

Strategies for Resolving Inter-Class Data Conflicts in Mixed Real-time Database Systems^{†*}

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
Fax: 852-2788-8614

Department of Computer Science and
Information Engineering²
National Taiwan University
Taipei, Taiwan 106, ROC
Email: ktw@csie.ntu.edu.tw

Abstract

Although many efficient concurrency control protocols have been proposed for real-time database systems, they are mainly designed for the systems with a single type of real-time transactions. Their performance objective is usually to minimize the number of deadline missing of soft real-time transactions or to guarantee the deadline satisfaction of hard real-time transactions. Due to the very different performance requirements of hard and soft real-time transactions, existing real-time concurrency control 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. In this paper, we propose strategies for resolving data conflicts between different types of transactions in a MRTDBS so that the performance requirements of each individual transaction type can be satisfied 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, which has been shown to give a better performance than the lock-based protocols for soft and firm real-time transactions. The performance of our proposed conflict resolution methods has also been investigated in a more realistic environment in which the number of priority levels supported in the system is limited, and some of the data may be resided on the disk.

Keywords: real-time database systems, concurrency control and real-time transaction scheduling

1 Introduction

The research in *Real-time Database Systems (RTDBS)* had 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

[†] A preliminary version of the paper will appear in the Euromicro Conference on Real-time Systems, Sweden, June 2000.

* The work described in this paper was partially supported by a grant from the Research Grants Council of the

on the consequences of missing their deadlines (Bestavros, 1996; Sha et al., 1991; Yu et al. 1994). 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 abort of the transaction as its value is totally lost after its deadline expires.

Concurrency control protocols designed for RTDBS with a single type of transactions may not fit the very different performance requirements of different types of real-time transactions. On the other hand, different types of real-time transactions may not satisfy the specific constraints of the protocols designed for a single type of transactions. For example, the well-known priority ceiling protocol (PCP) (Sha et al., 1990; Sha et al., 1991) (designed for HRT) requires a static system that is usually not true for a system with SRT. The Higher Priority Two Phase Locking Protocol (HP-2PL) (Abbott and Garcia-Molina, 1989; Abbott and Garcia-Molina, 1992), which is designed for SRT, cannot guarantee the deadlines of HRT or even be suitable to a RTDBS that consists of mixed SRT and non-real-time transactions (NRT). The response times of NRT may be greatly affected because of the restart policy of HP-2PL. A NRT may be repeatedly restarted due to data conflicts with SRT.

We call a RTDBS with different types of transactions a *Mixed RTDBS (MRTDBS)*. An example of MRTDBS is the programmed stock trading system (Adelberg et al., 1995; Kao and Garcia-Molina, 1993). 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 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 are preferred 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, is that real-time concurrency control protocols often assumes an unlimited number of priority levels and a memory-resident database. Obviously, a realistic database system may only have a limited number of priority levels for the scheduling of transactions (Gallmeister, 1995), and a database is often disk-resident, instead of main-memory-resident. Priority mapping mechanisms are needed (Adelberg et al. 1994) to convert the priorities of the transactions derived from the adopted priority scheduling algorithm to the priority levels supported by the system (Adelberg et al. 1994). Little work has been

done in exploring how the priority mapping algorithms affect the performance of real-time concurrency control protocols. The problem is further complicated when joint scheduling of different types of transactions is considered. The real-time scheduling of MRTDBS may become further complicated when a disk-resident database is considered. Most of the previous studies on concurrency control protocols for RTDBS often 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 methods that have been shown to be effective for a single type of transaction to resolve intra-class data conflicts and propose strategies for inter-class data conflict resolution. Here inter-class data conflicts refer to the data conflicts between different types of transactions, and intra-class data conflicts refer to the data conflicts among transactions belonging to the same type. We shall also evaluate the impacts of different practical issues of the system environment on real-time concurrency control, e.g., a limited number of priority levels and an unpredictable disk access delay, in terms of 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 conclusion.

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 (Abbott and Garcia-Molina, 1989; Abbott and Garcia-Molina, 1992; Haritsa et al. 1990; Haritsa et al., 1992; Huang et al. 1992; Sha et al., 1990; Sha et al., 1991; Ulusoy and Buchmann, 1998). Most lock-based real-time concurrency control protocols follow from the two phase locking scheme (2PL) (Bernstein et al., 1987). For example, the Higher Priority Two Phase Locking Protocol (HP-2PL) (Abbott and Garcia-Molina, 1989; Abbott and Garcia-Molina, 1992) may restart a lower priority transaction if it is in conflict with any lock request of a higher priority transaction, where the transactions lock data items in a 2PL fashion. 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 (Huang et al., 1992; Sha et al., 1990) is suggested to resolve the priority inversion problem in 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 the former. The priority ceiling protocol (PCP) (Sha et al., 1991) 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, where the priority ceiling of a data item is the maximum priority of all transactions which may lock the data item.

Many real-time concurrency control protocols are based on the optimistic concurrency control (OCC) methods, e.g., (Haritsa et al., 1990; Bernstein et al., 1987). 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 past decades. 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 its 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. The earliest deadline first (EDF) priority assignment method (Liu and Layland, 1976) is used very often in RTDBS research. Under the EDF, the transaction with the closest deadline will be assigned to have the highest priority. The least slack time first priority assignment method is also often used, where the slack of a transaction equals to the difference of its deadline and the sum of the current time and the remaining execution time of the transaction. The highest priority is assigned to the transaction with the smallest slack time in the system. On the other hand, a fixed priority assignment method such as the rate monotonic scheduling method (Liu and Layland, 1976) is used in many RTDBS research as well, e.g., (Sha et al., 1991).

Although the above methods are shown being 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 (Gallmeister, 1995). Although the priority mapping methods might have a significant impact on the performance of the concurrency control protocols, it is completely lack of previous work to investigate how they affect the performance of various types of concurrency control protocols in a RTDBS.

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

The reduced ceiling protocol (RCP) (Lam et al., 1997) 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-preemptively. Another approach in mixed scheduling of transactions is proposed in (Kim and Sang, 1996). 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 (Thomas et al., 1996), 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 method aims at a more general system situation that consists of all types of transactions, HRT, SRT and NRT. To our best knowledge, this is the first paper on concurrency control for a RTDBS with all types of real-time and non-real-time transactions. 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 on 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 (Abbott and Garcia-Molina, 1989; HSR92, DMK96, HCL90, SRL91]. 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. We adopt a two-layer scheduler: global and local schedulers. The first layer, which contains the global scheduler, is responsible for resolving inter-class data conflicts. There are three local schedulers at the second layer, and each local scheduler is responsible for resolving the intra-class data conflicts for each class of transactions. Whenever the global scheduler receives an operation

from a transaction, the global scheduler will check for inter-class conflict and forward the operation to the individual local scheduler for the class of the transaction.

