Comparing Two-Phase Locking and Optimistic Concurrency Control Protocols in Multiprocessor Real-Time Databases

Anthony Chiu Ben Kao
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
{achiu,kao}@cs.hku.hk

Kam-yiu Lam
Department of Computer Science
City University of Hong Kong
83 Tat Chee Avenue, Hong Kong
cskylam@cityu.edu.hk

Abstract

Previous studies (e.g., [5]) have shown that optimistic concurrency control (OCC) generally performs better than lock-based protocols in disk-based real-time database systems (RTDBS). In this paper we compare the two concurrency control protocols in both *disk-based* and *memory-resident* multiprocessor RTDBS. Based on their performance characteristics, a new lock-based protocol, called Two Phase Locking – Lock Write All (2PL–LW), is proposed. The results of our performance evaluation experiments show that different characteristics of the two environments indeed have great impact on the protocols' performance. We identify such system characteristics and show that our new lock-based protocols, 2PL-LW, is better than OCC in meeting transaction deadlines in both disk-based and memory-resident RTDBS.

Keywords: real-time databases, concurrency control, multiprocessor, transaction scheduling.

1. Introduction

Real-time database systems (RTDBS) are database systems whose transactions are associated with timing constraints such as *deadlines*. The performance and correctness of a RTDBS depend on how well these constraints are met. For example, an electronic program trading application may respond to a stock's price change by spawning a transaction to calculate a good holding strategy. This transaction needs to be completed by a certain deadline before the market condition is shifted too wide from the original price quotes. Besides meeting transaction timing constraints, a RTDBS needs to observe data consistency constraints as well. A number of studies have been done on this subject, e.g., [1, 5, 6, 7, 8, 9, 13], in which different algorithms and protocols are suggested to schedule transactions according to their priorities. For example, the Earliest-Deadline-First (EDF) algorithm assigns a higher priority to a transaction with an earlier deadline; the High-Priority (HP) concurrency control protocol allows a higher priority transaction abort a lower priority one when they conflict in accessing common data.

In this paper we study the performance of two prevalent concurrency control protocols in both *disk-based* and *memory-resident multiprocessor real-time database systems*, namely, *Two-Phase Locking – High Priority (2PL-HP) [1] and Optimistic Concurrency Control - Broadcast Commit (OCC-BC)* [11, 12]. In [5], it is shown that OCC-BC outperforms 2PL-HP under various system settings in disk-based RTDBS. One major reason is that under 2PL-HP, a (higher priority) transaction that restarts a (lower priority) one may later be restarted or miss its deadline due to the long and unpredictable disk access delay. This results in many *useless restarts* [5] and thus poor utilization of system resources. In this study we revisit the 2PL-HP Vs OCC-BC problem and investigate how data access delay affects the performance of the concurrency control protocols. The motivations behind this study are:

- Disk accesses are relatively slow and cause much unpredictability to transaction response time. Also, disk characteristics are particularly defiant to real-time scheduling. For example, serving disk requests *earliest-deadline-first* causes poor disk utilization and performance [14]. High performance real-time database systems, therefore, tend to move *real-time data* – those accessed by transactions with stringent timing constraints – into main memory for fast, predictable, and parallel accesses [15]. It is therefore useful to study the concurrency control protocols in a memory-resident RTDBS.
- Although optimistic concurrency control is shown to perform better in disk-based real-time systems, most commercially available database systems use two-phase locking. It is thus interesting to see how lock-based protocols can be improved to attain a better performance.

0-8186-8096-2/97 $10.00

The rest of this paper is organized as follows. In Section 2 we give a brief description and comparison of 2PL-HP and OCC-BC. We highlight the characteristics of the two concurrency control protocols that contribute to their different behavior in disk-based and memory-resident systems. Based on the analysis, we propose a variant of 2PL-HP – 2PL-LW which has the advantages of both 2PL-HP and OCC-BC. Section 3 describes our real-time database system model and the simulation model. Results of the simulation experiments are presented in Section 4. Finally, Section 5 concludes the paper.

2. Concurrency Control Protocols

In a database system, transactions interact with each other through reads and writes of data pages. Concurrency control protocols are designed to maintain the database consistency despite concurrent execution of transactions [2]. For RTDBS, these conventional protocols are unsatisfactory because they do not take transactions' priorities into account. For example, under Two-Phase Locking (2PL), a transaction holding a lock could have a lower priority than (and thus block) a high-priority requester – a phenomenon called *priority inversion* [1]. Since the low priority lock holder is discriminated against in its use of system resources (e.g., the CPU), the blocked high-priority transaction will experience extensive delay and is likely to miss its deadline. Applying the concurrency control protocols to a RTDBS thus requires modifications to the basic methods in resolving data conflicts. In the rest of this section, we mention two such modifications: 2PL-HP (Section 2.1) and OCC-BC (Section 2.2). We analyze their advantages and disadvantages. In section 2.3, we propose a new lock-based protocol *Two-Phase Locking – Lock Writes All (2PL-LW)* which adopts the advantages of both 2PL-HP and OCC-BC.

