

Designing Inter-Class Concurrency Control Strategies for Real-time Database Systems with Mixed Transactions¹

Kam-yiu Lam¹, Tei-Wei Kuo² and Tony S.H. Lee¹

Department of Computer Science¹
City University of Hong Kong
HONG KONG
83 Tat Chee Avenue, Kowloon
Email: cskylam@cityu.edu.hk

Department of Computer Science and
Information Engineering²
National Chung Cheng University
Chiayi, 621 Taiwan, ROC
Email: ktw@cs.ccu.edu.tw

Abstract

Although many efficient concurrency control protocols had been proposed for real-time database systems, they mainly design for the systems with a single type of real-time transactions. Due to the very different performance requirement of each type of real-time transactions, these proposed protocols may not be suitable to Mixed Real-time Database Systems (MRTDBS) where different types of real-time transactions, and even non-real-time transactions, may co-exist in the systems at the same time. In this paper, we propose strategies for resolving data conflicts between different types of transactions in a MRTDBS so that different performance requirements can be achieved and, at the same time, the overall system performance can be improved. The performance of the proposed strategies is evaluated and compared with a real-time optimistic approach. The performance of our proposed conflict resolution methods has also been investigated in a more realistic environment with limited number of priority levels and disk resident data items.

1 Introduction

The research in Real-time Database Systems (RTDBS) has received a lot of interests in recent years. Generally, a RTDBS is defined as a database system where the transactions have deadlines on their completion times. Normally, transactions in a RTDBS can be classified into different types based on the consequences of missing their deadlines [5, 14]. Any deadline violation of a hard real-time transaction (HRT) may be catastrophic, whereas the deadline violation of a soft real-time transaction (SRT) may result in a serious degradation in system performance. Firm real-time transactions (FRT) are SRT except that the deadline violation of a firm real-time transaction will cause the transactions to abort, as its value is totally lost after its deadline. Different types of

transactions may have very different properties and performance requirements. For example, the concurrency control protocols designed for HRT or SRT have to guarantee no deadline violations or to minimize the number of missed deadlines, respectively.

In the last decade, researchers had proposed various efficient concurrency control protocols to meet transaction deadlines, and, at the same time, to maintain database consistency for RTDBS, e.g., [3, 8, 9, 12, 16]. However, most of the previous work were focused on the concurrency control of a single type of transactions and at the same time, they always assumed the system is running in an ideal environment, where an unlimited number of priority levels are available, and data items are all memory-resident. Little work is done in the consideration of joint scheduling of mixed workloads, especially when a more realistic system model is considered.

Concurrency control protocols designed for RTDBS with a single type of transactions may not fit the performance requirements of other types of real-time transactions. For example, the well-known priority ceiling protocol (PCP) [8, 12] (designed for HRT) requires a static system that is usually not true for a system with SRT. The Higher Priority Two Phases Locking Protocol (HP-2PL) [9], which is designed for SRT, cannot guarantee the deadlines of HRT or even be suitable to a RTDBS that consists of mixed SRT and NRT. The response times of NRT may be greatly affected because of the restart policy of HP-2PL. The objective of our work is to explore the joint scheduling of different types of transactions under more realistic system conditions. We call a RTDBS with different types of transactions as a Mixed RTDBS (MRTDBS).

An example of MRTDBS is a programmed stock trading system [5, 11]. Updates of stock data are done by SRT that capture the up-to-date information of the stock market and the status of stocks. Missing the deadline of

¹ This research was supported in part by Hong Kong UGC CERG grant no. 9040353 and by a research grant from the National Science Council Taiwan under Grant NSC89-2213-E-194-011.

an update means that the information about a stock may become out-dated, and a lot of missed deadlines of updates will result in serious problems for stock analysis. Transactions for system management and general queries are usually NRT, which have no deadlines but they prefer to have good response times. Buy/sell transactions for stock trading are firm real-time transactions or even HRT if any of their deadline violations may result in a great loss in stock trading. They must be completed before pre-defined deadlines.

Another related problem, which has been greatly ignored in most of the previous work on the area, is that they often assume an unlimited number of priority levels and a memory-resident database. However, a realistic database system often only provides a limited number of priority levels for the scheduling of transaction [6] and the database is normally disk resident. Priority mapping mechanisms are needed [7] to convert the priorities of the transactions to the priority levels supported by the system. However, little work has been done in the study of the impact of different priority mapping algorithms on the performance of different real-time concurrency control protocols. The problem is further complicated when different types of transactions are considered at the same time. The real-time scheduling of MRTDBS becomes further complex when disk-resident data may exist for SRT and NRT. Most of the previous studies on concurrency control protocols for RTDBS have ignored the cost of transaction restart in a disk-resident environment.

In this paper, we explore the concurrency control issues in MRTDBS. Instead of proposing yet another concurrency control protocol, we focus on the design of strategies for resolving the inter-class data conflicts in a MRTDBS with hard, soft and non-real-time transactions. We adopt protocols that are effective for each single type of transaction to resolve intra-class data conflicts and propose methods for inter-class data conflict resolution. We shall also evaluate the impacts of different practical issues of the system environment on the real-time concurrency control protocols, e.g., a limited number of priority levels and disk access delay, using simulation.

The rest of this paper is organized as follows: Section 2 summarizes related work on real-time concurrency control protocols. Section 3 defines a MRTDBS model. Section 4 proposes concurrency control strategies for MRTDBS. The performance evaluation and experimental results are provided in Section 5. Section 6 is the Conclusions.

2 Real-time Concurrency Control Protocols

2.1 Real-time Concurrency Control Protocols for Single Type of Transactions

Real-time concurrency control protocols are often extended from traditional concurrency control protocols [3, 8, 9, 12, 16]. Most of them can be classified as either conservative (e.g., locked-based) or optimistic concurrency control protocols. Under the lock-based protocols, a transaction must lock a data item in a proper mode before it accesses the data item. Conservative concurrency control protocols often use blocking to prevent the occurrences of any serializability violation. Optimistic concurrency control protocols often allow transactions to run until their validation times. If there is any possibility of serializability violation, the conflicting transactions will be restarted.