In the model, we assume that the schedulers maintain a global lock table for detecting both intra-class and inter-class conflicts. When the global scheduler receives a lock request from an operation, it will try to set the appropriate lock for the operation. If there is a lock, which is already set by another transaction of a different class, on the same entry, and the lock is incompatible to the requesting lock, then the adopted conflict resolution strategy will be invoked to resolve the lock conflict. For intra-class conflict resolution, the local schedulers might also resolve lock conflicts based on the global lock table in a similar way when it receives a lock request from an operation. After setting the required lock for an operation, the global or the local scheduler will forward the operation to the resource manager (RM) that handles all the data access and recovery. RM controls the cache and disk access.

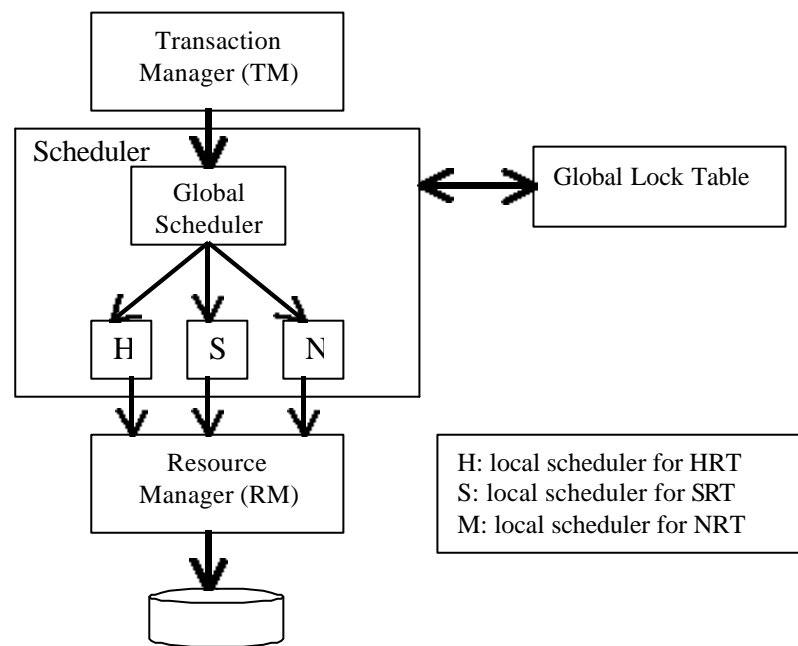


Figure 1: The MRTDBS Model

There are three types of transactions in the system: hard real-time (HRT), soft real-time (SRT) and non-real-time (NRT) transactions. We assume that HRT have higher priorities than SRT and NRT in the system because of their high criticality. The priorities of SRT are also assumed to be higher than NRT. It is assumed that all transactions 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 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 their executions after their deadlines expire. 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 for the transaction 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-Class 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.

In MCC, the Priority Ceiling Protocol (PCP)¹ is adopted for resolving the intra-class data conflicts among HRT. The reason of choosing PCP is because PCP can guarantee the schedulability of HRT. OCC-W50 is used to schedule SRT because OCC-W50, as an optimistic method, has been shown being more suitable to schedule soft real-time transactions than many lock-based methods, due to more number of fruitful restarts. Amongst the various conflict resolution strategies for the optimistic-based method for RTDBS, e.g., sacrifice and OCC-Wait (Haritsa et al. 1990), OCC-W50 gives the best performance, as shown in previous study. Under the OCC-W50, a validation test is performed to ensure that the resulted schedule is serializable. The validating transaction is blocked until most of the SRT which are in conflict with the validating transaction have lower priorities than the validating transaction. Once a validating SRT passes its validation test, all the conflicting SRT are restarted. We adopt the two phase locking (2PL) protocol (Bernstein et al., 1987) as the basic concurrency control protocol for NRT. The reason of choosing 2PL is because 2PL is one of the most popular protocols for conventional database systems. Its implementation and performance are well-known to the system designer. Under 2PL, a transaction cannot lock any data item once it releases any lock. In the following section, we shall consider strategies for inter-class conflict resolution.

¹ An important variant of the PCP is the read/write PCP (Sha et al., 1991). 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 will remain if we adopt the read/write PCP instead of the PCP for the concurrency control of hard real-time transactions.

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, which are scheduled using OCC-W50, should not cause any blocking or transaction restarts to other SRT before its validation in order to benefit from 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. The following table summarizes the conflict resolution strategies for different types of inter-class and intra-class data 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 the SRT is raised to be higher than the priorities of HRT when the SRT is at the validation phase

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

*** A deadlock resolution algorithm must be incorporated.

Table 1: Intra-Class and Inter-Class Conflict Resolution Methods

As shown in the above table, if a HRT tries to lock a data item, which is accessing by a SRT or a NRT in a conflicting mode, the system will restart the conflicting SRT or NRT. The purpose is to guarantee the deadlines of the HRT. However, blocking is used for the case 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. 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 later in its execution. On the other hand, the 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 if it can pass the validation phase. Delaying the conflict detection of SRT until the validation phase can optimize the number of fruitful restarts of NRT because the SRT may be restarted 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 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. (Note that the transaction time-stamp method may also be used for that purpose (Bernstein et al., 1987).) The system maintains the read/write data sets of every executing SRT for conflict resolution following the optimistic principles. A data item is included in the read/write data sets of a SRT if it has been read/written by the SRT. 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., PR-lock, (or pre-write locked, i.e., PW-lock) by the SRT. In order to prevent the blocking by other executing SRT due to the locking, all pre-locks are compatible to each other. During the validation phase, a SRT converts all its pre-locks to exclusive locks, i.e., the V-locks, which are used to prevent HRT and NRT to access the data item until the completion of the validation phase and the write phase since they have to be performed in a critical section. There are two kinds of locks for HRT: hard read lock (HR-lock) and hard write lock (HW-lock). There are also two kinds of locks (N-locks) for NRT: non-real-time read lock (NR-lock) and non-real-time write lock (NW-lock). The compatibility of the locks is shown in Table 2.

Note that when a validating transaction wants to convert a P-lock to a V-lock, 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 after all the P-locks have been converted to V-locks according to the principles defined in the previous sub-section. Since only

one SRT can be in the validation and write phase as required by the optimistic method (Bernstein et al., 1987), V-locks are not compatible with each other.