2.1. Two-Phase Locking – High Priority (2PL-HP)

The basic Two-Phase Locking protocol is the most common locking protocol in conventional database systems. With 2PL, a transaction execution consists of two phases. In the first phase, locks are acquired but may not be released. In the second phase, locks are released but new locks may not be acquired. In case a transaction T_R requests a lock that is being held by another transaction, T_H, T_R waits.

As we have just demonstrated, one basic problem of 2PL is the possibility of priority inversions. One solution to this problem is to restart the low-priority lock holder and let the high-priority lock requester proceed. This variant of 2PL is called Two-Phase Locking – High Priority (2PL-HP) [1]. Conflicts are thus resolved by a combination of blocking and restarts under 2PL-HP.

2.2. Optimistic Concurrency Control – Broadcast Commit (OCC-BC)

Although most commercially available database systems use lock-based concurrency control protocols, recent studies [5,10] have suggested that optimistic concurrency control protocols (OCC) provide better performance in RTDBS. Here, we describe a variant of OCC called *Optimistic Concurrency Control with Broadcast Commit (OCC-BC)* [11, 12].

Under OCC-BC, the execution of a transaction is divided into 3 phases: (1) read phase, (2) validation phase and (3) write phase. During the read phase, data pages are read into memory. Computations based on the values of these data pages are performed. New values are computed, but are not written into the database until the write phase. When a transaction T finishes its computation, it enters the validation phase in which all transactions that conflict with T are restarted. (This is done by checking the read-sets and the write-sets of transactions.) Finally, during the write phase, updated data pages are written back to the database. This strategy guarantees that as long as a transaction reaches it validation phase, it will always finish. Conflicts are thus resolved *mainly* through restarting transactions.[1]

2.3. Performance Comparison: 2PL-HP Vs OCC-BC

To compare the performance of 2PL-HP and OCC-BC under both disk-based and memory-resident systems, we need to understand the characteristics of the two concurrency control protocols and the characteristics of the two storage environments. Here, let us first identify three characteristics of a concurrency control protocol which directly affects its performance in a RTDBS:

Number of Restarts. Both 2PL-HP and OCC-BC use transaction restarts as a part of their conflict resolution strategies. Restarting a transaction means giving up the resources already invested into the transaction and is thus detrimental to the system's performance. While OCC-BC relies solely on restarts to

[1] A transaction T_1 that has passed its validation phase but before its write phase may conflict with another transaction T_2 which enters the system after T_1's validation. One solution to this race condition is to mark the data pages to be written by T_1 as "busy". Transaction T_2 will thus be blocked when it tries to read the "busy" pages. OCC-BC thus also carries a small blocking factor in its conflict resolution scheme.

resolve conflict, 2PL-HP uses both restarts and blocking of transactions. In this respect, 2PL-HP uses fewer restarts and wastes less system resources.

Useless Restarts. After a transaction T_1 restarts another transaction T_2, T_1 may later miss its deadline. We call the restart of T_2 *useless*, since this action does not help T_1 to make its deadline. Under 2PL-HP, T_1 may miss its deadline (and thus causes a useless restart) because of the delay experienced in further resource contention (CPU and/or I/O), or because it is later restarted by yet another higher priority transaction during conflict resolution. Under OCC-BC, a transaction that restarts others must be committing, and thus will never be restarted by others. In this respect, OCC-BC is better than 2PL-HP in avoiding useless restarts.[2]

Chained Blocking. Under 2PL-HP, it is possible to form a chain of blocked transactions. In this case, a blocked transaction (e.g., the tail of a chain) may need to wait for a number of transactions to finish before it is granted the access to a data item. The transaction will thus experience extensive delay and misses its deadline. Some of this blocking may, in fact, be unnecessary. For example, a tail transaction T may be delayed by another transaction in the chain which later becomes tardy. The delay or the blocking caused to T is thus *wasteful* and therefore *useless*. OCC-BC is not based on blocking and therefore OCC-BC is better than 2PL-HP in this respect. We summarize the advantages and disadvantages of 2PL-HP and OCC-BC in Table 1. In the table, we use a ✓ (✗) to indicate good (bad) performance with respect to a certain system characteristic.

	2PL-HP	OCC-BC
Number of restarts	✓	✗
Useless restarts	✗	✓
Chained blocking	✗	✓

Table 1. Relative advantages and disadvantages of 2PL-HP and OCC-BC

Next, we need to understand the difference between disk accesses and memory accesses and the implication of it on the different behaviors of the two protocols.