Most lock-based real-time concurrency control protocols follow the two-phase locking scheme (2PL) because of its simplicity and popularity in commercial database systems. For example, with the Higher Priority Two-Phase Locking Protocol (HP-2PL) [9] a lower priority transaction is restarted if it is in conflict with the lock request of a higher priority transaction. On the other hand, a lower priority transaction has to wait for all of its conflicting higher priority transactions to release the data items locked by them. The priority inheritance principle [3, 12] is often used to resolve the priority inversion problem in the 2PL-based protocols, where priority inversion is a phenomenon in which a higher priority transaction is blocked by a lower priority transaction. Under the principle of priority inheritance, the priority of a lower priority transaction is raised to the highest priority of the transaction that is blocked by it. The rationale behind priority inheritance is to reduce the blocking time of any higher priority transaction by completing the blocking lower priority transaction earlier. The priority ceiling protocol (PCP) [8] is proposed to bound the worst-case blocking time of any transaction. The lock request of a transaction is blocked if the priority of the transaction is not higher than the maximum priority ceiling of all data items locked by other transactions. The priority ceiling of a data item is the maximum priority of all transactions that may lock the data item.

Many real-time concurrency control protocols are based on the optimistic concurrency control (OCC) methods [16]. The execution of any transaction under the OCC-based protocols is divided into three phases: read (processing), validation, and write phases. Any violation of serializability is detected in the validation phase. Basically, if a transaction fails the serializability test, it is restarted or it will restart other conflicting transactions. Various priority-based validation strategies had been proposed. In the optimistic concurrency control with wait 50 (OCC-W50), the validating transaction is allowed to commit if its priority is greater than 50% of the conflicting transactions. In OCC-sacrifice, the validating transaction is restarted when its priority is not higher than all the conflicting transactions.

Transaction priorities are often used to resolve data conflict in concurrency control protocols. The earliest deadline first (EDF) priority assignment method is used very often in RTDBS researches, where EDF assigns the transaction with the closest deadline to have the highest priority. The least slack time first priority assignment method is also often used, where the slack time equals to the difference of the deadline, and the sum of the current time and the remaining execution time. The highest priority is assigned to the transaction with the smallest slack time in the system. Fixed priority assignment method such as the rate monotonic scheduling method [8] is also used in many RTDBS researches. Although the above methods are being shown effective in lots of previous work, most of them assume that the system can provide an infinite number of priority levels. Apparently, current operating systems can only provide a very limited number of priority levels, e.g. QNX (a real-time OS) and POSIX (real-time extension of UNIX) only have 128 priority levels [6]. As shown in the priority mapping methods, e.g., [7], the performance of a system highly depends on the number of priority levels provided in the system. Transactions that have different priorities under an infinite number of priority levels may now have the same priority, extra blocking time is thus introduced.

2.2 Real-time Concurrency Control Protocols for Mixed Types of Transactions

The reduced ceiling protocol (RCP) [10] is proposed to schedule hard and soft real-time transactions in a MRTDBS. The deadlines of hard real-time transactions are guaranteed using the principles of the PCP. The OCC principles are adopted for SRT, and the validation phase is made non-preemptive.

Another approach in mixed scheduling of transactions is proposed in [1]. Transactions in the system are classified based on their characteristics and criticality. Different principles are used to schedule different classes of transactions. However, it is assumed that hard real-time transactions (i.e., class-I transactions in the work) do not have any data conflict with other class-I transactions and soft real-time transactions. In [15], a two-level concurrency control framework is proposed for MRTDBS. In the framework, a master concurrency controller is proposed to detect the possible inter-class data conflicts using a serialization check on the time-stamps of the transactions. Intra-class data conflicts are detected and resolved by individual schedulers. Different concurrency control protocols can be incorporated into the systems by replacing each individual scheduler with a new one. The framework provides an excellent architecture in extending conventional non-real-time database systems to MRTDBS by adding the master concurrency controller and providing schedulers for real-time concurrency control.

Different from the previous work, which are usually restricted to conflict resolution between one or two types of transactions, our proposed methods will aim for a more general system situation where consists all types of transactions, HRT, SRT and NRT. We must emphasize that, different from the past work, this paper aims at exploring inter-class conflict resolution strategies and minimizing the impacts of different concurrency control protocols of different types of transactions on each another, instead of proposing yet another two-level concurrency control framework. At the same time, we will also focus the mixed scheduling of transactions under a more realistic system model.

3 The MRTDBS Model

A simple but typical RTDBS model is adopted in this work. The model consists of a transaction manager (TM), a scheduler (S), and a resource manager (RM), as shown in Figure 1 [3, 4, 8, 16]. The TM is responsible for transaction initialization, abort, commit, and restart. It is also responsible for assigning priorities to transactions according to their deadlines and criticality levels. The scheduler is responsible for concurrency control and scheduling of transactions in using the CPU. It receives operations from the TM and determines whether the operations should be accepted, blocked, or rejected based on the adopted concurrency control policy. It is assumed that it is a two-layer scheduler. The first layer, which contains the global scheduler, filters transactions by their classes, e.g., hard, soft and non-real-time transactions, and then sends them to the second level, which contains individual schedulers, called local schedulers, for intra-class conflict resolution. The global scheduler does the inter-class conflict detection and resolution. A common lock table is used for the conflict detection, and transactions may be blocked or restarted in case of data conflicts. The resource manager handles all the data accesses and recovery. It controls the cache and the disk.