		Requester/Validating transaction (SNT only)						
		HR-lock	HW-lock	PW-lock	PR-lock	V-lock	NR-lock	NW-lock
Holder	HR-lock	✓	✗	✗	✓	✓	✓	✗
	HW-lock	✗	✗	✗	✗	✓	✗	✗
	PR-lock	✓	✓	✓	✓	✓	✓	✗
	PW-lock	✓	✓	✓	✓	✓	✗	✗
	V-lock	✗	✗	✗	✗	✗	✗	✗
	NR-lock	✓	✓	✓	✓	✓	✓	✗
	NW-lock	✓	✓	✓	✓	✓	✗	✗

✓: 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 Discussions

In MCC, we use PCP, OCC-W50, and 2PL for intra-class conflict resolution. All of these methods have been shown being able to maintain serializable executions in previous work. When inter-class conflicts occur, we use either blocking or transaction restart to resolve the conflicts. A transaction, belonging to a higher priority class, e.g., hard real-time transaction, will always be given a preference in accessing a data item if it is in conflict with a transaction in a lower priority class, e.g., non-real-time transactions. The non-real-time transaction will be restarted when a hard real-time transaction wants to commit. Therefore, any serializable schedules amongst transactions belonging to different classes are non-cyclic, e.g., serializable. It is because a transaction in a higher priority class can not be dependent on the results produced by conflicting transactions in lower priority classes which execute at the same time. By following from the nice properties of PCP, OCC-W50, and 2PL, all the transaction executions under MCC are serializable.

In MCC, we use PCP and OCC-W50 for concurrency control of hard real-time transactions and soft real-time transactions, respectively. Since both PCP and OCC-W50 are deadlock-free, it is not possible to have a deadlock cycle formed amongst hard real-time transactions (or amongst soft real-time transactions). Furthermore, it is impossible to have a deadlock cycle between transactions belonging to different classes. If there is a data conflict between a soft real-time transaction and a non-real-time transaction, the conflict will always be resolved by transaction restart or by blocking of the non-real-time transaction. As a result, soft real-time transaction will not be blocked by non-real-time transactions. Thus, no deadlock cycle can be formed between soft real-time and non-real-time

transactions. When there is any data conflict between a hard real-time transaction and a non-real-time transaction, the non-real-time transaction will always be blocked or restarted. As a result, no hard real-time transaction can be blocked by non-real-time transactions. Thus, no deadlock cycle can be formed between hard real-time and non-real-time transactions. The only possibility of having a deadlock is that amongst non-real-time transactions. This is resolved in MCC by adopting a conventional deadlock resolution method, e.g., a wait-for-graph for 2PL.

4.5 Priority Mapping Problem

Previous works in real-time concurrency control usually assume an unlimited number of priority levels in the system. As a result, when these protocols are applied to realistic systems, we have to adopt a priority mapping algorithm to convert the *original priority*, which is ideally derived from its deadline and criticality, to the priority level (called *mapped priority*) supported by the system. Since we will have two kinds of priorities for each transaction: original priority and mapped priority, there can be two choices in using the priorities for conflict resolution. We may choose to use the mapped priorities for resource scheduling and the original priorities for data conflict resolution. We called this method the *Different Priority Method (DPM)*. Note that the original priorities cannot be used for transaction scheduling due to insufficient priority levels in the system. However, we can also use the mapped priorities for both resource and data conflict resolution. We call this method the *Same Priority Method (SPM)*.

These two alternatives (DPM and SPM) can have very different impacts on different types of concurrency control protocols. If the mapping is perfect, the SPM should be a good choice since the scheduling of transactions in using the resources and data accessing will follow the urgency and criticality of the transactions. However, it is not easy to find a perfect way for priority mapping. If the number of priority levels in the system is limited, transactions with different original priorities may be mapped to the same priority level. Thus, it is possible that a higher priority transaction is restarted by a lower priority transaction if they are mapped to the same priority level. As a result, the restart overhead might be higher under a limited number of available priority levels. In (Adelberg et al. 1995), several priority mapping methods are proposed and studied, e.g., earliest deadline relative (EDREL)², earliest deadline absolute (EDABS), and least slack relative (LSREL). As shown in (Adelberg et al., 1995), EDREL, in general, gives a better performance than the other methods in terms of the deadline miss rate. Furthermore, it is difficult to determine the slack of a soft real-time transaction due to many factors, such as dynamic workloads. In this paper, we choose the EDREL in our study to map the original priority level of a transaction to a priority level supported by

² Although other methods such as least slack relative is also suggested in (Liu and Layland, 1976), it is not suitable to our study because it requires the information of the expected execution time of a

the system. Under EDREL, a transaction's priority is determined at the transaction arrival time by a simple function: $priority(T) = (dl(T) - ar(T)) / ts$, where $dl(T)$ is the deadline of a transaction T , $ar(T)$ is the arrival time of T , and ts is a tuning parameter. In (Adelberg et al., 1994), it is suggested that ts be set as

$$ts = \frac{\mathbf{m} + 3\mathbf{S} + S_{\max}}{n}$$

where

\mathbf{m} and \mathbf{S} are the average and standard deviation of transaction execution time, respectively,

S_{\max} is the maximum amount of transaction's slack, and

n is the number of priority levels supported by the static priority scheduler.

On the other hand, the DPM may not be suitable to some restart-based protocols, e.g., HP-2PL, because the restart of a transaction may occur repeatedly. Let us use the following example to illustrate the problem: Consider two transactions T_i and T_j . Assume that the original priority of T_i is higher than that of T_j , but T_i and T_j have the same mapped priority. As a result, they will share the CPU cycles in the system on an equal base, and T_j will affect the waiting time of T_i is in using the CPU. While T_i is executing, it may have data conflict with T_j . Under HP-2PL, T_j will be restarted since T_i 's original priority is higher. However, when T_j restarts its execution from the beginning, it will share the CPU with T_i again if T_i has not yet finished. Consequently, it will increase the response time of T_i . Even worse, it is possible that T_j may be restarted again and again by T_i for several times. Thus, the impacts of T_j 's execution on T_i can be very significant, and the resource workload may be greatly increased as well.

Note that the above repeated restart problem will not occur in an optimistic method because once a transaction has passed the validation phase, it will commit and will not be restarted by any other transactions. In the next section, we will study the performance impacts of the priority mapping algorithm, e.g., EDREL, with DPM and SPM on the performance of the concurrency control protocols for MRTDBS, when there are a limited number of available priority levels in the system.

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 the optimistic concurrency control with wait-50 (OCC-W50) is implemented for comparison. The reason to compare our protocol, MCC, with OCC-W50, instead of transaction. This assumption is usually not true for SRT.

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 a MRTDBS because its assumptions are not suitable to soft real-time and non-real-time transactions. Note that the purposes of the simulation study 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, including the HRT and NRT, have to go through three phases: read, validation and write phases. HRT are assigned the highest priority in accessing data items while all the NRT are given the same lowest priority in the system. The rate monotonic algorithm is used to assign priorities to 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 strategy used to resolve any data conflicts between a HRT and a SRT (or a NRT) in the modified version of OCC-W50 follows the OCC-sacrifice principle, instead of the original OCC-W50 principle. For example, when a SRT (or a NRT) is in the validation, it will be restarted if it is in conflict with a HRT. For the conflict resolution between a SRT and a NRT, and the intra-class conflicts of individual transaction types, the OCC-W50 principle will be followed.