First, disk accesses are slow and unpredictable. The response time of a disk request depends on the position of the disk head (and thus on the previous disk access). Typical disk access time is in the order of milliseconds with a relatively large variance. Moreover, disk requests are served one at a time. Therefore, it is possible that a disk request (and the issuing transaction) is blocked by a few others due to *disk conflict* (even if there is no data conflict among the transactions). Disk access delay and, more importantly, its variance, are amplified by the *chain of requests*. These factors are detrimental to satisfying transaction deadlines. Memory accesses, on the other hand, are pretty constant. Apart from data locking, data requests from different transactions can be served in parallel [15]. Blocking is thus not induced by data accesses in memory-resident system.

In [5], it is found that OCC-BC outperforms 2PL-HP in disk-based RTDBS. However, as we will see in our simulation analysis (Section 4), 2PL-HP performs better than OCC-BC in memory-resident RTDBS. Here, let us briefly explain these observations.

Recall that there are three adverse factors affecting the performance of the concurrency control protocols (Table 1). For example, 2PL-HP is good because it generates fewer restarts, but it is bad because more of the restarts are useless and the blocking of a transaction can result in chained blocking. Whether OCC-BC or 2PL-HP is the better protocol depends on the seriousness of these factors.

As we have just mentioned, data access in a disk-based system could be long, and with a large variance in response time. Under 2PL-HP, a transaction that restarts another one may later suffer extensive delay when it tries to acquire locks on other data items. As a result, the transaction may miss its deadline, wasting the restart it executed. This intensifies the useless restarts problem for 2PL-HP. Also because of long data access time, transactions are holding locks longer. This significantly increases the probability of data conflict and magnifies the extent of chained blocking. Moreover, a high performance disked-based RTDBS uses deadline-cognizant techniques to schedule disk requests [14]. Requests from transactions with earlier deadlines have a higher chance of being served first. The disk thus acts (indirectly) as an agent controlling the progress of the active transactions. Under OCC-BC, the result is that higher priority transactions are advancing at a bit faster pace while lower priority ones are delayed (or blocked) at the disk access level. This improves the performance of OCC-BC by causing (1) fewer restarts, and (2) transactions being restarted at an earlier stage, on average.

The situation in a memory-resident system is just the opposite to a disk-based system. Since data accesses are much faster and can be done in parallel, the adverse effects of useless restarts and chained blocking are much reduced for 2PL-HP. Also due to parallel data accesses, under OCC-BC, low priority transactions are progressing at a similar pace as high priority ones. This

[2] If buffer management is not used, OCC-BC does not prevent useless restarts completely. This is because after the validation phase, new data still needs to be written to the physical database before the deadline. Storage access delay may still cause the transaction to become tardy.

results in a higher level of concurrency, a higher chance of data conflicts and more restarts, and a higher probability that transactions are being restarted at a later stage. OCC-BC thus wastes more resources and performs worse than 2PL-HP.

2.4. Two Phase Locking – Lock Writes All (2PL-LW)

With an understanding of the protocols' behavior and the data access characteristics of disk-based and memory-resident systems, our goal now is to devise a concurrency control protocol that performs well under various system environments: both memory-resident and disk-based RTDBS. Here, we propose the following variant of 2PL: Two-Phase Locking – Lock Writes All (2PL-LW). The rules are:

1. A transaction is divided into a *read phase* and a *write phase*. All reads precede writes. In the read phase, all physical database operations are read operations. Any updates to the data items are written into the private workspace of a transaction instead of into the database immediately. In the write phase, permanent updates of the database will be performed by creating write operations to apply the updates from the private workspace to the database.
2. A data page can be locked under two modes: R-lock for reading and W-lock for writing. R-locks can be shared; W-locks are exclusive and are only acquired in the write phase.
3. When a transaction T tries to read a page that is being W-locked by another transaction, T waits; otherwise, T acquires the R-lock on the page and proceeds.
4. When a transaction T enters its write phase, it checks if any pages in its write set are W-locked by others. If none, T acquires all the W-locks, releases all the R-locks, and restarts all transactions that are holding R-locks on any pages in T's write-set; Otherwise, T waits.
5. When a transaction T finishes, it releases all its W-locks. Transactions that are waiting for the corresponding data pages are rescheduled.

Similar to OCC-BC, under 2PL-LW, a transaction can restart others only when it is in its write phase, and therefore it will never be a target of restart itself. Thus, 2PL-LW generates fewer useless restarts than 2PL-HP does. Also, if a transaction T_A waits for another transaction T_B, T_B must be in its write phase, and thus it will never be blocked by others. Therefore, chained blocking does not occur. Moreover, similar to 2PL-HP, 2PL-LW resolves conflicts by a combination of blocking (rules 3 and 4) and restarts (rule 4). 2PL-LW thus generates fewer restarts than OCC-BC, which uses a purely restart-based conflict resolution. Finally, it can be shown that 2PL-LW is serializable and deadlock-free.