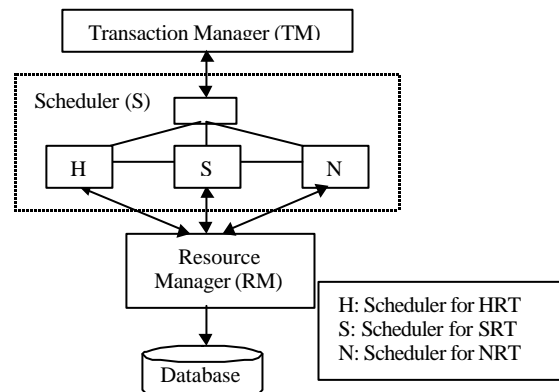


Figure 1: The MRTDBS model

There are three types of transactions in the system:

hard real-time (HRT), soft real-time (SRT) and non-real-time (NRT) transactions. It is assumed that all of them consist of a sequence of read/write operations with the last operation as the commit or abort operation. HRT are periodic. The data items to be accessed by the HRT are assumed to be at the main memory buffer. (Since the data requirements of the HRT are pre-defined, the system will pre-fetch their required data objects into the buffer before their executions.) Other types of transactions may find their data items in the main memory buffer or in the disk. SRT are associated with soft deadlines, and they are allowed to continue after their deadline expiration. The number of missed deadlines and the mean lateness of SRT should be optimized, where the lateness of a transaction is defined as its completion time minus the deadline if the completion time is after the deadline. NRT do not have any deadline constraint. However, the mean response time of NRT is preferable to be kept low.

4 Mixed Concurrency Control (MCC) Protocols

4.1 Intra-Conflict Resolution

In this section, we introduce a *Mixed Concurrency Control (MCC)* protocol to resolve intra-class and inter-class data conflicts in a MRTDBS. Since our study is focused on inter-class conflict resolution, we adopt proper and well-known strategies for intra-class conflict resolution in the MCC protocol to meet the performance requirements of each type of transactions. We assume that HRT have higher priorities than SRT and NRT because of their high criticality. The priorities of SRT are also assumed to be higher than NRT.

In MCC, the Priority Ceiling Protocol (PCP)² principles are adopted for resolving the intra-class data conflicts between HRT. The reason is that it can guarantee the schedulability of HRT. OCC-W50 is used to schedule SRT because of its superior performance in scheduling SRT. The validating transaction is blocked until most of the conflicting SRT have lower priorities. Once a validating SRT passes its validation test, all the conflicting SRT are restarted. We adopt the two-phase locking (2PL) protocol as the basic protocol for NRT. Under 2PL, a transaction cannot lock any data item after

² An important variant of the PCP is the read/write PCP [12]. The read/write PCP and the PCP use the same principles for the concurrency control, except that the PCP only provides exclusive locks, and the read/write PCP provides read and write locks for read and write operations, respectively. Since the focus of the paper is not to study the PCP, we choose to use the PCP in the study in order to simplify the discussion and to focus this work on the mechanisms for resolving data access conflicts between soft and hard real-time transactions. It should be noted that all of the nice properties of the proposed protocol would remain if we adopt the read/write PCP.

it releases a lock. In the following section, we shall consider strategies for inter-class conflict resolution.

4.2 Inter-Class Conflict Resolution

For inter-class conflict resolution, the blocking time of HRT due to SRT executions should be bounded in order to maintain the schedulability of HRT. (The system must guarantee that all the deadlines of the HRT can be met.) SRT should not cause any blocking or transaction restarts to other SRT before its validation in order to benefit the important properties of OCC-W50, e.g., having a greater number of fruitful restarts. The blocking of SRT and HRT by NRT should be prevented as NRT are much less critical and do not have any deadline constraint. In order to minimize the amount of workload wasted on a SRT due to data conflict with a HRT, a SRT may be blocked if it wants to access a data item, which is accessing by a HRT in a conflicting mode. Table 1 summarizes the conflict resolution strategies for different types of inter-class conflicts.

		Requester/Validating transaction (SRT only)		
		HRT	SRT	NRT
Holder	HRT	PCP	Block Requester	Block Requester
	SRT	Restart Holder**	OCC-W50	Block Requester
	NRT	Restart Holder	OCC-W50	2PL***

* Priority of SRT is raised to be higher than the priority of HRT when a SRT is at the validation phase

** Requester waits if holder is in its validation phase.

*** Deadlock resolution algorithm is incorporated.

Table 1: Intra-Class and inter-Class conflict resolution methods

As shown in above, when a HRT tries to lock a data item, which is accessed by a SRT or a NRT in a conflicting mode, the system restarts the conflicting SRT or NRT. The purpose is to guarantee the deadlines of the HRT by minimizing the blocked. However, blocking is used if the conflicting SRT is in the validation phase. When a SRT enters its validation phase, its priority will be raised to the highest priority level in the system, and any conflicting HRT must wait until the SRT commits. The above priority raising policy is reasonable because the SRT is near its completion (and the write phase is usually short in SRT when common operating system buffering mechanisms are provided).

As astute readers may notice, SRT may check locks (against HRT) while they are executing (in the read phase) and check data conflicts (against other SRT and NRT) at their validation phase. In other words, it combines both pessimistic and optimistic approaches for data conflict detection. It uses a pessimistic approach for the detection of possible conflicts with HRT, which are more critical than SRT. Any such kind of conflicts will

cause the SRT to be blocked or restarted. If a SRT finds itself in conflict with any HRT while it is in its validation phase, it will restart itself. If a SRT is in the read phase, it may block itself to avoid being restarted. On the other hand, optimistic approach is used for resolving possible conflicts between soft and non-real-time transactions. If a SRT finds itself in conflict with any NRT when it is in its validation phase, it will restart those conflicting NRT. Delaying the conflict detection of SRT until the validation phase can maximize the number of fruitful restarts of NRT because the SRT may be aborted due to data conflicts with HRT.

