

Evaluation of Concurrency Control Strategies for Mixed Soft Real-Time Database Systems

Kam-Yiu Lam¹, Tei-Wei Kuo², Ben Kao³, Tony S.H. Lee¹ and Reynold Cheng³

Department of Computer Science¹
City University of Hong Kong
83 Tat Chee Avenue, Hong Kong

Department of Computer Science and
Information Engineering²
National Taiwan University
Taipei, Taiwan, ROC

Department of Computer Science and Information Systems³
The University of Hong Kong
Pokfulam Road, Hong Kong

Email: cskylam@cityu.edu.hk, ktw@csie.ntu.edu, kao@csis.hku.hk, ckcheng@csis.hku.hk

Abstract

Previous research in real-time concurrency control mainly focuses on the schedulability guarantee of hard real-time transactions and the reducing of the miss rate of soft real-time transactions. Although many new database applications have significant response time requirements, not much work has been done in the joint scheduling of traditional non-real-time transactions and soft real-time transactions. In this paper, we study the concurrency control problems in mixed soft real-time database systems (MSRTDBS), in which both non-real-time and soft real-time transactions exist simultaneously. The objectives are to identify the cost and the performance tradeoff in the design of cost-effective and practical real-time concurrency control protocols, and to evaluate their performance under different real-time and non-real-time supports. In particular, we are interested in studying the impacts of different scheduling approaches for soft real-time transactions on the performance of non-real-time transactions. Instead of proposing yet another completely new real-time concurrency control protocol, our objective is to design an efficient integrated concurrency control method based on existing techniques. We propose several methods to integrate the well-known two phase locking (2PL) and optimistic concurrency control (OCC) with the aims to meet the deadline requirements of soft real-time transactions and, at the same time, to minimize the impact on the performance of non-real-time transactions. We have conducted a series of experiments based on a sanitized version of stock trading systems to evaluate the performance of both soft real-time and non-real-time transactions under different real-time supports in the system.

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

1 Introduction

Research in real-time database systems (RTDBS) has received a lot of interests in the past decade [AG92, B96, BLS87, DSK00, HR00, KWL99, L94, OS95, SKS96, YWLS92]. Transactions in a real-time database system are usually associated with deadlines. Any deadline violation of a *hard real-time transaction* may result in a catastrophe, while deadline violations of *soft real-time transactions* might only cause a degraded level of system performance without a total system failure [AG89, HCL92, ORP96, YWLS92]. In the past two decades, researchers have proposed various efficient real-time concurrency control techniques to either reduce the number of deadline violations for soft (and firm¹) real-time transactions or guarantee the deadlines of hard real-time transactions [AG92, BN96, HCL90, HS99, HSR92, KWL99, SLC91, SRSC91, UB93, UB98]. Some of the potential applications of RTDBS are programmed stock trading systems [AGK95, KR99], air-traffic management systems, air-navigation systems, and critical patient monitoring systems [KS96]. An important performance index of these systems is to handle events and requests in a “real-time” manner [R93, YWSL92].

Most of the previous studies in real-time concurrency control protocols often assume that the system may consist of only one single type of real-time transaction: soft or hard real-time transaction. Different assumptions have been made on the system and transaction models, e.g., the arrival pattern and data requirements of the transactions, such that different scheduling techniques can be engineered to satisfy the different requirements of real-time transactions. However, little work has been done in the development of an integrated approach that can handle a RTDBS satisfactorily where a mixed population of real-time and non-real-time transactions exists simultaneously. However, for many RTDB applications, such a mixture of workloads is very common. It is very important that real-time concurrency control protocols designed for a single type of real-time transactions be re-evaluated in a *mixed soft real-time database system* (MSRTDBS) – one that supports both non-real-time and soft real-time transactions.

A good example of a MSRTDBS is a programmed stock trading system [AGK95, KG93, KR99]. Although some of the transactions are soft real-time transactions, such as those for stock data update and analysis, there are also many non-real-time transactions, e.g., those for system management and query purposes. Stock updates are important in maintaining the up-to-the-second status of the stock market. In order to capture trading opportunities, stock updates are associated with deadlines. Missing the deadline of a stock update transaction is undesirable because it may result in out-dated stock data. However, if deadlines are missed only rarely, their occurrences are tolerable. Another example of MSRTDBS is online transaction processing (OLTP), especially those Internet-based OLTP systems and mobile computing systems [LKTL, U98]. Such systems usually impose different degrees of “*soft real-time*” constraints on the completion times of transactions. Although transactions may or may not have individual deadlines, the system may be associated with a *statistical timing constraint*. For example, a system may require that “95% of the transactions be completed within 5 seconds from their generation times.” Although a single

¹ Firm real-time transaction is a special type of soft real-time transactions. A firm real-time transaction will become totally useless after its deadline expires while the value of a soft real-time transaction decreases rapidly after its deadline.

deadline violation is tolerable, the satisfaction of certain *statistical timing requirements* is an important performance objective.

In this paper, we focus on the design of *cost-effective* and *practical* concurrency control protocols for MSRTDBS. We consider different performance goals in processing soft real-time and non-real-time transactions at the same time. The performance goal in processing soft real-time transactions is to minimize the number of deadlines violation (or to meet the system's statistical timing constraint), while the performance goal in processing non-real-time transactions is to maximize the system throughput (or to minimize the mean response time, for example). As astute readers may notice, conventional real-time concurrency control protocols designed for soft (or firm) real-time transactions often ignore the performance requirements of non-real-time transactions. For example, most of the real-time concurrency control protocols resolve access conflicts based on the idea of restarting of "less urgent" transactions. When they are applied to a MSRTDBS, it is likely that non-real-time transactions are restarted repeatedly by soft real-time transactions, due to data conflicts. This may cause the poor response times of non-real-time transactions. How to explore the balance and tradeoff between the performance of the two types of transactions is an important research topic.

An important issue in the design of concurrency control protocols for MSRTDBS is the "cost" and "benefit" in the supporting of the protocols and their effectiveness in reality, especially when the system cannot provide a perfect "real-time" environment. The impacts of different non-real-time features on different real-time concurrency protocols have been greatly ignored in most previous studies in this area. Most real-time concurrency control protocols require a priority-cognizant scheduler to schedule system resources and to resolve data conflicts based on transaction deadlines (priorities). Such a scheduler often requires the support of other system components, such as the operating system, the I/O system, and the database system. In practice, implementing all the required real-time features often means a much longer development period, a higher cost, and a *closed* system design – with a fixed set of hardware and software services. Significant cost may be introduced for system maintenance and system upgrading. Furthermore, since a real-time environment is required only for the soft real-time transactions, it becomes less cost-effective to transform a non-real-time environment into a real-time one, especially when the population of real-time transactions in the system is a minority. As a result, the most common (economic) approaches to serve MSRTDBS are mainly based on hardware upgrading (e.g., with a faster CPU and larger memory), instead of implementing a full-fledged real-time system. One of our main contributions of this paper is to study how the real-time concurrency control protocols behave in an environment that only provides limited real-time supports.

It is not the purpose in this paper to propose yet another completely new concurrency controller. Instead, our focus is on how existing real-time concurrency strategies, e.g., locking-oriented and OCC approaches, can be engineered for resolving the different types of data conflicts in a MSRTDBS. Although many efficient real-time concurrency control protocols have been proposed in the past two decades, very few of them have been adopted in practical real-time database systems. In our opinions, one of the major obstacles is the significant implementation

cost in integrating the protocols into existing systems and platforms, especially when the system has to support non-real-time transactions at the same time. Therefore, an important consideration in the design of the protocols is to minimize their requirements (or changes) on the original architecture of a traditional database system so that the cost in incorporating the real-time techniques in the database system is affordable, and at the same time to minimize the impacts on the non-real-time jobs. We adopt the well-known conventional concurrency control protocols, 2PL and OCC, as the base protocols to identify the cost and performance tradeoff, and their performance characteristics in MSRTDBS. The reason behind the selection of these two methods is that they are widely used in conventional and real-time database systems. Their performance characteristics and implementation overheads are well understood to database designers.