3. Simulation Models

To compare the performance of the concurrency control protocols, we model a disk-based RTDBS and a memory-resident RTDBS. Both models consist of the following five components (Figure 1): a *Source* that generates transactions; a *Transaction Manager* that models the execution of transactions; a *Concurrency Control (CC) Manager* that implements the details of the concurrency control protocols; a *Resource Manager* that models CPU and/or disk I/O resources; and a *Sink* that gathers various statistics, such as the percentage of missed deadlines.

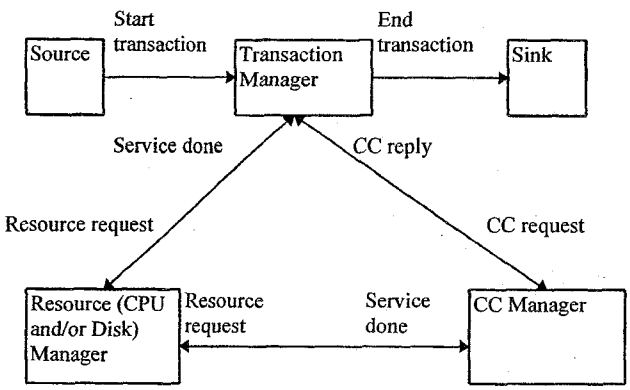


Figure 1. RTDBS model.

3.1. Disk-based RTDBS Model

The disk-based model consists of a shared-memory multiprocessor system operating on disk-resident data. A single transaction queue is shared by the multiple processors. CPU scheduling is preemptive-resume earliest-deadline-first. There are *NumDisks* disks in the system, each with its own service queue. The disks are scheduled independently according to the priority-based variant of the Elevator Algorithm [3]. We assume that all data is accessed from disk and ignore buffer pool considerations. The database itself is modeled as a collection of data pages and data accesses are done one page at a time. Disk access time is calculated by the formula:

$$\text{Disk access time} = SeekFactor \times sqrt(n) + DiskDelay$$

where n represents the number of tracks the disk head moves to service the request. In the simulation, we assume that disk requests are uniformly distributed across all disks and across all tracks within a disk.

Transactions are generated at the source as a Poisson process with a mean arrival rate of *ArrivalRate*. A transaction consists of a series of (disk read, CPU processing) pairs followed by a series of disk writes (Figure 2). The number of data pages read is uniformly distributed in the range [0.5 × *PageCount*, 1.5 × *PageCount*]. *PageCount* is the average number of pages accessed by a transactions. A page read by a transaction may be updated with a probability of *WriteProb*.

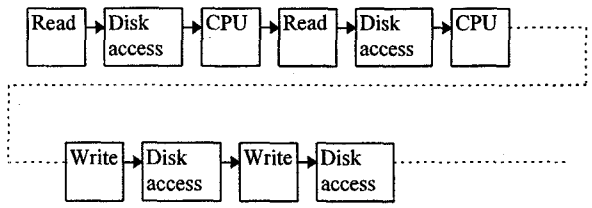


Figure 2. Disk-based RTDBS transaction model

Each transaction has an associated deadline. We use the following formula for deadline assignment: $D_T = A_T + SF \times R_T$, where

D_T = deadline of transaction T,
A_T = arrival time of transaction T,
R_T = resource time of transaction T,
SF = slack factor.

The resource time of a transaction T is its response time if T is executed alone in the system (without encountering any contention for resources and data). The slack factor is a simulation parameter which controls how tight the deadlines are. We assume a *firm deadline* system. I.e., tardy transactions are discarded even if they are incomplete.

Table 2 summarizes the parameters in the simulation model.

Parameter	Meaning	Baseline value
ArrivalRate	Poisson rate of transaction arrivals	5-40 tx/sec
TotalPage	Number of pages in the database	1000 pages
PageCount	Average pages accessed/transaction	16 pages
WriteProb	Write probability/accessed page	0.25
SlackFactor	Slack factor in deadline assignment formula	4.0
NumCPUs	Number of processors	10
NumDisks	Number of disks	20
NumDiskPrio	Number of priority levels at each disk	5
NumTracks	Number of tracks per disk	1000
PageCpuTime	CPU time for processing data page	10ms
BCopyTime	Page memory block copy time	0.5ms
DiskDelay	Disk rotational + transfer delays	15ms
SeekFactor	Factor relating seek time to distance	0.5ms

Table 2. Simulation parameters and their baseline values

3.2. Memory-Resident RTDBS Model

The memory-based model is similar to the disk-based model, except that disk requests are replaced by memory copy (between shared space and local space of transactions). We use the parameter *BCopyTime* to represent the copy time of a page. Figure 3 illustrates the transaction model.