When a NRT requests to lock a data item, which is locked by a HRT in a conflict mode, the NRT is blocked until the lock is released. The reason is that it is likely to be restarted even though it is allowed to continue. When a NRT requests to lock a data item, which has been accessed by a SRT in a conflict mode, the NRT is also blocked. Otherwise, it will also be restarted when the SRT enters its validation phase.

4.3 Implementation - Conflict Detection

In the implementation of the MCC protocol, a common lock table is used for both inter-class and intra-class conflict detection. Although SRT are scheduled using the optimistic principles, they are also required to set a lock (a shared lock) before they access the corresponding data item. The locks for SRT can be used for resolving data conflicts while a SRT is in the validation phase. The system maintains the read/write data sets of every executing SRT for conflict resolution following the optimistic principles. When a data item is in the read set (or write set) of a SRT, we say that the data item is pre-read locked, i.e., PRL, (or pre-write locked, i.e., PWL) by the SRT. In order to prevent the blocking by other executing SRT due to the locking, the pre-locks are compatible to each other. During the validation phase, a SRT converts all its pre-locks to exclusive locks, the VL, which are used to prevent HRT and NRT to access the data item until the completion of the validation phase and the write phase.

There are two kinds of locks for HRT: hard read lock (HRL) and hard write lock (HWL). There are also two kinds of locks (N-locks) for NRT: read lock (NRL) and write lock (NWL). The compatibility of the locks is shown in Table 2.

Note that when a validating transaction wants to convert a pre-lock to a VL, it will be allowed even though other H-locks and N-locks are already set on the same data item. For that case, the data conflict between the SRT and other types of transactions will be resolved according to the principles defined in the last sub-section and the resolution of the inter-class data conflicts will be performed after all the P-locks have been converted to VL. Since only one SRT can be in the validation and

write phase as required by the optimistic method, the VL are not compatible with each other.

		Requester/Validating transaction(SRT only)						
		HRL	HWL	PWL	PRL	VL	NRL	NWL
Holder	HRL	✓	✗	✗	✓	✓	✓	✗
	HWL	✗	✗	✗	✗	✓	✗	✗
	PRL	✓	✓	✓	✓	✓	✓	✗
	PWL	✓	✓	✓	✓	✓	✗	✗
	VL	✗	✗	✗	✗	✗	✗	✗
	NRL	✓	✓	✓	✓	✓	✓	✗
	NWL	✓	✓	✓	✓	✓	✗	✗

✓: The lock requester is allowed to set the lock. When PCP is adopted, priority ceiling must be checked.

✗: The lock requester is denied.

Table 2: Lock compatibility table

4.4 Correctness of MCC Protocol

Theorem 1: All hard and soft real-time transactions in MCCP schedules are deadlock-free.

Proof. The correctness of this theorem follows from the deadlock-freeness property of PCP, OCC-W50 and the deadlock resolution mechanism for NRT. Note that there can not be any deadlock involving different types of transactions. Q.E.D.

Theorem 2: All MCC schedules are serializable.

Proof. The correctness of this theorem follows from the serializability properties of PCP, OCC-W50, and 2PL. Note that soft real-time and non-real-time transactions will be aborted if their commitment may result in serializability violation with hard real-time transactions or any soft real-time and non-real-time transactions. Q.E.D.

5 Performance Studies

5.1 The Performance Model

We have implemented a MRTDBS simulation model based on the MRTDBS model introduced in Section 3. In order to study the performance of the proposed mixed concurrency control (MCC) protocol, a modified version of optimistic concurrency control wait-50 (OCC-W50) is implemented for comparison. The reason to compare our protocol with OCC-W50 instead of other real-time concurrency control protocols is because OCC-W50 has shown its good performance for RTDBS with a single type of transactions, especially for SRT. Although the PCP can guarantee the deadlines of HRT, it cannot be adopted in MRTDBS because its assumptions are not suitable to soft real-time and non-real-time transactions. Note that the purposes of the simulation studies are not to investigate the performance of the proposed protocols and approaches for a specific real-time database application. Instead, the objectives are to identify the performance characteristics of the proposed approaches, and to demonstrate the capability of the proposed protocols in improving the performance of MRTDBS.

In the implementation of the modified version of OCC-W50, all the transactions have to go through three phases: read, validation and write phase. HRT are assigned the highest priority in accessing data items while all the NRT are given the same lowest priority. The rate monotonic algorithm is used for the priority assignment of HRT. Priorities of SRT are assigned according to the earliest deadline first (EDF) policy, and their priorities are set to be between the priorities of NRT and HRT. Since HRT are much more critical than soft real-time and non-real-time transactions, the OCC-sacrifice principle is used, e.g., when a SRT or a NRT is in the validation, it will be restarted if it is in conflict with a HRT. The conflicts between SRT and NRT, and the inter-class conflicts are resolved by the OCC-W50 principle.

5.2 Model Parameters and Performance Measures

The following table summarizes the parameters and their baseline values. In the first part of the simulation experiments, we assume a main memory resident database, i.e. all read and write operations on data items are performed at the main memory.