In the comparison of the concurrency control strategies, we consider the performance of both soft real-time and non-real-time transactions. The performance of the two well-known real-time concurrency control protocols, High Priority Two Phase Locking (HP-2PL) [AG92] and the optimistic concurrency control protocol with wait 50 (OCC-W50) [HCL90, HCL92], are examined in a MSRTDBS. Based on these methods and their performance characteristics, we propose several integrated methods for resolving different types of data conflicts. As a preview, the objectives of the paper are:

- (1) To study the performance tradeoff of different concurrency control strategies, i.e., lock-based and optimistic method, in a MSRTDBS.
- (2) To design efficient methods to minimize the impacts of providing real-time scheduling (to soft real-time transactions) on non-real-time transactions based on existing real-time concurrency control approaches.
- (3) To study the impacts of non-real-time environments on the concurrency control protocols, especially when the underlying operating system supports only a limited number of priority levels.

To investigate the performance characteristics of the proposed methods and the performance tradeoffs of various techniques, we have designed and implemented a stock trading database system model. Extensive simulation experiments have been conducted to study various important real-time and non-real-time issues, such as CPU scheduling and disk I/O scheduling on the performance of the protocols. The experiment results provide important insights in the design of a MSRTDBS because most conventional database systems with soft real-time requirements only support very limited real-time features.

The rest of the paper is organized as follows. Section 2 reviews the concurrency control protocols for real-time database systems. Section 3 defines the model for a real-time database system. Section 4 proposes an integrated concurrency control protocol for mixed real-time database systems, and its correctness is shown. Section 5 discusses the implementation issues of the integrated protocol. Section 6 is the performance studies of the protocols under different environments that provide different levels of real-time supports. Section 7 is the Conclusion and Future Works.

2 Concurrency Control Protocols and Priority Scheduling

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

The most serious problem of conventional non-real-time concurrency control protocols when applied to a RTDBS is priority inversion [SRL87, SRSC91]. It is a situation in which a higher-priority transaction is blocked by a lower-priority transaction. When this happens, the higher-priority transaction will have a high probability of missing its deadline. In order to alleviate this situation, various priority-cognitive conflict resolution strategies have been suggested for different types of real-time transactions, e.g., soft, firm, and hard real-time transactions. The strategies are mainly based on transaction restart and/or data reservation. One of the most important conflict resolution strategies is the High Priority Two-Phase Locking (HP-2PL) protocol [AG92], which is based on 2PL. In HP-2PL, if the priority of a lock-requesting transaction is higher than the lock-holding transaction, the lock-holding transaction is restarted. Otherwise, the lock-requesting transaction is blocked. Other lock-based methods are priority inheritance and priority ceiling protocol (PCP). Although PCP guarantees all hard deadlines, it is not suitable for soft and firm real-time transactions as PCP assumes that the system consists of periodic transactions only, and the set of data items to be accessed by a transaction has to be known a priori. These assumptions are usually not true to soft and firm real-time transactions, due to their dynamic and unpredictable behaviors [KS96, R93].

In addition to the lock-based protocols, many real-time concurrency control protocols are based on the optimistic concurrency control (OCC) method. Various forward validation schemes, such as OCC-wait, wait-50 and sacrifice, were proposed for OCC in RTDBS [HCL90, HCL92]. In the wait-50 scheme, if the number of conflicting transactions with a priority higher than the validating one is greater than or equal to half of the total number of conflicting transactions, the validating transaction is blocked until the number is smaller than 50%. Otherwise, all conflicting transactions are restarted even though some of their priorities are higher than the priority of the validating transaction. In the sacrifice method, if there is a conflicting transaction whose priority is higher than the validating transaction, the validating transaction is restarted. Otherwise, all conflicting transactions are restarted. Haritsa, et al. [HCL92] had shown the superiority of the optimistic concurrency control protocols over the lock-based protocols, due to more fruitful restarts.

2.2 Concurrency Control for Mixed Real-time Database Systems

The study of concurrency control protocols for MSRTDBS has received growing interests in recent years [DMKV96, KS96, LKTL00, TSH96]. One of the pioneer studies is [TSH96] in which a two-level concurrency control architecture is proposed for MSRTDBS. The upper level is a master concurrency controller (MCC) which is used to detect data conflicts between transactions belonging to different classes. The MCC also forwards data requests to individual concurrency controller for detecting data conflicts amongst transactions in the same class. To ensure the final serializability of the mixed transactions, a serialization checking method based on the time-stamps of transactions is proposed to resolve the problem [TSH96]. The two-level controller architecture provides an easy way to extend conventional non-real-time scheduler for concurrency control of MSRTDBS. In their simulation

experiments, the base protocols for non-real-time and soft real-time transactions are 2PL and OCC-Wait, respectively. The MCC also adopts the OCC-Wait policy for resolving the inter-class conflicts.

Different from the MCC method, in this paper we propose another alternative to resolve the problem. We propose to use an integrated approach, e.g., using a single controller for detecting and resolving data conflicts between transactions belonging to the same type and also between different types of transactions. A common lock table is employed for conflict detection and resolution. The main advantage of the integrated approach is that it will be more flexible in the design of concurrency control schemes and in optimizing the system performance, although the additional overhead in the implementation of the controller may be higher due to the need of re-implementing the whole controller. In order to minimize the additional cost in implementation, the strategies adopted for concurrency control in the proposed integrated methods are basically based on those commonly used methods, e.g., lock-based and optimistic methods. Three important contributions of our study as compared with [TSH96] are that (1) we are designing methods to support real-time scheduling for soft real-time transactions and at the same time would like to minimize the impact on non-real-time transactions; (2) different variations and tradeoff in the design of the conflict resolution strategies have also been examined; (3) the performance characteristics have been investigated and we also have compared the performance of our proposed integrated methods with the MCC methods under both real-time and non-real-time environments.

In addition to [TSH96], other important previous works in the area include [LKTL00] and [KS96]. In [LKTL00], the reduced ceiling protocol (RCP) is proposed for a RTDBS that consists of both hard and soft real-time transactions. In RCP, the deadlines of hard real-time transactions are guaranteed by using the PCP principles. Soft real-time transactions are processed by a modified optimistic protocol to reduce the number of restarts of soft real-time transactions due to data conflicts. In [KS96], a framework for RTDBS with mixed transactions is proposed such that transactions with different characteristics and criticality levels are classified into different classes. Different scheduling principles are used to schedule transactions in different classes. In the proposed framework, it is assumed that hard real-time transactions (class-I transactions as defined in their work) will not have any data conflicts with other class-I transactions and soft real-time transactions.

2.3 Priority Assignment

As shown in the previous studies, priority scheduler schemes can greatly affect the performance of a concurrency control protocol [AGM95, LLKL00]. The majority of the previous studies on concurrency control for soft (and firm) RTDBS assume that the earliest deadline first (EDF) [LL76] priority assignment scheme is used for CPU scheduling and data conflict resolution. Under EDF, the transaction with the closest deadline is assigned to the highest priority level. Although EDF scheduling has been shown to be suitable for RTDBS, it has been found that the performance of the system will be very poor when the system workload is heavy as the highest priority is assigned to a transaction that is going to miss its deadline [HCL91]. An adaptive EDF scheduling algorithm is proposed [HCL91] to solve the problem. However, it lacks any studies to investigate the relationship between EDF

and different concurrency control strategies and what will be their performance when EDF cannot be fully supported in the system.

