

[22] R. E. Stearns and D. J. Rosenkrantz. Distributed database concurrency control using before-values. In *Proceedings of the ACM SIGMOD Confernece on the Management of Data*, pages 74-83, 1981.