Parameters	Baseline Values
Database size	1000
Transaction length of hard real-time transactions	5 operations
Mean inter-arrival times of soft real-time transactions	1 to 11 ms
Mean inter-arrival rate of non-real-time transactions	500 ms
Number of hard real-time generators	5
Periods of hard real-time transactions	60, 75, 90, 110, 125 (ms)
Transaction length of a soft real-time transaction	1 – 10 operations (uniformly distributed)
Transaction length of a non-real-time transaction	2 – 15 operations (uniformly distributed)
Probability of write operation in a soft real-time transaction	0.5
Probability of write operation in a non-real-time transaction	0.5
Slack range of soft real-time transactions	1.5 – 3
Number of CPUs in the system	2
CPU execution time for an operation	2 ms to 4 ms (uniformly distributed)
Number of disks in the system	4
CPU time to update a data item at memory	0.1 ms to 0.2 ms
Disk access time for retrieving a data item	4 ms to 8 ms
Disk access time for updating a data item	5 ms to 10 ms
Buffer hit probability	0.5
CPU time for checking a lock	0.1 ms

CPU time for validating a data item	0.1 ms
-------------------------------------	--------

Five hard real-time transaction generators are defined in the model, and their periods are 60, 75, 90, 110 and 125 (ms). In other words, the total hard real-time transaction workload is about 15%. The deadline of a hard real-time transaction is defined based on its period and arrival time:

$$\text{Deadline of a HRT} = \text{slack factor} \times (\text{arrival time} + \text{period}),$$

where the value of slack factor is set to be smaller than 1. The purpose is to make their deadlines tighter and increase the probability of missing deadlines so that the performance characteristics of the protocol can be identified easier. The deadline of a SRT is calculated from a slack, its start time and its expected execution time:

$$\text{Deadline}(T) = \text{st}(T) + \text{exe}(T) \times \text{SF},$$

where SF is the slack, st(T) is the start time of transaction T and exe(T) is the expected execution time of the transaction T.

Since a SRT is still useful after its deadline, a SRT is allowed to continue its execution after its deadline. It is assumed that a SRT will become totally useless at a certain time after its deadline, and the point of time is called its *final deadline*. At this point, it will be aborted immediately. In the experiments, the final deadline of a SRT is set to be 1.3 times of the slack of the transaction at its arrival time. When there are conflicts with HRT or other SRT after the first deadline, SRT still have to restart.

In the simulation program, the locality of transactions in accessing data items is not modeled explicitly. The reason is that data locality is application-dependent, and it is not our objective to study the performance of the system for a specific application. Instead, a small database is used which allows us to study the effect of hot-spot, in which a small part of the database is accessed frequently by most of the transactions. Another benefit of using a small database is to create a high data contention environment. This helps us understand the performance characteristics of the concurrency control protocols. A small database means that the degree of data contention in the system can be easily controlled by the sizes of the transactions.

The following tables summarize the performance measures for different classes of transactions.

Miss Rate	The number of deadline missing HRT / the total number of HRT generated
-----------	--

Table 3. Performance measures for HRT

Miss Rate	The number of deadline missing SRT / the total number of SRT generated
Abort Rate	The number of useless (aborted) SRT / the total number of SRT generated

Table 4. Performance measures for SRT

Response time	The mean response time of all NRT
---------------	-----------------------------------

Table 5. Performance measures for NRT

Since missing the deadline of a HRT can result in some serious consequence, Miss Rate of HRT is highly preferred to be zero.

5.3 Performance Results

The MRTDBS simulator is implemented in CSIM-18. Statistics are gathered by completing 20,000 soft real-time transactions. The length of the simulation is determined after a number of trial runs using different simulation lengths until stable results are obtained, e.g., the confident levels of the results are within 97%. In the first two sets of experiments, we study the performance of the proposed mixed concurrency control protocol (MCC) as compared with OCC-W50 in a real-time environment where all the data items are assumed to be in the main memory, and the system can support unlimited number of priority levels. In the remaining sets of experiments, we study the impact of different non-real-time issues, e.g., non-real-time disk scheduling such as FCFS, and a limited number of priority levels in the system, on the protocol performance. For a system with a limited number of priority levels, we use the EDREL to map the deadline of a soft real-time transaction to the defined priority levels of SRT.

5.3.1 Performance in an Ideal Real-time Environment

In this set of experiment setting, it is assumed that the EDF scheduling algorithm is used to schedule the use of the CPU and all the data items are resided at the main memory. Figure 2, 3 and 4 show the performance of MCC and OCC-W50 at different soft real-time transaction workloads (using different mean inter-arrival times of soft real-time transactions). Figures 5 to 7 show the results when the hard real-time transaction workload is doubled by reducing their generation periods by a half. (The total system utilization for the hard real-time transactions is about 30% after reducing the generation periods.)

Two important observations from Figure 3 and Figure 6 are: (1) the miss rate of HRT can be kept zero in the MCC except under a very heavy soft real-time workload; and (2) the miss rate of HRT is consistently greater than zero in OCC-W50 even when the soft real-time transaction workload is not heavy. Thus, the MCC can provide a much better guarantee to the deadlines of HRT due to the use of the PCP principles for HRT. The reason for deadline missing of HRT, when the soft real-time transaction workload is heavy, is because of the priority raising of the soft real-time transactions when they are committing. The OCC principles used in the OCC-W50 cannot guarantee any deadlines for HRT because a HRT may be restarted by another HRT due to data conflicts. Therefore, its performance is largely affected by the hard real-time transaction workload as can be observed in Figure 3 and Figure 6.

Although both OCC-W50 and MCC use the optimistic approach for resolving the intra-class data conflicts of SRT, the performance of SRT is consistently much better in MCC than that in OCC-W50, e.g., a smaller miss rate and abort rate (Figure 2 and Figure 3). The difference is more significant at a heavier hard real-time transaction workload, as shown in Figures 5 and 6. The better performance of MCC is due to the different methods used for resolving the inter-class conflicts between soft and hard real-time transactions. Blocking is used in MCC when a SRT wants to access a data item, which is being used by a HRT. In this way, we can reduce the cost and overhead wasted on a SRT, which has to be restarted due to a data conflict with a HRT. Furthermore, raising the priority of a SRT in the validation phase in MCC can further increase its commit probability before deadline. (Although this may affect the performance of HRT, the impact is only significant when the soft real-time transaction workload is very heavy as we can see in Figures 3 and 6.)