Although EDF has been shown to be effective in meeting transaction deadlines, it may not be easily implemented in a traditional or even a real-time operating system. Traditional operating systems usually work with a fixed number of discrete priority levels, instead of a continuous range of deadlines. For example, QNX, which is a real-time operating system, supports only 128 levels of priorities [G95]. The priority-mapping problem in a general-purpose operating system has been discussed in [AGK94]. The paper describes how to convert or map a transaction's timing constraint (such as a deadline, or the amount of slack that a transaction has) into an integral priority level. Their idea is to pick a mapping function wisely such that a static priority scheduler would mimic the behavior of a certain real-time scheduling algorithm.

In [AGK94], different mapping functions for various target real-time schedulers are studied. Here we briefly mention one of them, namely, "EDF Relative (EDREL)". For EDREL, a transaction's priority is determined at arrival time by a simple mapping 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 [AGK94], it is suggested that ts be set to:

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

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

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

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

As shown in [AGK94], the performance of different priority mapping schemes highly depends on the number of priority levels provided by the operating systems. If the number is very small, transactions with different deadlines may be mapped to the same priority. Thus, their priorities may not be a very good indicator for their urgency. In Section 6, we will study the performance of different concurrency control protocols for MSRTDBS with EDREL when there are only a limited number of priority levels in the system. Different from the work in [HCL91], we are not to investigate the performance of EDF in a MSRTDBS. Instead, we would like to study how EDF and the priority mapping problem in EDF affects the performance of different strategies in concurrency control.

3 Database System Model

The purpose of this section is to define a MSRTDBS model which is based on previous studies in the area [AG92, TSH96, UB93]. It is assumed that the database system consists of three major components [BHG87]: the transaction manager (TM), the scheduler, and the data manager (DM). TM maintains a transaction table to record the execution status of transactions in the system and to preprocess transactions if necessary. TM forwards operations issued by a transaction to the scheduler. The scheduler is responsible for concurrency control. When the

scheduler receives an operation, it determines whether the operation should be processed, blocked, or rejected. If an operation is rejected, the corresponding transaction will be restarted. The scheduler maintains an access-status table to detect any possible data conflicts. For example, if 2PL is adopted for concurrency control, the scheduler may maintain a lock table to record the locking status of any data items accessed by all executing transactions (and the collection of the blocked transactions due to lock conflicts). The same table can also be used for the optimistic methods [HCL91].

The DM is divided into two parts: the recovery manager (RM) and the cache manager (CM). All data access requests issued by an operation are handled by the DM, which retrieves the required data items from the disk or the cache buffer. After obtaining the required data items, the processing of the operation may continue. We do not use a main-memory resident database in our model because it is not common in commercial database systems. Furthermore, a main-memory database may require a proper mechanism to solve the database recovery problem. We, therefore, adopt a disk-based database model throughout this paper. The CM is for efficient access and management of the cached data.

There are two classes of transactions in the system: soft real-time and non-real-time transactions, which all have similar structures. Each transaction consists of a sequence of read and write operations, and ends with a commit or an abort operation. Each soft real-time transaction is associated with a deadline and a priority which is derived from the deadline based on the adopted priority mapping function. Although there are no deadlines for non-real-time transactions, it is desirable to have a small mean response time for the non-real-time transactions. Note that the classification of a transaction as a soft real-time transaction or as a non-real-time transaction is based on the application requirements such as the consequences of a delayed response time on their usefulness.

4 Concurrency Control Protocols for MSRTDBS

The purpose of this section is to explore the two well-known approaches, namely the lock-based and the optimistic protocols, for concurrency control in MSRTDBS. The rationales behind choosing these two approaches are as follows:

- (1) Many existing real-time concurrency control protocols are based on 2PL and OCC, and they have shown to be efficient in the concurrency control of real-time data accesses; and
- (2) 2PL and OCC belong to two distinct major classes of concurrency control protocols: conservative and optimistic protocols. They have been widely used in conventional database systems. Their implementation overheads and performance characteristics are well understood.

Amongst the existing real-time protocols, we choose HP-2PL² and OCC-W50³ as the target lock-based and OCC protocols, respectively. The selection is also for their simplicity in implementation and good performance.

² Although various optimizations have been suggested to further improve the performance of the HP-2PL, we choose HP-2PL as the basis for discussion. Note that other variants of HP-2PL such as cautious waiting and

Based on their performance characteristics, we shall then propose integrated concurrency control protocols. In a MSRTDBS, there are three types of data conflicts: (1) those among soft real-time transactions; (2) those among non-real-time transactions; and (3) those between soft-real-time and non-real-time transactions. The first two types are intra-class access conflicts, and the last type is inter-class conflict [TSH96]. We may use either the same concurrency control scheme or different concurrency control schemes to resolve the intra-class conflicts and the inter-class conflicts.

In the following sub-sections, we will first discuss how HP-2PL and OCC-W50 can be used for integrated concurrency control, and then we will discuss their tradeoffs and performance characteristics. Later, we will discuss how these strategies can be integrated for MSRTDBS.

4.1 Basic Protocols

4.1.1 Mixed High Priority Two Phase Locking (MHP-2PL)

In HP-2PL, the basic principle in resolving data conflicts between two transactions is to restart the lower-priority transaction. When HP-2PL is applied to a MSRTDBS, soft real-time transactions are always assigned higher priorities than non-real-time transactions. This approach provides real-time concurrency control for soft real-time transactions and is deadlock-free if the deadlines of the soft real-time transactions are unique. The main concern is the significant cost in resolving the lock conflicts, e.g., transaction restarts. Thus, when there is a conflict between a non-real-time transaction and a soft real-time transaction, the non-real-time transaction is likely to be restarted leading to a poor performance of non-real-time transactions. Moreover, there is the problem of assigning priorities to non-real-time transactions, which are not associated with any deadlines. (Most of the proposed priority assignment methods are based on transaction deadlines.) One simple solution is to assign the same lowest priority to all non-real-time transactions. In this way, the resolution of any data access conflict between soft-real-time and non-real-time transactions follows the original principle of HP-2PL. We call this variant of HP-2PL as *Mixed HP-2PL (MHP-2PL)*. The following table summaries the conflict resolution methods for MHP-2PL:

		Lock-requester	
		Soft real-time	Non-real-time
Lock-holder	Soft real-time	HP-2PL	Block requester
	Non-real-time	Restart holder	2PL

Table 1: Conflict resolution strategies in MHP-2PL

Since 2PL is used for non-real-time transactions, a deadlock resolution algorithm is needed for the executions of non-real-time transactions. It is obvious to see that all soft-real-time transactions in MHP-2PL schedules are deadlock-free.

conditional priority can also be applied.

³ The performance of various optimistic protocols for RTDBS has been investigated in [HCL91]. It has been shown that, in general, OCC-W50 gives the best performance.