5.2 Model Parameters and Performance Measures

The following table summarizes the baseline values of the important parameters. Some of the baseline values are determined based on those adopted in previous studies (Thomas et al., 1996), e.g., the database size and the write operation probability, and others are set based on the characteristics of the typical applications of RTDBS. For example, most of the transactions in a RTDBS are short transactions, especially hard real-time transactions, that often consist of only a few operations. The processing time of an operation should not be very long, and it is usually within a few milliseconds since the computation should not be very complex. The processing of a disk access is set around double of the processing time of a main memory data access in order to balance the workload between the CPU and the disks. Note that the number of disks is twice of the number of CPU's in our simulation. In the first part of the simulation experiments, we assume a main-memory-resident database. In other words, all the read and write operations are performed on main-memory-resident data.

Parameters	Baseline Values
Database size	1000 data items

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 CPU's 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
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 such as follows:

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

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

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

SF (the slack), which is a random variable, is uniformly distributed between two values. $\text{st}(T)$ is

the start time of transaction T, and $exe(T)$ is the expected execution time of the transaction T.

Since a SRT is still valuable after its deadline, a SRT is allowed to continue its execution after its deadline, although the value of the SRT might greatly decrease with time. 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 as 1.3 times of the slack of the transaction at its arrival time. While a SRT is executing, it may be restarted due to a data conflict with a HRT or another SRT. If it has not passed its final deadline, it will be restarted from the beginning. Otherwise, it will be aborted immediately.

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 such that we can understand the performance characteristics of the concurrency control protocols, especially when the data conflict probability is higher. A small database is also a convenient way to manage the degree of data contention in the system because the data contention degree can be easily controlled by the sizes of transactions.

The following table summarizes the performance measures for different kinds of transactions.

Miss rate of hard real-time transactions	number of deadline missing of HRT / total number of HRT generated
Miss rate of soft real-time transactions	number of deadline missing of SRT / total number of SRT generated
Abort rate of soft real-time transactions	number of useless (aborted) SRT / total number of SRT generated
Restart rate of soft real-time transactions	number of restarts of SRT / total number of SRT generated
Response time of non-real-time transactions	Σ response time of all completed NRT/ total number of NRT completed
Restart rate of non-real-time transactions	number of restarts of NRT / total number of NRT completed

Table 2. Performance Measures

Since any deadline violation of a HRT may result in some serious consequence, the miss rate for 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 a simulation run is determined after a number of trial runs using different simulation lengths until stable results are obtained. Although the length of a simulation is defined in terms of the number of soft real-time transactions completed, the number of completed hard real-time transactions and that of completed non-real-time transactions are directly proportional to the number of completed soft real-time transactions. The simulation length is chosen when the statistics for all three classes of transactions are stable such that the reported statistics are reliable for all three classes of transactions. 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 an unlimited number of priority levels. In the remaining parts 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 deadlines of soft real-time transactions to the available priority levels in the system.

5.3.1 Performance in an Ideal Real-time Environment

In this part of experiments, it is assumed that the EDF scheduling algorithm is used to schedule transactions, and all the data items reside 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 remains 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 under OCC-W50 even when the soft real-time transaction workload is not heavy. On the other hand, 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 violations of HRT, when the soft real-time transaction workload is heavy, is because of the priority raising of the soft real-time transactions (at the validation and write phases) when they are committing. Note that the blocking time of SRT on HRT can be easily quantified in the schedulability analysis of HRT by having a blocking time in the PCP schedulability analysis formula. 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 observed in Figure 3 and Figure 6.

Although both OCC-W50 and MCC adopt 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 fact that different methods are used to resolve the inter-class conflicts between soft and hard real-time transactions. Blocking exists 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 overheads 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 its 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 shown in Figure 4, the performance of NRT is 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 because the workload wasted on conflict resolution is smaller. Consequently, the waiting time of NRT for the resources and their restart probability, due to data conflicts, are smaller since their priorities are lower than those of SRT. The performance improvement achieved by MCC is more significant at a heavier hard real-time transaction workload, as shown in Figure 7. More SRT is 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 simulated protocols. Let parts of the data items be located at the disk, and the rest are at the cache buffer. The scheduling of the disk access is assumed being FIFO since real-time disk scheduling is not common in many real-life systems. The cache hit probabilities for soft and non-real-time transactions are both set as 0.5. For example, an operation will have a 50% probability in finding its required data being located at the cache. Since the required data items of HRT are assumed being 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 a non-real-time environment. It is consistent with our expectation that the performance of the

transactions, regardless of real-time or non-real-time, is much worse than their counterparts in a real-time environment (e.g., the results shown in Figures 2 to 4). It is due to longer data access times. However, the performance of MCC is consistently better than that of 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 of the disk. Since the scheduling of the disk is FIFO, the disk data access times of the transactions, including 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.

Figures 11 and 12 depict the miss rates of SRT and HRT, and the response time of NRT, respectively, when the cache hit probability is varied. Consistent to the results in Figure 9 and Figure 10, the performance of MCC is consistently better than that of OCC-W50. As expected, increasing the cache hit probability improve the system performance, e.g., lower miss rate and smaller response time. When the cache hit probability is low, the performance of both protocols is poor due to the long disk data access delay times. The problem is more serious to HRT under OCC-W50. Its miss rate is much higher when the cache hit probability is low. When the cache hit probability is high, the system enjoy a good performance due to small overhead in disk access. Both protocols give a good performance.

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 and 32 levels in Windows NT. In this part of experiments, we aim at studying the impacts of priority levels on the protocol performance. We will also investigate their performance, when the SPM and DPM are used. We will use the EDREL to map the deadlines of SRT to the priority levels supported by 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 available priority levels in the system.

Figures 13 to 16 show the impacts of the available priority levels on the performance of MCC and OCC-W50, when SPM and DPM are used. As shown in Figure 13, the abort rates of MCC are lower than that of OCC-W50. For both MCC and OCC-W50, the abort rate is lower under the DPM than that under the SPM, especially when the number of available priority levels in the system is

small. It is because many SRT are assigned 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 farther deadline in using the CPU. The problem is more serious to HRT, as shown in Figure 14. Although the DPM has the problem of repeated restart (discussed in Section 4.4), its impacts on the system performance is less significant to OCC-W50 and MCC because most of the data conflicts under these two protocols are resolved when a transaction wants to commit. As shown in Figure 16, NRT are less affected by the number of available priority levels in the system and by the method used for priority assignment for data conflict resolution. Their response time remains similar when different numbers of available priority levels are used.

6 Conclusions

Due to the advances in real-time database technology and the trend in 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 *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 have 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 data conflicts. We adopt the priority ceiling protocol for the concurrency control of hard real-time transactions, the optimistic concurrency control with wait 50 for soft real-time transactions, and the well-known 2PL for 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 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 importantly, the performance of the protocols can also 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 access.