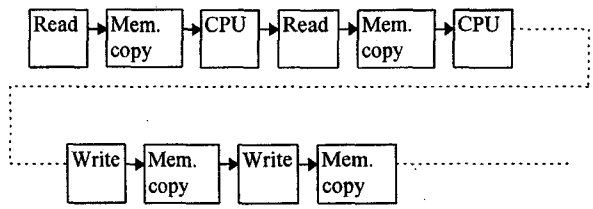


Figure 3. Memory-resident RTDBS model.

4. Simulation Results

In this section we present our simulation results comparing the three concurrency control protocols: 2PL-HP, OCC-BC, and 2PL-LW in both disk-based and memory-resident RTDBS. Due to space limitation, our discussion here is brief and will concentrate on the results which can illustrate the performance characteristics of the protocols. Interested readers are referred to [16] for more details.

The simulation is written in the simulation language CSIM-17 [4]. At least 20000 transactions are generated at each simulation run. To highlight the protocols' performance difference, we model both the disk-based and memory-based systems under a high data contention and resource contention scenario. The baseline parameter settings are shown in Table 2.

4.1. Disk-Based Model

Figure 4 shows the miss rate (the fraction of missed deadlines) of the three protocols in the disk-based environment as the transaction arrival rate ranges from 5 to 40 transactions per seconds. Similar to the observation discussed in [5], 2PL-HP misses more deadlines than OCC-BC does. The modified two-phase locking protocol, 2PL-LW, on the other hand, performs the best. Under the experiment setting, 2PL-LW outperforms 2PL-HP and OCC-BC by a wide margin when the load to the system is moderate (15-25 transactions/second). For example, at *ArrivalRate* = 20, 2PL-LW misses 7.5%

of deadlines, which is 2.3 times less than that of OCC-BC, and is 3.4 times less than that of 2PL-HP.

As we have discussed in Section 2, one of the major factors that affects the protocols' performance is the amount of system resources wasted in restarting transactions. Figure 5 shows the number of restarts per transaction under the three protocols versus the system load. Also shown are the curves of *useful* restart per transaction (a restart is useful if the transaction that restarts another meets its deadline).

From the figure, we see that the general shape of the restart curves is ascending at low *ArrivalRate* and descending at high *ArrivalRate*. The reason is that at low load, as *ArrivalRate* increases, more transactions enter the system creating more conflict and restarts; At high load, however, system resources, like disk and CPU, are highly utilized. Queues start to build up at the resources and transactions experience long delay waiting for the resources. A transaction is thus likely to become tardy while it is waiting and is discarded before it conflicts with other transaction causing restarts. The restart curves thus go down at high value of *ArrivalRate*.

Comparing the restart curves of the protocols, we see that OCC-BC generates the highest number of restarts. This is due to its restart-based conflict resolution. However, the number of *useless* restarts are relatively small. (I.e., the difference between the "restart" curve and the "useful restart" curve is relatively small.) This is because only the transactions which have already entered its validation phase can restart others, and these transactions are very close to finishing. 2PL-HP, on the other hand, generates fewer restarts than OCC-BC does, but a good number of these restarts are wasteful. (The curves labeled "2PL-HP" and "2PL-HP useful" are pretty wide apart.) Letting a transaction restart others when it is in the middle of its execution therefore does not seem to do well. Finally, 2PL-LW generates few restarts and when it does, the restart is usually useful. This is because 2PL-LW uses a combination of blocking and restarts (which implies low restart rate) and restarts are done by those transactions in their write-phase only.

To find out how much resource does a transaction restart waste, we measure the *restart point* of transactions [5]. Figure 6 shows, on average, the fraction of work a restarted transaction has completed. From the figure, we see that OCC-BC restarts transactions at a much earlier stage than 2PL-HP does. For example, at *ArrivalRate* = 30, under OCC-BC, a transaction is only 30% complete when it is restarted. Comparing this with the 60% completion under 2PL-HP, a restart caused by OCC-BC is about half as expensive as it is under 2PL-HP. As a combination of 2PL-HP and OCC-BC, the average transaction restart point for 2PL-LW falls in between that of the two basic protocols. For example, at *ArrivalRate* = 30, a restarted transaction under 2PL-LW is about 45% complete.

The effect of wasting system resources due to restarts is reflected by the system utilization. This is shown in Figure 7 in which we compare the *useful CPU utilization* under the three protocols. I.e., the fraction of time the CPUs are used to serve transactions that make their deadlines. We see that 2PL-HP makes the worse use of CPU cycles while 2PL-LW is the best within the range of *ArrivalRate* shown. This helps explain the relative miss rates of the protocols indicated in Figure 4.