4.1.2 Mixed OCC Wait-50 Protocol (MOCC-W50)

Past research suggests that optimistic approaches are better than lock-based protocols in concurrency control for soft (and firm) real-time transactions when it is combined with the forward validation scheme [HCL92, L94]. To apply an optimistic protocol (such as OCC-W50) to a MSRTDBS, the simplest way is to assign the same priority to non-real-time transactions and assign higher priorities to soft real-time transactions. (Note that different variants of priority cognitive optimistic concurrency control methods have been proposed. Here we choose OCC-W50 as an example for the discussion of how real-time OCC can be modified for MSRTDBS.) When a transaction enters its validation phase, the system compares the write data set of the validating transaction with the read data sets of other executing transactions. The resolution method then follows the principle of OCC-W50. Non-real-time transactions are always restarted if they are involved in any data conflict with soft real-time transactions. Non-real-time transactions are scheduled by a conventional OCC protocol. We call this variant of OCC-W50 as *Mixed OCC wait 50 (MOCC-W50)*. Table 2 summarizes the conflict resolution methods for MOCC-W50.

		Validating transaction	
		Soft real-time	Non-real-time
Conflicting executing transaction	Soft real-time	OCC-W50	Restart validating transaction
	Non-real-time	Restart executing transaction	OCC method

Table 2: Conflict resolution strategies in MOCC-W50

Note that in order to eliminate the restart of a soft real-time transaction caused by a conflict with non-real-time transactions, in counting the number of lower-priority transactions in the OCC-W50, the non-real-time transactions will not be counted.

4.1.3 Performance Comparison: Lock-based Vs. Optimistic protocols

Before we introduce the integrated methods, we would like to have a qualitative comparison of the performance of the lock-based protocols and optimistic protocols for MSRTDBS. The comparison can be used as guidelines for the design of the integrated protocols for MSRTDBS. Note that in a MSRTDBS, we are interested in both the performance of soft real-time transactions (in terms of the probability of meeting deadlines) and the performance of non-real-time transactions (in terms of the mean response time). We choose HP-2PL and OCC-W50 as the representatives of the lock-based and optimistic protocols, respectively. Here, let us first identify five characteristics of a real-time concurrency control protocol, which directly affect the performance of a MSRTDBS:

Number of Restarts: Both HP-2PL and OCC-W50 use transaction restarts as their main conflict resolution strategy. Restarting a transaction means giving up the resources already invested in the transaction and is thus detrimental to the system's performance. Moreover, a restarted transaction has a higher probability of missing its deadline since it has lost a substantial amount of its slack. Hence, restarts should be avoided. Compared with the

part-blocking and part-restarting conflict resolution strategies of HP-2PL, in general, OCC-W50 generates more restarts than HP-2PL does as it delays the detection of data conflict when a transaction wants to commit, e.g., in the validation phase. Since non-real-time transactions are lower priority transactions, the number of restarts of non-real-time transactions is larger under the OCC-W50 than HP-2PL.

Restart Overhead: Not only does OCC-W50 cause more restarts, the restart overhead is also higher. It is because in OCC-W50 data conflicts are generally detected later than in HP-2PL as it resolves data conflicts only when one of the conflicting transactions enters the validation phase. The restart overhead increases the response time of the transactions, especially the non-real-time transactions, because they are given lower priorities in using the system resources. In HP-2PL, blocking may be used to resolve data conflicts between transactions. Blocking can reduce the contention in using the system resources and thus can reduce the waiting time of a transaction for system resource, especially the non-real-time transactions, because they are more sensitive to the system workload.

Useless Restarts: After a real-time transaction T_1 restarts another transaction T_2 , T_1 may later miss its deadline. We call the restart of T_2 *useless*, since the action does not help T_1 in meeting its deadline. In HP-2PL, T_1 may miss its deadline (and thus causes a useless restart) because of the delay experienced in further resource contention (CPU and/or I/O), or because it is later restarted by or waits for another higher-priority transaction during conflict resolution. The impact of useless restarts on non-real-time transactions is an increase in response time. In OCC-W50, a transaction that restarts other transactions must be committing, and thus will seldom be restarted by or waits for other transaction executions. In this aspect, OCC-W50 is better than HP-2PL in avoiding useless restarts. Note that OCC-W50 does not prevent useless restarts completely. This is because after the validation phase, new data values still need to be written to the physical database before the transaction deadline. Storage access delay may still cause the transaction to become tardy.

Useless Waits: In HP-2PL, when a transaction T_1 is blocked by another transaction T_2 due to a lock conflict, T_1 is delayed. If T_2 is a soft real-time transaction, and it later becomes tardy, then the wait caused to T_1 is *wasted*. We call the blocking caused by T_2 *useless wait*. Similar to the arguments for *useless restarts*, in HP-2PL, T_2 may miss its deadline (and thus cause a useless wait) because of further resource contention (CPU and I/O) and/or conflict resolution. The impact of useless waits on transactions is similar to the case of useless restarts. OCC-W50 is better than HP-2PL in avoiding useless waits.

Harmful Waits: In HP-2PL, when a transaction T_1 is blocked by another transaction T_2 due to a lock conflict, T_1 is delayed. If T_1 is restarted while it is waiting, due to a data conflict with another higher-priority transaction, then the wait caused to T_1 is harmful. If T_1 does not need to wait, then it may be able to complete earlier without being restarted by other transactions. Harmful waits occur more likely to non-real-time transactions because they are more likely being blocked due to their low priorities.

We summarize the advantages and disadvantages of HP-2PL and OCC-W50 in Table 3. In the table, we use a ✓ (✗) to indicate good (bad) performance with respect to a certain system characteristic.

	HP-2PL	OCC-W50
Number of restarts	✓	✗
Restart overhead	✓	✗
Useless restarts	✗	✓
Useless waits	✗	✓
Harmful waits	✗	✓

Table 3: Relative advantages and disadvantages of HP-2PL and OCC-W50.

Note that we have only identified the factors that may affect the system performance. The extent of the performance impacts (on which each of the factor exerts) depends on various system parameters, such as the system workload, the data access time, and the degree of data contention among transactions. Furthermore, in the discussion of the impact on system performance, we have ignored the effect of disk scheduling, data caching, and priority scheduling of the CPU and disk. The significance of such factors will be very different when their impact considerations are combined with those for disk scheduling, data caching, and priority scheduling. For example, disk accesses are slow and unpredictable. The response time of a disk request depends on the position of the disk head (and thus on the previous disk access) and how the disk schedules requests. Typical disk access time is in the order of milliseconds with a relatively large variance. Moreover, disk requests are served one at a time, and preemption is not possible. Therefore, it is possible that a disk request (and the issuing transaction) is blocked by a few others due to *disk contention* (even if there is no data conflict among the transactions). Disk access delay and, more importantly, its variance, are amplified by the *chain of requests*. These factors are detrimental to satisfying the transaction deadlines [CSKT91]. On the other hand, the probability of cache hit can significantly affect the impact of the disk scheduling on the performance of the protocols.

If a deadline-cognizant technique is adopted to schedule disk requests [AG89], requests from transactions with earlier deadlines may have a higher chance of being served first. The disk thus acts (indirectly) as an agent that controls the progress of the transactions, and the waiting time of non-real-time transactions will be much longer than that of the soft real-time transactions. For OCC-W50, we might observe that higher-priority transactions are advancing at a much faster pace, while lower-priority ones are delayed (or blocked) at the disk access level. This can result in a very poor performance to the non-real-time transactions, and they will have a much higher probability to be restarted while they are waiting for the disk (harmful wait).

4.2 Integrated Methods (IM)

4.2.1 The Methods

As discussed in the pervious section and reported from previous works, optimistic protocols are suitable for soft real-time transactions [HCL92] while the lock-based protocols are more suitable for non-real-time transactions [BGH87]. Therefore, one simple way to design concurrency control strategies for MSRTDBS is to use an integrated

method (IM), where OCC-W50 is used to resolve data conflicts amongst soft real-time transactions, and 2PL is used to resolve data conflicts amongst non-real-time transactions. In order to resolve the inter-class conflicts, soft real-time transactions also require a lock being set before they access a data item even though they follow the optimistic principles. The implementation of the optimistic method uses locking instead of time-stamps. The details of conflict detection are summarized in the following table.