Extensive simulation experiments have been done, and the experimental results have shown that the proposed strategies can effectively improve the overall system performance and, at the same time, maintain the performance objective of each individual transaction type. Furthermore, we have also investigated their performance when some of the data items reside at the disk, and when the system can only support a limited number of priority levels for transaction scheduling. The results show that, although the performance of the proposed strategies can be affected with the “non-real-time” features

of system support, they still can achieve a pretty good performance, especially compared with the optimistic methods.

References

- Abbott, R. and H. Garcia-Molina. 1989. Scheduling Real-time Transactions with Disk Resident Data. In: *Proceedings of the 15th VLDB Conference*, pp. 356-396.
- Abbott, R. and H. Garcia-Molina. 1992. Scheduling Real-time Transactions: A Performance Evaluation. *ACM Transactions on Database Systems*, 17(3), 513-560.
- Adelberg, B., H. Garcia-Molina and B. Kao. 1994. Emulating Soft Real-Time Scheduling Using Traditional Operating System Schedulers. In: *Proceedings of IEEE Real-Time System Symposium*, pp. 292-298.
- Adelberg, B, H. Garcia-Molina and B. Kao. 1995. Applying Update Streams in a Soft Real-time Database System. In: *Proceedings of the 1995 ACM SIGMOD Conference*, pp. 245-256.
- Bestavros, A. 1996. Advances in Real-time Database System Research. *ACM SIGMOD Record*, 25(1), 3-7.
- Bernstein, P.A., Hadzilacos, V., and Goodman, N. 1987. *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Reading, Massachusetts.
- Bestavros, A. and S. Nagy. 1996. Value-Cognitive Admission Control for RTDBS. In: *Proceedings of Real-time Systems Symposium*, pp. 230-239.
- Datta, A., Mukherjee, S., Konana, P., Viguier, I., Bajaj, A. 1996. Multiclass Transaction Scheduling and Overload Management in Firm Real-time Database Systems. *Information Systems*, 21(1), 29-54.
- Gallmeister, B.O. 1995. *Programming for the Real World POSIX.4*, O'Reilly & Associates, Inc..
- Haritsa, J.R., Carey, M.J., and Livny, M. 1990. On Being Optimistic about Real-Time Constraints. In: *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 331-343.
- Haritsa, J.R., Carey M.J. & Livny, M. 1992. Data Access Scheduling in Firm Real-time Database Systems. *Journal of Real-Time Systems*, 4(3), 203-242.
- Huang, J., Stankovic, and Ramamritham, K. 1992. Priority Inheritance in Soft Real-time Databases", *Journal of Real-Time Systems*, 4(3), 243-268.
- Kao, B. & Garcia-Molina, H. 1993. An Overview of Real-time Database Systems. *Advances in Real-time Computing*, edited by W.A. Halang & A.D. Stoyenko, Springer-Verlag.

- Kim, Y.K., and Son, S.H. 1996. Supporting Predictability in Real-time Database Systems. In: *Proceedings of IEEE Real-Time Technology and Applications Symposium*, Massachusetts, pp. 38-48.
- Lam, K.Y., T.W. Kuo and W. H. Tsang. 1997. 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.
- Liu, C.L., and Layland, J.W. 1976. Scheduling Algorithms for Multi-programming in a Hard-Real-Time Environment. *Journal of ACM*, 20(1), 41-64.
- Sha, L., Rajkumar, R., Son S.H. and Chang, C.H. 1991. A Real-time Locking Protocol. *IEEE Transactions on Computers*, 40(7), 793-800.
- Sha, L., Rajkumar, R., and Lehoczky, J.P. 1990. Priority Inheritance Protocols: An Approach to Real-Time Synchronization", *IEEE Transactions on Computers*, 39(9), 1175-1185.
- Thomas, S., S. Seshadri and J.R. Haritsa. 1996. Integrating Standard Transactions in Real-time Database Systems", *Information Systems*, 21(1), 3-28.
- Ulusoy O. and A. Buchmann. 1998. A Real-time Concurrency Control Protocol for Main-Memory Database Systems. *Information Systems*, 23(2), 109-125.
- Yu, P.S., Wu, K.L., Lin, K.J. and Son, S.H. 1994. On Real-time Database: Concurrency Control and Scheduling. *Proceeding of IEEE*, 28(1), 140-157.

MCC: Mixed Concurrency Control
HRT: hard real-time transactions
NRT: non-real-time transactions
SPM: Same Priority Method

OCC-W50: Optimistic Concurrency Control with Wait 50
SRT: soft real-time transactions
EDF: earliest deadline first
DPM: Different Priority Method

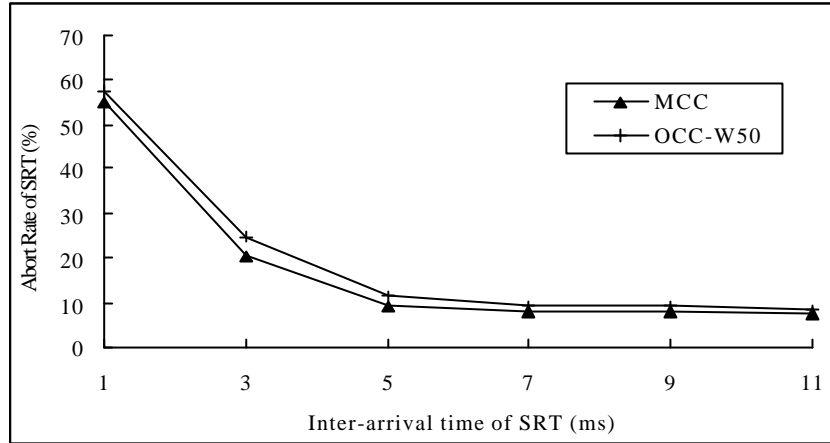


Figure 2: Impact of Inter-arrival time of SRT on Abort Rate of SRT (EDF scheduling for CPU, main-memory data items, HRT periods = 60, 75, 90, 110 and 125 ms)