As can be observed in Figure 4, the performance of NRT is also marginally better in MCC than in OCC-W50. It is because by blocking the SRT, which have data conflicts with HRT, the total soft real-time transaction workload is lower as only the workload wasted on conflict resolution is smaller. Consequently, the waiting time of NRT for the resources and their probability to be restarted due to data conflicts are smaller since their priorities are lower than SRT. The improvement with MCC is more significant at a heavier hard real-time transaction workload as shown in Figure 7. More SRT will be restarted due to data conflicts with HRT when the hard real-time transaction workload is heavy in OCC-W50.

5.3.2 Performance in a Non-Perfect Real-time Environment

5.3.2.1 Non-Real-time Disk Data Access

In this part of experiments, we first study the impact of non-real-time disk data scheduling on the performance of the protocols. Let parts of the data items be located at the disk and the remaining data items are at the cache buffer. The scheduling of the disk access is assumed to be FIFO since real-time disk scheduling is not common in real-life systems. The cache hit probabilities for soft and non-real-time transactions are both set to be 0.5, e.g., an operation will have 50% probability of finding its required data located at the cache. Since the required data items of HRT are assumed to be known in advance, the data items are assumed to be cached in the main memory. Thus, the cache probability for HRT is 100%.

Figures 8, 9, and 10 show the performance results of soft real-time transaction workloads under such a non-real-time environment. Consistent with our expectation, the performance of the transactions, both real-time and

non-real-time transactions, is much worse than their counterparts at a real-time environment (e.g., the results depict in Figures 2 to 4). It is due to longer data access times. The performance of MCC is consistently better than OCC-W50 for all types of transactions, especially when the soft real-time transaction workload is not very heavy. Surprisingly, the performance improvement for SRT under MCC is even greater than the case in the previous experiments, especially when the soft real-time transactions workload is light. (Compare the results in Figures 8 and 9 with those in Figures 2 and 3). The main reason is that under OCC-W50, it uses the optimistic concurrency control principles to resolve all the data conflicts. Non-real-time transactions may be restarted by soft and hard real-time transactions. The restarted transactions can greatly increase the workload in the disk. Since the scheduling of the disk is FIFO, the disk data access times of the transactions, including the real-time transactions, will also be increased greatly due to heavier disk workloads. Consequently, the deadline missing probability of the soft real-time transactions and even the hard real-time transactions will be affected.

5.3.2.2 Priority Levels

Most conventional operating systems can only support a limited number of priority levels, e.g., 128 priority levels in Solaris. In this set of experiments, we study the impact of priority levels on the protocol performance. We use the EDREL [13] to map the deadlines of SRT to different priority levels supported in the operating system. With this method, different priorities may be mapped to the same priority level for resource scheduling due to an insufficient number of priority levels.

As depicted in Figures 11 and 12, the abort rate and miss rate are higher when the number of priority levels in the system is smaller. This is consistent with our expectation because many SRT are assigned to the same priority level even though their deadlines are different. Thus, a transaction with a closer deadline may have to wait for another transaction with a further deadline in using the CPU.

NRT are less affected by number of priority levels in the system as shown in Figure 13. Their response time is only slightly increased with the number of priority levels under the MCC. It is surprised to see that the response time of NRT decreases gradually with the number of priority levels. Under normal situations, it should be expected that reducing the number of available priority levels should improve their performance, e.g., reducing the mean response time, as they have to wait less time for the CPU (because they may be assigned to the same priorities as some soft real-time transactions). The main reason for the result is that when the number of priority levels is very limited, some SRT are restarted by another

SRT, which have further deadlines but the same priority in using the CPU. This kind of transaction restart can greatly increase the overall system workload. Consequently, the NRT have to wait longer due to a higher system workload. Increasing the priority levels can resolve this problem partially and the overall system performance will be improved, including the NRT.

6 Conclusions

Due to the advances in real-time database technology and the trends of system integration, it becomes more and more common to have a real-time database system with mixed transactions. We call this kind of real-time database systems as *Mixed Real-time Database Systems* (MRTDBS), where different types of transactions, e.g., hard real-time, soft real-time and non-real-time, may co-exist in the systems at the same time. Although different efficient real-time concurrency control protocols had been proposed, they may not be suitable to MRTDBS due to the unique characteristics and performance requirements of different types of transactions.

In this study, we explore strategies to resolve inter-class conflict resolution. We adopt the priority ceiling protocol for concurrency control of hard real-time transactions, the optimistic concurrency control method with wait 50 for soft real-time transactions, and the 2PL for the non-real-time transactions. By exploring the performance characteristics of each individual concurrency control method, we have designed different strategies for inter-class data conflicts resolution and to minimize the impacts of different intra-class concurrency control protocols on each another. Our goal is to guarantee the deadline of hard real-time transactions, to minimize the miss rate of soft real-time transactions, and to minimize the response time of non-real-time transactions. More important, the performance of the protocols can work well in a non-perfect real-time environment, e.g., that with a limited number of priority levels and non-real-time disk data accesses.

Extensive simulation experiments have been done, and the experimental results have shown that the proposed methods can effectively improve the overall system performance and, at the same time, maintain the performance objective of individual transaction type. Furthermore, we have also investigated their performance when some of the data items are resided at the disk, and when the system can only support a limited number of priority levels for scheduling. The results show that, although the performance of the protocol will be affected without the real-time features, it still can achieve a good performance, especially when it is compared with the optimistic methods.