		Requester/validating transaction	
		Soft real-time	Non-real-time
Holder/executing transaction	Soft real-time	OCC-W50	Block requester
	Non-real-time	Restart holder	2PL

Table 4: Conflict resolution strategies in IM

When a non-real-time transaction wants to lock a data item, which is already accessed by a soft-real-time transaction, the non-real-time transaction will be blocked. It is because even if the non-real-time transaction is granted the lock and allowed to access the data item, it is likely to be restarted by the soft real-time transaction when the soft real-time transaction enters its validation phase. Of course the disadvantage of the blocking is a higher probability of *useless wait* and *harmful wait*. If a non-real-time transaction locks a data item before a soft real-time transaction accesses the data item, the data conflict will be resolved when the soft real-time transaction enters its validation phase, or when the non-real-time transaction wants to commit, depending on which occurs first. For the first case, the non-real-time transaction will be restarted if the soft real-time transaction can pass the validation test. However, if the soft real-time transaction cannot pass the validation phase due to data conflicts with another higher priority soft real-time transaction, the non-real-time transaction can continue its execution. For the second case, the non-real-time transaction will be restarted after it has detected that it is in conflict with a soft real-time transaction.

For resolving intra-class conflicts of soft real-time transactions, we use the OCC-W50 principles. If the validating soft real-time transaction can pass the validation test, it will restart all conflicting soft real-time transactions and non-real-time transactions. Note that in order to eliminate the impact of non-real-time transactions on the validation result of soft real-time transactions, all non-real-time transactions will not be counted in the validation test.

The advantage of IM is that it may be able to reduce the *number of restarts* and the *restart overhead*. It is because blocking is used to resolve the data conflicts between non-real-time transactions and also for the case when a non-real-time transaction wants to access a data item, which has been accessed by a soft real-time transaction. The tradeoff is between *harmful wait* and *useless wait* with *restart overhead* and *number of restarts*. This is especially important when the system workload is heavy. A problem of IM is that deadlocks are possible among non-real-time transactions, although the probability should be smaller than the case when 2PL is adopted for both transaction types. A conventional deadlock resolution method may be used to resolve the deadlock problem.

In IM, when a soft real-time transaction accesses a data item, which is locked by an executing non-real-time transaction, the soft real-time transaction will be allowed to access the data item. The execution of the non-real-time transaction will not be affected at this stage. The resolution of the conflict is delayed until one of the conflicting transactions wants to commit. The incurred problem is that the delay of the resolution of the data conflicts will make the amount of resources wasted on the restarted transactions higher (higher *restart overhead*). One way to resolve the problem is to resolve the data conflict immediately when the soft real-time transaction wants to access the data item, e.g., by restarting the non-real-time transaction immediately in case of any data conflict. We call this variant of IM as *IM with early detection* (IM-ED).

Resolving data conflict using IM-ED has the possibility of *useless restart* but can reduce the *restart overhead*. An example of useless restart is as follows: Suppose a soft real-time transaction S_j has a data conflict with a non-real-time transaction. Due to the early restart principle of IM-ED, the non-real-time transaction is restarted before S_j enters its validation phase. If S_j is aborted subsequently, then the restart of the non-real-time transaction is useless.

With the integrated concurrency control of soft real-time and non-real-time transactions, non-real-time transactions may inevitably have a higher restart probability when the system is heavily loaded. To reduce the restart cost of non-real-time transactions, we may use a *deferred update* mechanism for the non-real-time transactions. We call this variant *IM with deferred updates* (IM-DU). Under IM-DU, update operations issued by a non-real-time transaction are first applied only to the transaction's private workspace. The updated values are only copied to the database when the transaction commits. With the deferred update mechanism, their execution of a non-real-time transaction is divided into three phases: execution phase, validation phase, and commit phase. Before a non-real-time transaction commits, it has to pass a validation phase to check whether it is in conflict with any soft real-time transactions. Note that the concurrency control among the non-real-time transactions is still following the 2PL principles. The actions taken by conflicting read and write operations are summarized in table 5:

			Requester/Validating Transaction			
			Soft-RT transaction		Non-RT transaction	
			Read	Write	Read	Write
Holder/ Executing Transaction	Soft-RT Transaction	Read	✓	OCC-W50	✓	Block requester
		Write	✓	✓	Block requester	✓
	Non-RT Transaction	Read	✓	Restart holder	✓	2PL
		Write	✓	✓	2PL	2PL

Table 5: Conflict resolution strategies in IM-DU

The inter-class conflicts are resolved when a soft real-time transaction enters its validation phase, or when a non-real-time transaction wants to commit and permanently update the database. The inter-class conflict resolution strategy may, in general, follow those described in IM. Therefore, IM-DU adopts both *optimistic* and *lock-based*

approaches for resolving data conflicts of non-real-time transactions. The lock-based approach is used for resolving intra-class data conflicts among the non-real-time transactions while an optimistic approach is used for resolving data conflicts with soft real-time transactions.

With local data copies for each transaction under the IM-DU, the inter-class resolution mechanism can be optimized as follows to reduce the number of restarts: If a read/write conflict happens between a validating soft real-time transaction and any executing non-real-time transaction, the executing non-real-time transaction is restarted. If a write/write conflict happens between a validating soft real-time transaction and any executing non-real-time transaction, the non-real-time transaction is allowed to continue its execution due to the Thomas Write Rule [BHG87]. If a read/write conflict happens between a committing non-real-time transaction and any executing soft real-time transactions, the non-real-time transaction is aborted. However, if only write/write conflicts occur, the non-real-time transactions can commit due to the Thomas Write Rule.

4.2.2 The Correctness of the Integrated Methods (IM)

Theorem 1: All soft real-time transactions in IM schedules are deadlock-free.

Proof. The correctness of this theorem follows directly from the deadlock-free property of OCC-W50 [BHG87, HCL92] and the fact that no deadlock involves both soft-real-time and non-real-time transactions. Note that the early detection mechanism and the deferred update mechanism will not introduce any new chance for deadlocks. Q.E.D.

Note that since 2PL is used to schedule non-real-time transactions, a deadlock resolution algorithm is needed for the execution of non-real-time transactions.

Theorem 2: All IM schedules are serializable.

Proof. The correctness of this theorem follows directly from the serializability properties of OCC-W50 [BHG87, HCL92] and 2PL. Note that the early detection mechanism and the deferred update mechanism will not introduce any possibility of serializability violation. Q.E.D.

5 Conflict Detection in the Integrated Methods

The purpose of this section is to discuss how the data conflict detection mechanisms of IM can be implemented and what will be the implementation cost and run-time overhead. Since the detection of possible data conflicts in 2PL, HP-2PL, and OCC-W50 is very straightforward and have been discussed in many previous works, we will not discuss their detection mechanisms here. An important consideration in the implementation of IM is that its run-time overhead and implementation cost should be comparable to that of other conventional techniques, i.e., two phase locking. Therefore, we choose to use a lock table to detect data conflicts in the implementation of IM.