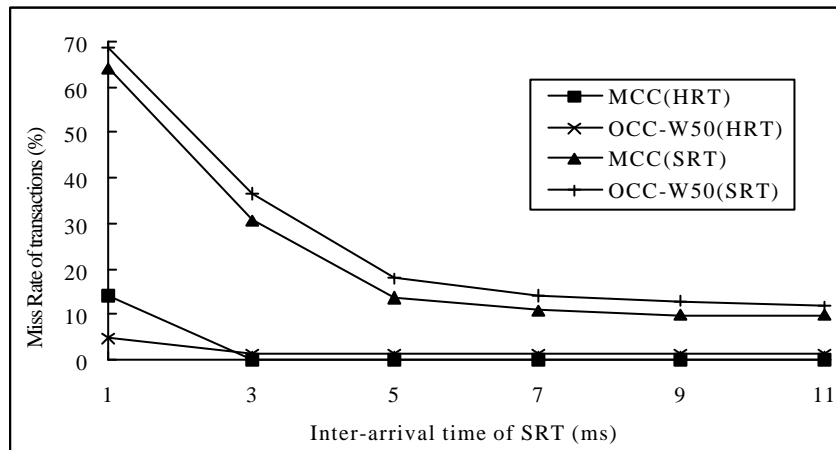


Figure 3: Impact of Inter-arrival time of SRT on Miss Rate of SRT and HRT (EDF scheduling for CPU, main-memory data items, HRT periods = 60, 75, 90, 110 and 125 ms)

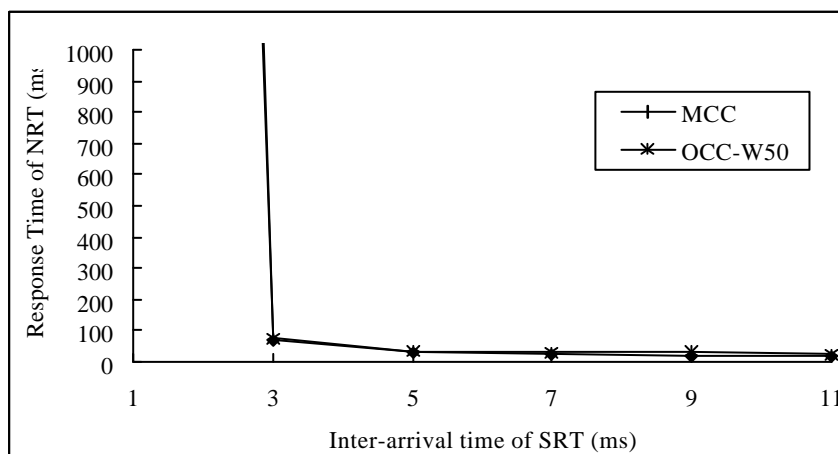


Figure 4: Impact of Inter-arrival time of SRT on Response time of NRT (EDF scheduling for CPU, main-memory data items, HRT periods = 60, 75, 90, 110 and 125 ms)

MCC: Mixed concurrency control
HRT: hard real-time transactions
NRT: non-real-time transactions
SPM: Same Priority Method

OCC-W50: Optimistic Concurrency Control with Wait 50
SRT: soft real-time transactions
EDF: earliest deadline first
DPM: Different Priority Method

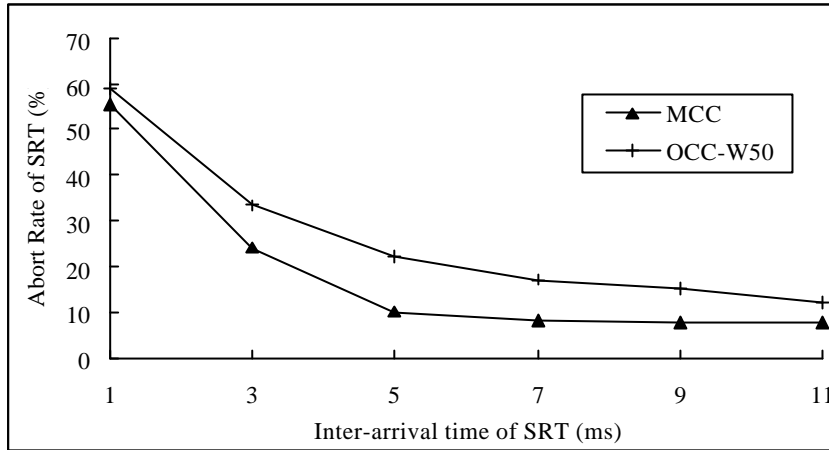


Figure 5: Impact of Inter-arrival time of SRT on Abort Rate of SRT (EDF scheduling for CPU, main-memory data items, HRT periods = 30, 37, 45, 55 and 62 ms)

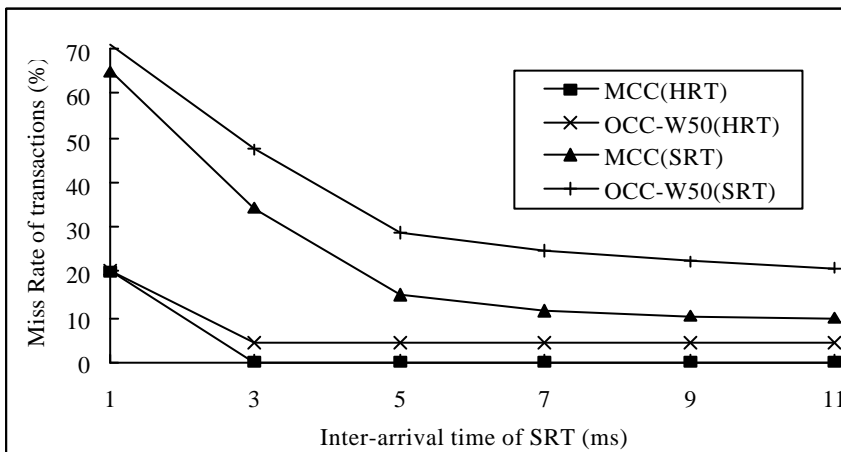


Figure 6: Impact of Inter-arrival time of SRT on Miss Rate of SRT and HRT (EDF scheduling for CPU, main-memory data items, HRT periods = 30, 37, 45, 55 and 62 ms)

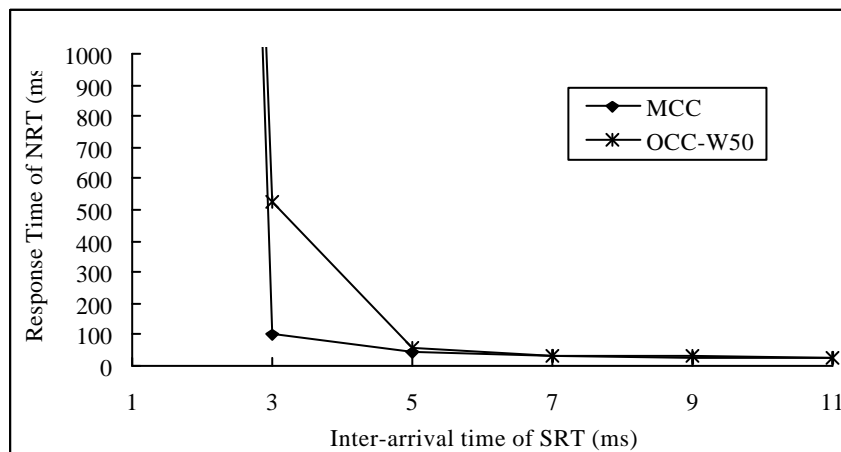


Figure 7: Impact of Inter-arrival time of SRT on Response time of NRT (EDF scheduling for CPU, main-memory data items, HRT periods = 30, 37, 45, 55 and 62 ms)