4.2. Memory-Based Model

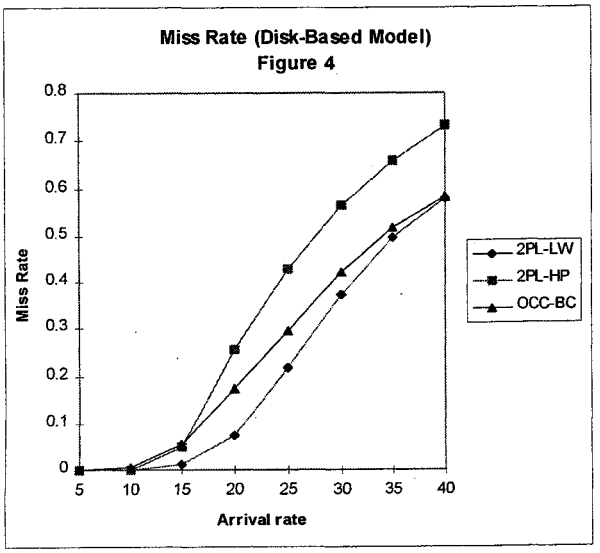
We repeat the experiments comparing the three concurrency control protocols under the memory-resident RTDBS model. The result is presented in Figures 8, 9, 10, and 11 showing the miss rates, (useful-) restarts, restart points, and CPU utilization respectively.

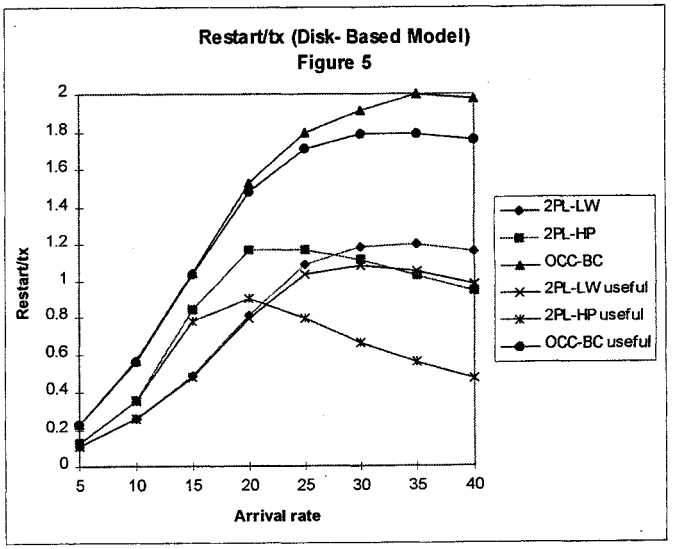
Interestingly, OCC-BC misses many more deadlines than the two locking protocols (Figure 8). The performance of 2PL-HP and 2PL-LW are similar with 2PL-HP having a slight edge. The reason for 2PL-HP's good performance is that memory allows parallel and fast accesses. Data access requests can thus be served promptly. This reduces the amount of lock holding time, the probability of conflict and thus the number of restarts (Figure 9). Moreover, since transaction execution becomes much shorter, it is likely that a restarted transaction still has enough amount of time to meet its deadline. This is shown in Figure 9 where the *useful restarts* curves are very close to the restarts curves. The result of these is that 2PL-HP gives the highest *useful* system utilization (Figure 11), and thus it has the lowest miss rate among the protocols.