In the lock table, there are three types of locks. They are soft pre-locks (SP-locks), validation locks (V-locks), and conventional locks (C-locks). Soft pre-lock and validation locks are for soft real-time transactions. Since

a soft real-time transaction may read or write a data item, soft pre-locks are classified as soft pre-read-locks (SPR-locks) and soft pre-write-locks (SPW-locks). V-locks are converted from pre-locks when a soft real-time transaction is in the validation phase. Conventional locks are for non-real-time transactions. Since there are also two types of operations, read and write for non-real-time transactions, conventional locks can also be classified as conventional read locks (CR-locks) and conventional write locks (CW-locks). The compatibility of the locks is shown in table 6.

Lock Requester	Lock Holder				
	SPR-lock	SPW-lock	CR-lock	CW-lock	V-lock
SPR-lock	✓	✓	✓	✓	✗
SPW-lock	✓	✓	✓	✓	✗
CR-lock	✓	✗	✓	✗	✗
CW-lock	✗	✗	✗	✗	✗
V-lock	✓	✓	✓	✓	✗

Table 6: Lock Compatibility Table

✓ : the lock requester is allowed to set the lock

✗ : the lock request is denied

During the read phase, a soft real-time transaction sets a SP-lock on a data item before it accesses the data item. If it wants to read the data item, it sets a SPR-lock. If it wants to write the data item, it sets a SPW-lock. All SP-locks are compatible with each other. The lock request is denied if a V-lock is already set on the same data item by another transaction. If there is a V-lock, then it means that another soft real-time transaction, which has accessed the data item, is now in the validation phase or in the write phase. As a result, the lock-requesting soft real-time transaction is not allowed to access the data item because the validation phase and the write phase must be performed in a critical section [BHG87]. Although it may create the priority inversion problem, it will not seriously affect the performance of the higher-priority transactions because the procedure can be performed in a very short time interval.

When a soft real-time transaction finishes its read phase, it enters the validation phase in which it converts its SPR-locks and SPW-locks into V-locks. The conversion of a SP-lock to a V-lock will not be allowed if there is a V-lock on the same data item by another transaction. In this case, the soft real-time transaction has to wait until the validating transaction completes its validation and releases the V-lock. Once the soft real-time transaction has converted all of its Pre-locks into V-locks, the system will check data conflicts for the validating transaction against other soft-real-time and non-real-time transactions. The lock table will be examined for SPR-locks and CR-locks being held by other transactions. The resolution of conflicts then follows the principles of OCC-W50. If the number of conflicting transactions, whose priorities are higher than the priority of the validating transaction, is greater than 50% of the total number of conflicting transactions excluding non-real-time transactions, the validating transaction will be blocked. Otherwise, all the conflicting transactions will be restarted.

A non-real-time transaction has to set a C-lock before it accesses the corresponding data item. The lock request will be denied if there is a V-lock, a C-lock, or a SP-lock in conflicting mode on the same data item. In that case, the non-real-time transaction will be blocked.

It can be seen from the above discussion that the implementation and run-time overheads should be similar to that of the 2PL and OCC methods. The main differences comparing with the 2PL are the introduction of different modes of locks and the conversions of locks to V-locks in the validation phase.

The above lock principles can be easily revised for IM-ED or IM-DU. To support IM-ED, conflicts will be checked against C-locks when the scheduler receives a SP-lock request. The conflicting holder of a C-lock will be restarted if the lock mode is not compatible. The revision for IM-DU is also trivial. It can easily be done by setting all of the C-locks and SP-locks to be compatible with each other, and data conflicts will be detected when the non-real-time transactions or the soft real-time transactions enter the validation phase in which they convert all their C-locks or SP-locks to validation locks. At the same time, it will detect lock conflicts.

6 Performance Studies

6.1 Performance Model

We have developed a model for a stock trading system, as shown in Figure 1, to study the performance of the proposed protocols in a MSRTDBS and to identify their performance characteristics. The reason for choosing a stock trading system is that it is a typical application of RTDBS, and it possesses many unique characteristics of a RTDBS, e.g., a large number of real-time and non-real-time transactions, and the existence of temporal data items, e.g., derived and base items [KG93, R93].

In the model, transactions are classified into two types: soft real-time (SRT) and non-real-time (NRT) transactions. Two generators are defined in the model for their generations. The first one is for generating soft real-time transactions, which are mainly stock update transactions. The second one is for generating non-real-time transactions, which are queries for stock information, and transactions for trading and system management. The inter-arrival times of both soft real-time and non-real-time transactions follow exponential distributions.

Data items in the system are classified as base items and derived items [KLACL99]. Base items record the “real-time” status of the stocks. Their values are highly dynamic. Usually, each base item corresponds to the information about one stock. Some examples are the bid and ask status of a stock and the last traded prices of the stock. The values of derived items are derived from one or more base items or other derived items. Some examples are the total traded volumes of the stock up to the current time and the highest and lowest traded prices. In addition to the information of each individual stock, some derived data items are used to record information of a group of stocks and the whole market. Some examples are the index for individual stock type and the total transaction volume. We assume that the database resides on disks. Note that both base items and derived items are temporal data items. Timely completions of updates on the data items are required to maintain the external consistency and relative

consistency [R93] of the database. Although some previous studies in RTDBS have suggested the use of absolute validity interval (AVI) to define the validity of a temporal data item, we do not associate such a timing constraint on the temporal data items in our model. The reason is that the temporal data items of a stock system are discrete data items. The validity of a data item depends on the arrival time of the next update which is totally sporadic [KLACL99]. Furthermore, the difference in value between successive updates can be very large in a stock system. Therefore, it is very difficult to define a suitable AVI to the data items.