MCC: Mixed concurrency control
HRT: hard real-time transactions
NRT: non-real-time transactions
SPM: Same Priority Method

OCC-W50: Optimistic Concurrency Control with Wait 50
SRT: soft real-time transactions
EDF: earliest deadline first
DPM: Different Priority Method

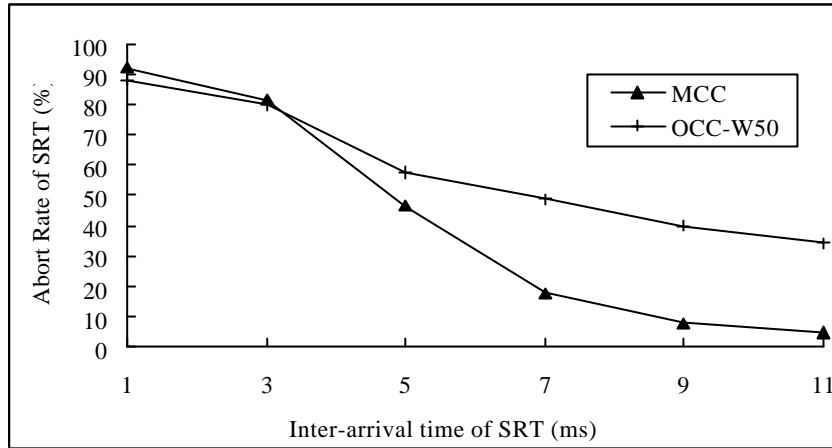


Figure 8: Impact of Inter-arrival time of SRT on Abort Rate of SRT (EDF scheduling for CPU, FCFS scheduling for disk access, cache hit probability = 0.5)

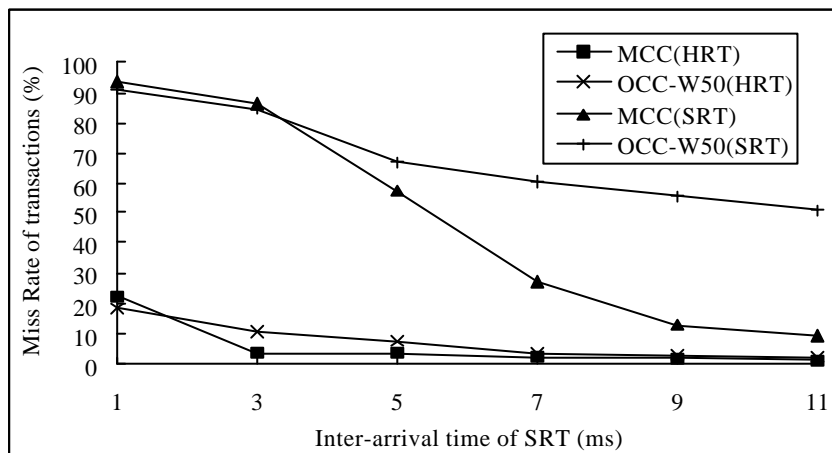


Figure 9: Impact of Inter-arrival time of SRT on Miss Rate of SRT and HRT (EDF scheduling for CPU, FCFS scheduling for disk access, cache hit probability = 0.5)

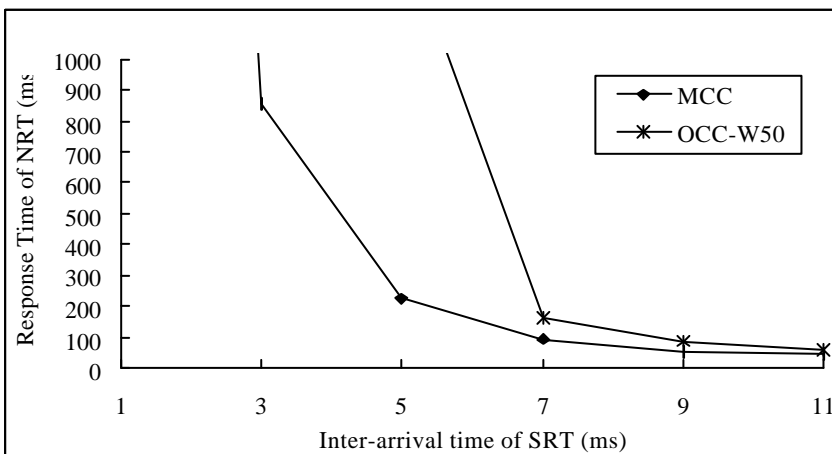


Figure 10: Impact of Inter-arrival time of SRT on Response time of NRT (EDF scheduling for CPU, FCFS scheduling for disk access, cache hit probability = 0.5)

MCC: Mixed concurrency control
HRT: hard real-time transactions
NRT: non-real-time transactions
SPM: Same Priority Method

OCC-W50: Optimistic Concurrency Control with Wait 50
SRT: soft real-time transactions
EDF: earliest deadline first
DPM: Different Priority Method

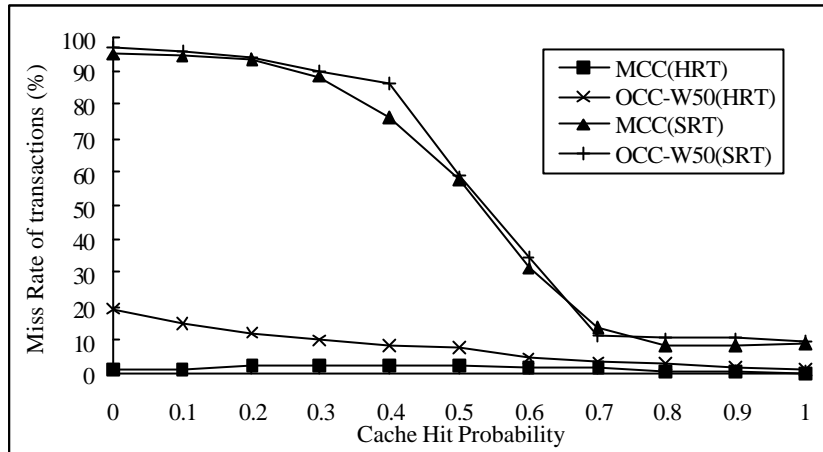


Figure 11: Impact of Cache Hit Probability on Miss Rate of SRT and HRT (EDF scheduling for CPU, FCFS scheduling for disk access, inter-arrival rate of SRT=5ms)