References

- [1] Pang, H., Carey, M.J. and Livny, M., "Multiclass Query Scheduling in Real-time Database Systems", *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 4, pp. 533-551, 1995
- [2] Adelberg, B., H. Garcia-Molina and B. Kao, "Applying Update Streams in a Soft Real-time Database System", in *Proceedings of the 1995 ACM SIGMOD Conference*, pp. 245-256, 1995.
- [3] J. Huang, J. Stankovic & K. Ramamritham, "Priority Inheritance in Soft Real-time Databases", *Journal of Real-Time Systems*, vol. 4, no. 3, 1992, pp. 243-268, 1992.
- [4] A. Datta, S. Mukherjee, P. Konana, I. Viguier, A. Bajaj, "Multiclass Transaction Scheduling and Overload Management in Firm Real-time Database Systems", *Information Systems*, vol.21, no. 1, 1996, pp. 29-54.
- [5] Strosnider, J.K. J.P. Lehoczkym & L. Sha, "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-time Environment", *IEEE Transactions on Computers*, vol. 4, pp.1405-1419, 1995.
- [6] Gallmeister, B.O., Programming for the Real World POSIX.4, O'Reilly & Associates, Inc., 1995.
- [7] Adelberg, B., H. Garcia-Molina and B. Kao, "Emulating Soft Real-time Scheduling Using Traditional Operating System Schedulers", in *Proceedings of IEEE Real-time System Symposium*, 1994.
- [8] Sha, L., Rajkumar, R., Son S.H. and Chang, C.H., "A Real-time Locking Protocol", *IEEE Transactions on Computers*, vol. 40, no. 7, pp. 793-800, 1991.
- [9] Abbott, R. and H. Garcia-Molina, "Scheduling Real-time Transactions: A Performance Evaluation", *ACM Transactions on Database Systems*, vol. 17, no. 3, pp. 513-560, 1992.
- [10] Lam, K.Y., T.W. Kuo and W. H. Tsang, "Concurrency Control for Real-time Database Systems with Mixed Transactions", in *Proceedings of 4th International Workshop on Real-time Computing and Application Systems*, pp. 104-109, 1997.
- [11] Kao, B. & Garcia-Molina, H., "An Overview of Real-time Database Systems", in *Advances in Real-time Computing*, edited by W.A. Halang & A.D. Stoyenko, Springer-Verlag, 1993.
- [12] Sha, L., Rajkumar, R., and Lehoczky, J.P., "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", *IEEE Transactions on Computers*, vol. 39, no. 9, 1990.
- [13] Adelberg, B., H. Garcia-Molina and B. Kao, "Emulating Soft Real-Time Scheduling Using Traditional Operating System Schedulers", in *Proceedings of IEEE Real-Time System Symposium*, 1994.
- [14] Bestavros, A., "Advances in Real-time Database System Research", *ACM SIGMOD Record*, vol. 25, no. 1, 1996.
- [15] Thomas, S., S. Seshadri and J.R. Haritsa, "Integrating Standard Transactions in Real-time Database Systems", *Information Systems*, vol. 21, no. 1, 1996.
- [16] Haritsa, J.R., Carey, M.J. and Livny, M., "On Being Optimistic about Real-Time Constraints", in *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database System*, pp. 331-343, 1990.

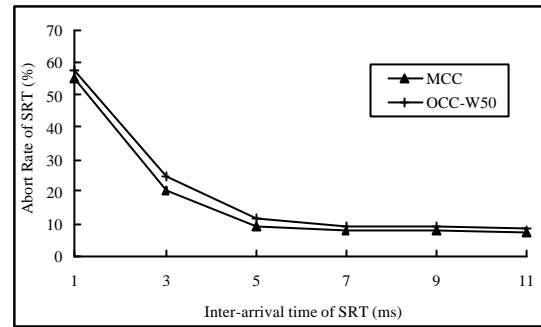


Figure 2: Impact of Inter-arrival time of SRT on Abort Rate of SRT

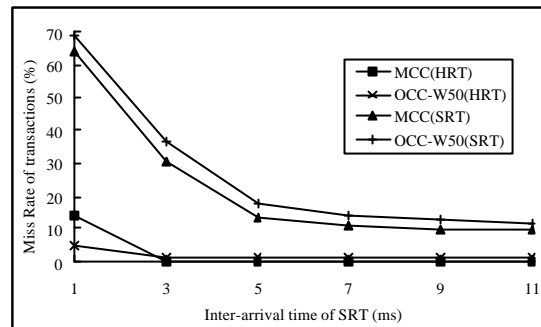


Figure 3: Impact of Inter-arrival time of SRT on Miss Rate of SRT and HRT

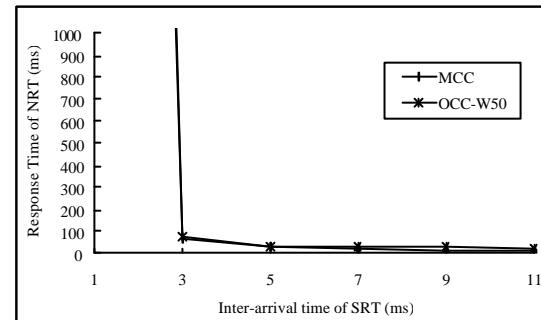


Figure 4: Impact of Inter-arrival time of SRT on Response time of NRT

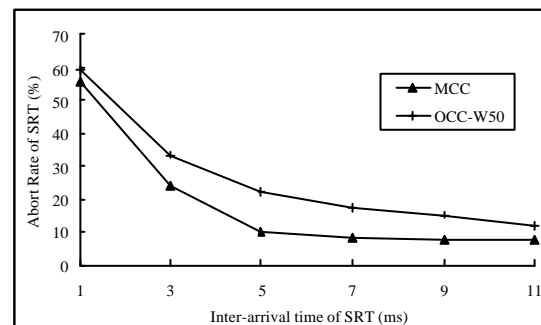


Figure 5: Impact of Inter-arrival time of SRT on Abort Rate of SRT (HRT periods are 30, 37, 45, 55, 62 ms)