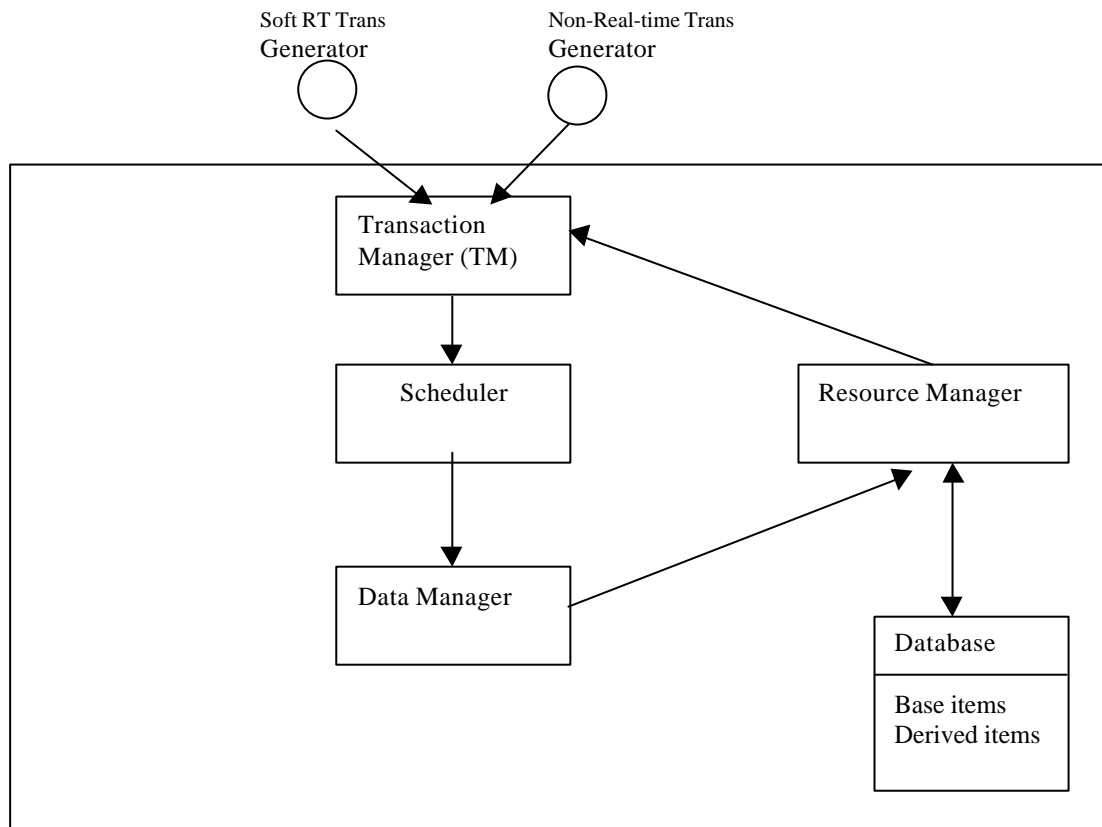


Figure 1: System Architecture of a MSRTDBS model

We assume that most of the soft real-time transactions are stock update transactions whose arrival rate is very high. Each stock update transaction is associated with a priority and a deadline. We also assume that the usefulness of a stock update transaction degrades rapidly after its deadline. Whenever a stock update transaction arrives, it will update a base item. After completing the update of the data items, the stock update transaction may perform re-computations for related derived items, such as the re-computation of the highest traded price if there is a change in the traded price. The number of re-computation operations in an update transaction is called the *fan-out* [KLACL99].

Non-real-time transactions may read base items and read/write derived items, e.g., the individual stock information, the information of some stock types, and the market information. It is assumed that an operation of a non-real-time transaction has a probability of $prob_base_item$ of accessing a base item and a probability of $(1 - prob_base_item)$ of accessing a derived item. Their arrivals are purely sporadic, and there are no timing constraints on their completion times.

We consider a high performance multi-processors system. In the baseline setting, we employ EDF as the CPU scheduling policy. (Other scheduling algorithms, e.g., FCFS and EDREL, are also experimented in the performance studies.) Each transaction consists of a number of operations. Each operation consumes some CPU time before generating a data access. If the data item requested by an operation is not found in the cache, a disk request is generated. The transaction issuing the disk request is blocked until the request is served. The CPU may switch to serving the next highest priority transaction when the current one is blocked on disk. Data conflicts are resolved according to the adopted concurrency control policy. We do not assume a main memory database since it is not common in commercial stock trading systems. However, we assume that the system may maintain a large cache to reduce the number of disk I/O.

6.2 Model Parameters and Performance Measures

The following table summarizes the parameters and the values used in the baseline setting.

Parameters	Baseline Values
Database size	1250 data items
Number of base items	1000
Number of derived items	250
Mean inter-arrival time of soft real-time transactions (SIA)	8 ms
Mean inter-arrival time of non-real-time transactions (NIA)	100 ms
Transaction length of soft real-time transactions	1 – 9 operations
Fan-out	0 – 8
Transaction length of non-real-time transactions	2 – 15 operations
Probability of write operation in a soft real-time transaction	0.5
Probability of write operation in a non-real-time transaction	0.5
Probability of accessing base items of a non-real-time transaction ($prob_base_item$)	0.5
Slack range of soft real-time transactions	1.5 – 3
Number of CPUs in the system	4
CPU execution time for an operation	Light computation (75%): 2 ms to 4 ms heavy computation (25%):

	5 ms to 15 ms
Number of disks in the system	4
CPU scheduling policy	EDF
CPU time to update a data item at memory	0.1 ms to 0.2 ms
Disk access time for retrieving/updating a data item	5 ms to 10 ms
Disk scheduling policy	EDF
Buffer hit probability	0.5
CPU time for checking a lock	0.1 ms
CPU time for validating a data item	0.1 ms

Table 7: Model parameters and the baseline settings

The database size and the number of base items are set based on the stock market characteristics in Hong Kong. There are about 1000 stock items in the Hong Kong Stock Exchange. Different investment companies may keep different kinds of derived information. Here, we set the number of derived data items to 250. Note that although the total number of stock items (1000) is not large in the simulation experiments, the total amount of other information in the database (e.g., trends of the stock prices, data for trading analysis, customer profiles, etc) can be very large. The mean inter-arrival time of soft real-time transactions is chosen to be 8ms. Since most of the soft real-time transactions are assumed to be stock update transactions, 8ms inter-arrival time represents an arrival rate of 125 updates/sec which is similar to the average arrival rate of stock updates in the Hong Kong Stock Exchanges. The arrival rate of non-real-time transactions is difficult to determine, and different systems will have different non-real-time transaction workload. In this paper, we assume that most of the non-real-time transactions are for decision-support queries, which arrive at a much lower rate comparing to the stock updates. Hence, in the baseline setting, we set the inter-arrival time to 100 ms.

We assume that the CPU processing time for different operations can be very different. For example, some operations may perform simple updates, and others may be complicated analysis. To simplify the analysis, we assume that there are two types of operations: light computation and heavy computation. For light computation, a CPU time of 2 to 4 ms is assumed. For heavy computation, a CPU time of 5 ms to 15 ms is assumed. Similar to many previous studies for RTDBS, the deadline of a soft real-time transaction, T , is defined according to the expected execution time of the transaction [AGM92, DMKV96, HCL92, U95, U98] such as:

$$Deadline = ar(T) + \text{slack time}$$

where *slack time*: $pex(T) \times (1 + SF)$;

SF : the slack factor which is a random variable uniformly chosen from a slack range;

$ar(T)$: the arrival time of transaction T ;

$pex(T)$: the predicted execution time of T .

Although the deadlines of soft real-time transactions in commercial stock database systems are defined based on the application requirements instead on the expected execution time, the use of the above equation can give us an indication of the tightness of the deadlines. We assume that the value (or usefulness) of a soft real-time transaction decreases rapidly after its deadline, and its value will be reduced to zero at a certain point in time after its deadline. Since different types of RTDBS will have different value functions for real-time transactions, we do not measure the total values returned from completing the real-time transactions. Instead, we measure the lateness and miss rate of the transactions. They are defined in the following table.

Miss Rate (MR) (for soft real-time transactions only)	the number of deadline-missing soft real-time transactions / the total number of soft real-time transactions generated.
Lateness (for soft real-time transactions only)	Σ completion time of a missed deadline soft real-time transaction t – deadline of t / the total number of deadline-missing soft real-time transactions
Soft real-time restart rate	the number of restarts of soft real-time transactions / the total number of soft real-time transactions generated from the generator

Table 8: Performance measures for soft real-time transactions

Furthermore, we assume that a soft real-time transaction, T , will become totally useless after the time point of $ar(T) + \text{useless rate} \times \text{slack time}$. At that time point, the transaction will be aborted since it is useless. The baseline value for the useless rate is chosen to be 1.3 since most soft real-time transactions will be totally useless soon after their deadlines. In the experiments, we have investigated the impact of the useless rate on the protocol performance.

The following table summarizes the performance measures for non-real-time transactions.

Response time	The average response time of all of non-real-time transactions
Non-real-time restart rate	The number of restarts of non-real-time transactions / the total number of non-real-time transactions generated

Table 9: Performance measures for non-real-time transactions

6.3 Performance Results and Discussions

The simulator is implemented in CSIM-18, which is a simulation tools for the C programming language. Each simulation run completes 50,000 soft real-time transactions. The length (50,000 transactions) is determined from a number of trial runs until the results are stable, e.g., less than 0.2% of difference. In the simulation experiments, we have compared the performance of the protocols IM, IM-DU and IM-ED with MHP-2PL, MOCC-W50 and MCC. The design of the MCC is similar to that used in the simulation experiments in [TSH96] except that we use OCC-W50 instead of OCC-wait in order to have a better comparison. (Note that as shown in [HCL92], OCC-W50 gives a generally better performance than OCC-Wait.) The concurrency control method for the non-real-

time transactions is using 2PL in MCC. Due to space limitation, we only present the results that illustrate the most important aspects of the different protocols.