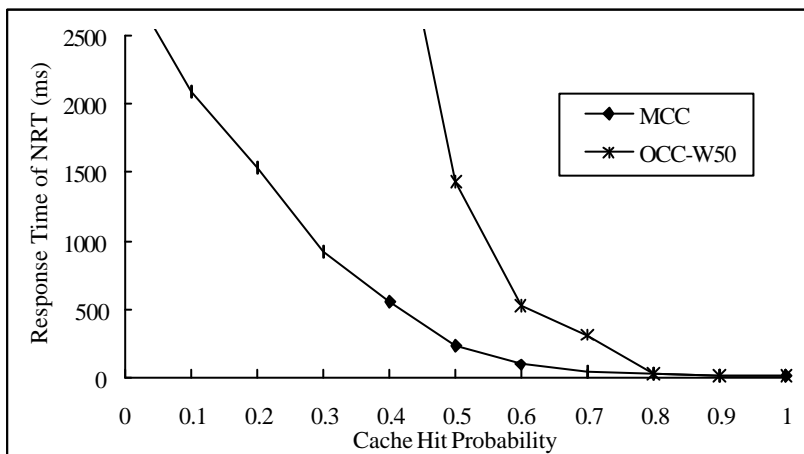


Figure 12: Impact of Cache Hit Probability on Response time of NRT (EDF scheduling for CPU, FCFS scheduling for disk access, inter-arrival rate of SRT=5ms)

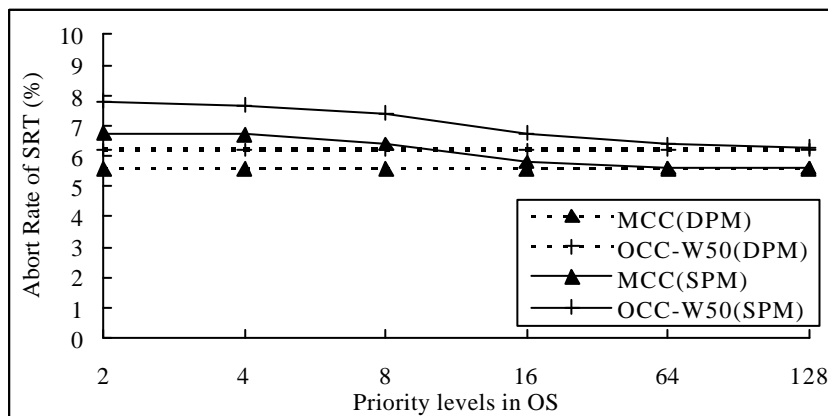


Figure 13: Impact of Priority levels on Abort Rate of SRT (main memory data item, inter-arrival rate of SRT=5ms)

MCC: Mixed concurrency control
HRT: hard real-time transactions
NRT: non-real-time transactions
SPM: Same Priority Method

OCC-W50: Optimistic Concurrency Control with Wait 50
SRT: soft real-time transactions
EDF: earliest deadline first
DPM: Different Priority Method

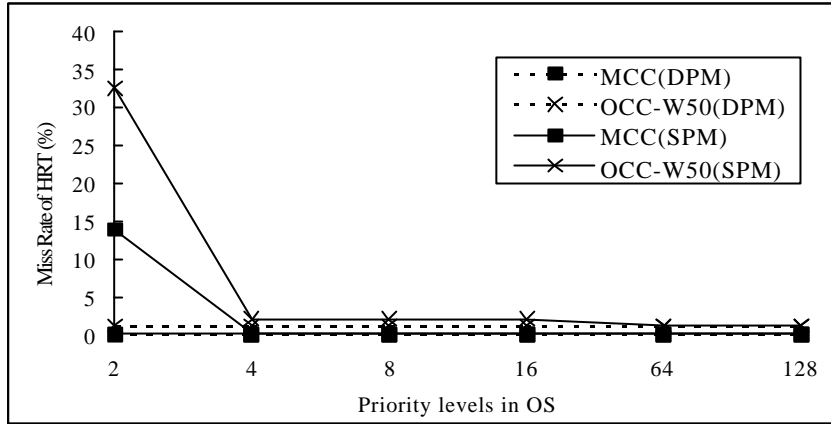


Figure 14: Impact of Priority levels on Miss Rate of HRT (main memory data item, inter-arrival rate of SRT=5ms)

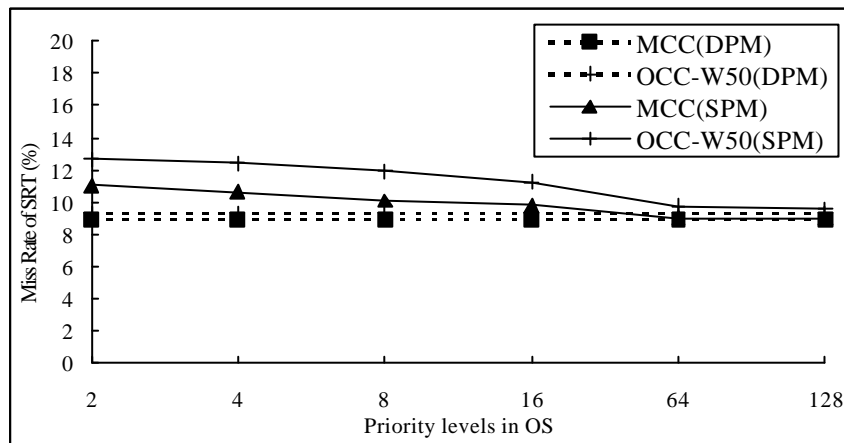


Figure 15: Impact of Priority levels on Miss Rate of SRT (main memory data item, inter-arrival rate of SRT=5ms)

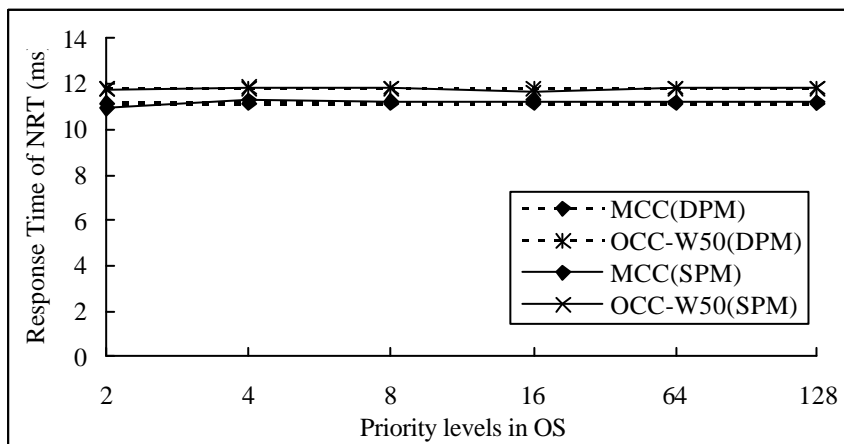


Figure 16: Impact of Priority levels on Abort Rate of SRT (main memory data item, inter-arrival rate of SRT=5ms)