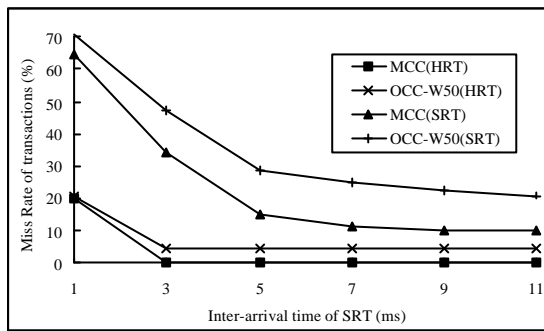


Figure 6: Impact of Inter-arrival time of SRT on Miss Rate of SRT and HRT (HRT periods are 30, 37, 45, 55, 62 ms)

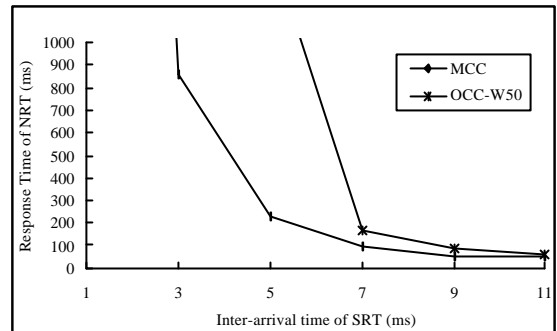


Figure 10: Impact of Inter-arrival time of SRT on Response time of NRT (EDF scheduling for CPU, FCFS scheduling for disk access, cache hit probability = 0.5)

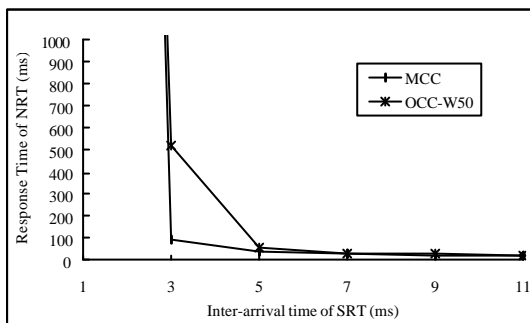


Figure 7: Impact of Inter-arrival time of SRT on Response time of NRT (HRT periods are 30, 37, 45, 55, 62 ms)

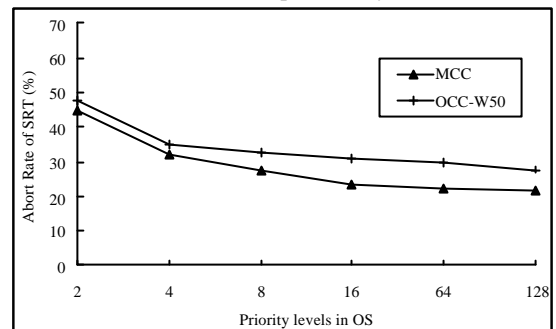


Figure 11: Impact of Priority levels in OS on Abort Rate of SRT (Inter-arrival time of SRT = 5ms)

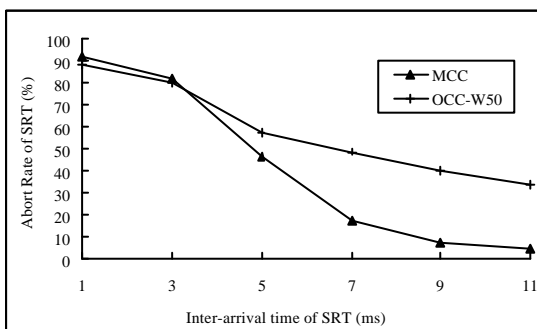


Figure 8: Impact of Inter-arrival time of SRT on Abort Rate of SRT (EDF scheduling for CPU, FCFS scheduling for disk access, cache hit probability = 0.5)

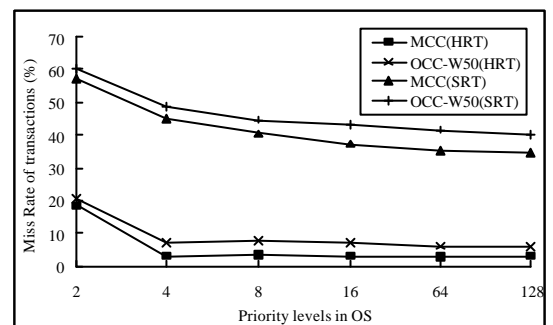


Figure 12: Impact of Priority levels in OS on Miss Rate of SRT and HRT (Inter-arrival time of SRT = 5ms)

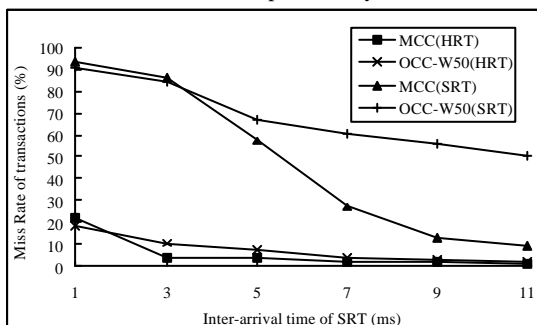


Figure 9: Impact of Inter-arrival time of SRT on Miss Rate of SRT and HRT (EDF scheduling for CPU, FCFS scheduling for disk access, cache hit probability = 0.5)

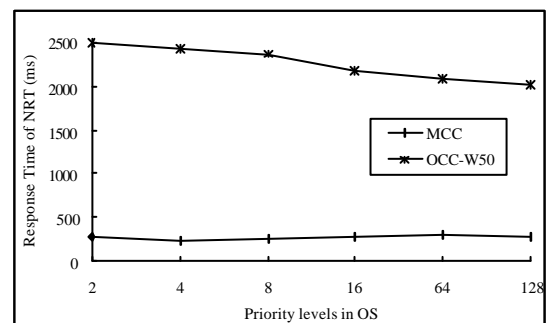


Figure 13: Impact of Priority levels in OS on Response time of NRT (Inter-arrival time of SRT = 5ms)