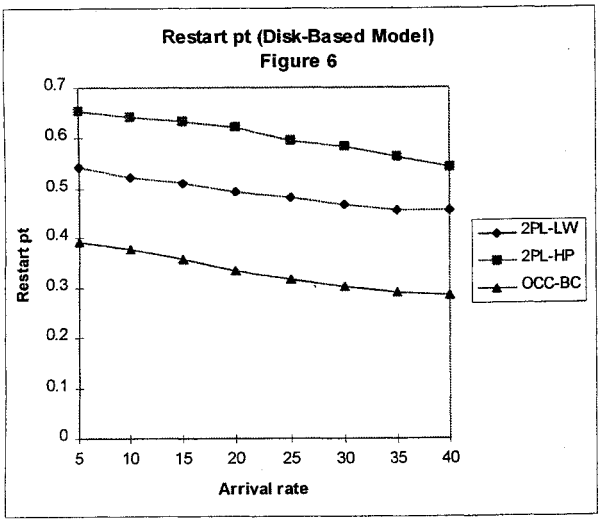
5. Conclusions

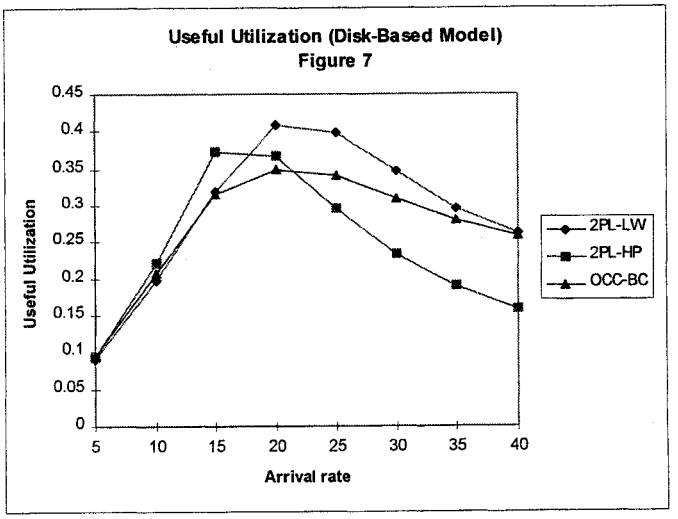
In this paper we compared the performance of 2PL-HP and OCC-BC in both disk-based and memory-resident RTDBS. Although it was shown that OCC-BC performs better than 2PL-HP in disk-based systems, our study shows that due to the differences in the access characteristics between disk and memory, 2PL-HP outperforms OCC-BC in a memory-resident system. We indicated three important factors, namely, the number of restarts, useless restarts, and chained blocking, which have great impact on the protocols' performance. Based on the analysis, we proposed a new lock-based protocol, 2PL-LW, which adopts the advantages of both 2PL-HP and OCC-BC. Our simulation results showed that 2PL-LW performed the best among the protocols in a disk-based system, and comparably to 2PL-HP in a memory-resident system. Since lock-based concurrency control

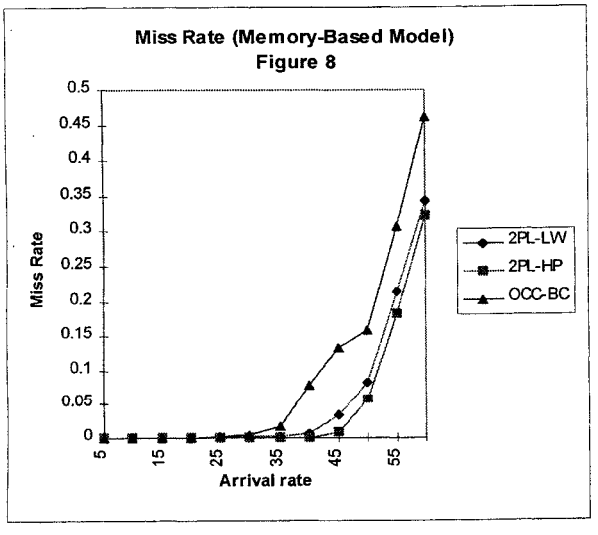
protocols are more popular and more easily implemented, 2PL-LW should be a better alternative than OCC-BC in multiprocessor real-time database systems.

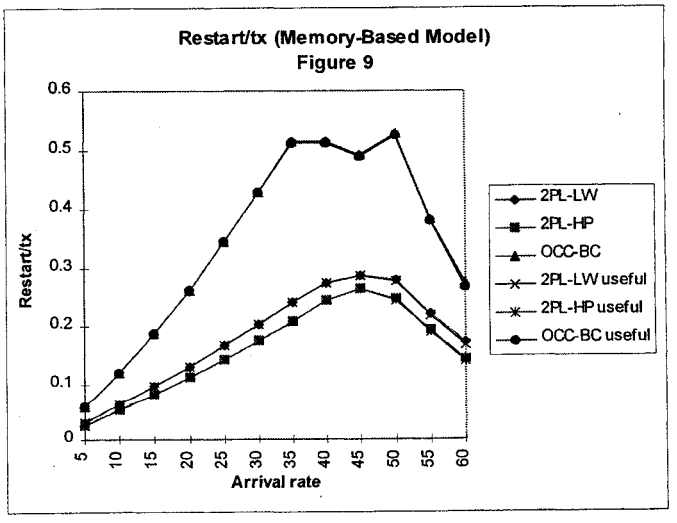


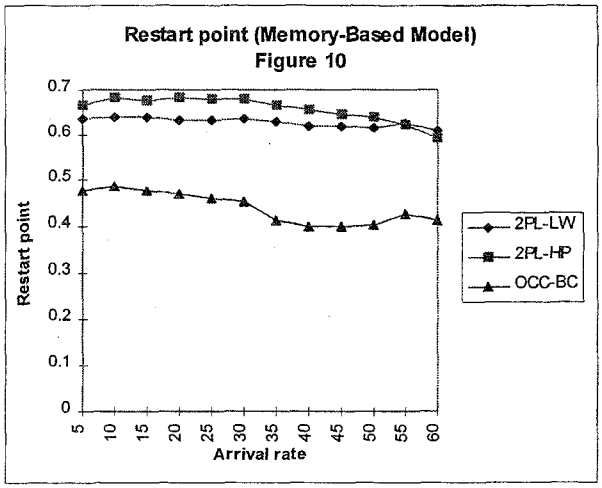


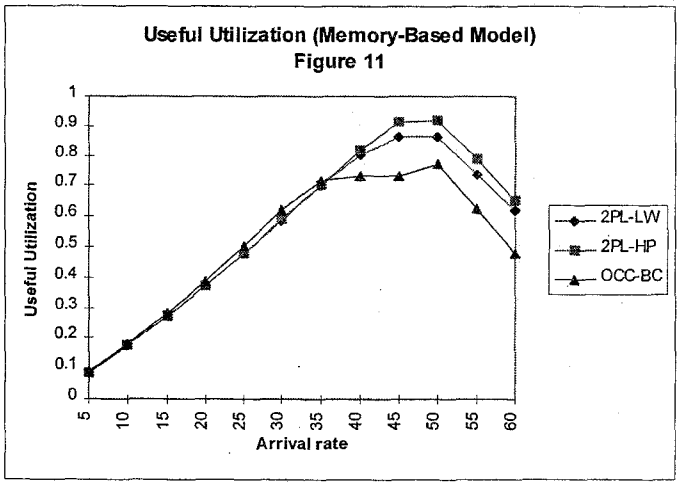












References

[1] Abbott, R., and Garcia-Molina, H., "Scheduling Real-Time Transactions: A Performance Evaluation," *Proc. of the 14th Conference on Very Large Database Systems*, Aug. 1988.

[2] Bernstein, P.A., Hadzilacos, V. and Goodman, N., "Concurrency Control and Recovery in Database Systems", Addison-Wesley, Reading, Mass., USA, 1987.

[3] Carey, M., Jauhari, R., and Livny, M., "Priority in DBMS Resource Scheduling," *in the Proceedings of the 15th Conference on Very Large Database Systems*, Aug., 1989.

[4] "CSIM 17 User Guide," Mesquite Software, Inc. (URL: http://www.mesquite.com/1_intro.html).

[5] Haritsa, J. R., Livny, M., Carey, M. "On Being Optimistic about Real-Time Constraints," *in the Proceedings of the 9th ACM Symposium on Principles of Database Systems*, 1990.

[6] Haritsa, J. R., Livny, M., Carey, M. "Earliest Deadline Scheduling for Real-Time Database Systems," *in the Proceedings of IEEE Real-Time System Symposium*, pp. 232-242, 1991.

[7] Haritsa, J. R., Carey, M., and Livny, M., "Data Access Scheduling in Firm Real-Time System," *in the Proceedings of 13th International Conference on Distributed Computing Systems*, pp. 428-437, 1993.

[8] Haritsa, J. R., Livny, M., Carey, M., "Value-Based Scheduling in Real-Time Database Systems," *The VLDB Journal*, vol. 2, no. 2, pp. 117-152, 1993.

[9] Huang, J, and Stankovic, J., Ramamritham, K. and Towsley, D., "Priority Inheritance in Soft Real-Time Databases," *Journal of Real-Time Systems*, vol. 4, no. 3, pp. 243-268, 1992.

[10] Huang, J. and Stankovic, J., "Experimental Evaluation of Real-Time Concurrency Control Schemes," *in the Proceedings of the 17th VLDB Journal*, vol. 2, no. 2, pp. 117-152, 1993.

[11] Menasce, D., and Nakanishi, T., "Optimistic versus Pessimistic Concurrency Control Mechanisms in Database Management Systems," *Information Systems*, vol. 7-1, 1982.

[12] Robinson, J., "Design of Concurrency Controls for Transaction Processing Systems," *Ph.D. Thesis, Carnegie Mellon University*, 1982.

[13] Ulusoy, O., "Processing of Real-time Transactions in a Replicated Database Systems," *Journal of Distributed and Parallel Databases*, vol. 2, no. 4, pp. 405-436, 1994.

[14] Abbot, R., Garcia-Molina, H., "Scheduling I/O Requests with Deadlines: A Performance Evaluation," *IEEE Real-Time System Symposium*, 1990, pp. 113-124.

[15] Huang, K. "Advanced Computer Architecture: Parallelism, Scalability, Programmability," McGraw Hill, 1993.

[16] Chiu, A., and Kao, B., "Comparing Two-Phase Locking and Optimistic Concurrency Control Protocols in Multiprocessor Real-Time Databases," Technical Report, TR-96-09, Department of Computer Science, The University of Hong Kong.