6.3.1 Performance in a Real-time Environment

In this set of experiments, we study the performance of the six protocols, MHP-2PL, MOCC-W50, MCC [THS96], IM, IM-ED, IM-DU, in a real-time environment. In particular, we assume that the system is supported with a “perfect” real-time priority scheduler using EDF for both CPU and disk scheduling.

Figures 2, 3 and 4 show the performance of the protocols at different inter-arrival times of soft real-time transactions (different soft real-time transaction workloads). As we can see in Figures 2 to 4, IM-DU consistently gives the best performance for the soft real-time and non-real-time transactions (smaller values of MR, response time and lateness) at different soft real-time transaction workloads, especially at a heavier workload.

Although the performance of MOCC-W50 is similar to IM-DU when the workload is not heavy (e.g., > 8 ms), its performance is poor at heavy workloads. It is because it uses a restart approach to resolve data conflicts. Both the number of restarts and restart overheads are high, and a lot of resources are wasted on restarted transactions especially at heavy workload. Consequently, the total workload in the system will be very heavy, and the time required to complete the soft real-time transactions and non-real-time transactions becomes much longer. We can see in Figure 3 that the performance of non-real-time transactions is seriously affected and its response time becomes very large at heavy soft real-time workload. Since the impact of transaction restarts on system performance in MOCC-W50 is smaller at a lighter workload, the soft real-time transactions can enjoy a good performance due to a smaller number of useless restarts. When the workload is not heavy, the MR of MOCC-W50 is slightly smaller than that of MHP-2PL, which suffers from the problems of harmful wait, useless wait and useless restart.

The performance of MCC is similar to IM-DU with IM-DU performs marginally better than MCC. The better performance of MCC and IM-DU comparing with MOCC-W50 and MHP-2PL, suggests that a pure OCC or a lock-based strategy is not suitable to MSRTDBS. The good performance of IM-DU is due to several reasons. Firstly, it has less useless restarts as it uses the optimistic approach to resolve the inter-class data conflicts and the intra-class data conflicts amongst the soft real-time transactions. Secondly, the restart overheads are also lower than that in MOCC-W50 due to the deferred updates of the non-real-time transactions. Thirdly, deferring the updates of the non-real-time transactions also help to reduce the number of restarts of non-real-time transactions. A non-real-time transaction will be restarted only if its read data set overlaps with the write data set of some validating soft real-time transaction.

Although IM and IM-ED also integrate the optimistic and locking approaches as what is done in IM-DU, their performance is much worse than IM-DU. It is because the blocking of non-real-time transactions may increase their probability of being restarted by soft real-time transactions (higher probability of harmful waits). The performance of IM-ED is slightly better than IM. This indicates that restart overhead is one of the most important

factors on their performance. Early detection of data conflicts is important in reducing the restart overhead although it increases the number of useless restarts at the same time. The poor performance of IM and IM-ED suggested that a simple integration of OCC and 2PL may even degrade the system performance since the non-real-time transactions may have higher probability to be restarted due to the blocking by soft real-time transactions.

Figures 5 to 7 depict the results when the useless rate is increased to 1.5. The implication is that the soft real-time transactions will exist in the system for a longer time even though they have missed their deadlines. The consequence is that the soft real-time transactions will have a greater impact on the performance of the non-real-time transactions and the total system workload will be heavier. As can be observed in Figures 5 to 7, the relative performance of the protocols is similar to the results shown in Figures 2 to 4 except that the differences in their performance are larger especially when the soft real-time workload is lighter, i.e., greater than 7ms. As shown in Figure 6, the performance of the non-real-time transactions is much worse when a larger useless rate is used comparing with the results in Figure 3. This is due to the greater impact of soft real-time transactions on the non-real-time transactions when a larger useless rate is used. Note that although the mean response time of non-real-time transactions is also greater in IM-DU, it is less affected comparing with the non-real-time transactions under other protocols.

6.3.2 Performance of the Protocols in a Non-Real-time Environment

In this set of experiments, we assume that FCFS is used for the CPU and disk I/O scheduling. Figures 8 and 9 show the performance of the protocols. As expected, the performance of the soft real-time transactions are much worse than that in a real-time environment (Figure 5). On the contrary, the performance of the non-real-time transactions is improved as their waiting times for system resources and the probability of being restarted by soft real-time transactions becomes smaller.

Similar to the results in a real-time environment, IM-DU consistently gives the best performance for both real-time and non-real-time transactions. The differences comparing with other protocols are even greater (comparing with Figure 5) indicating that IM-DU is less affected by the non-real-time scheduling. One interesting observation is that the performance of the soft real-time transactions under MOCC-W50 and MCC is more affected by the non-real-time scheduling than MHP-2PL. (The performance of MHP-2PL, MOCC-50 and MCC becomes similar.) The main reason is that the cost of transaction restart is more expensive in a non-real-time environment. The non-real-time transactions, which have been restarted by real-time transactions, will compete with the soft real-time transactions in using system resources and increase the queuing delays of soft real-time transactions for the system resources. The result suggested that a real-time concurrency control protocol must be matched with the support of the underlying operating system. Otherwise, introducing a priority-cognitive scheduler in concurrency control may not be effective, and harmful effect may be resulted.

6.3.3 Performance in an Incomplete Real-time Environment

The purpose of this section is to explore a practical issue in the system implementation of the real-time concurrency control protocols. That is, any practical system can only have a limited number of priority levels for real-time scheduling. A priority mapping mechanism must be adopted to map the deadlines of soft real-time transactions to the limited number of priority levels. The mapped priorities of soft real-time transactions highly depend on the available number of priority levels in the system. If the system can only support a very limited number of priority levels, such as 2 priorities, many soft real-time transactions will be mapped to the same priority level even though they have different deadlines. Note that the mapping of transactions with different deadlines into the same priority level is different from using FCFS for CPU and resource scheduling. The mapped priority will be used for scheduling the transactions in using system resources and also be used for conflict resolution. When more transactions are mapped into the same priority, the probability of transaction restarts due to data conflicts will be smaller. Only the transactions with a large difference in deadlines may result in transaction restart. We use the priority mapping method described in Section 3, called EDREL, to map EDF to the fixed number of priorities in the system.

Figures 10 to 13 show the results when the system has different number of priority levels for the scheduling of the soft real-time and non-real-time transactions. Figure 10 shows that when the number of priority levels is smaller than 128, MR decreases with an increase in the number of priority levels. The better performance of the soft real-time transactions at more priority levels is due to two reasons. Firstly, if the number of priority levels is small, some urgent transactions, which should be assigned to higher priorities, are now assigned to the same priority level as other less urgent transactions. Therefore, they have to wait longer for system resources and have higher probability of missing their deadlines. Secondly, when the number of priority levels in the system is small, increasing the number of priority levels reduces the degree of interleaving in transaction executions. Higher priority transactions will be given preference in execution while lower priority transactions have to wait for the completion of the higher priority transactions. The consequence is smaller probability of data conflicts and smaller number of transaction restarts. The smaller number of transaction restarts at higher priority levels can be observed in Figure 11 and Figure 12.

Smaller number of transaction restarts also improves the performance of non-real-time transaction. As shown in Figure 11, the response time of non-real-time transactions decreases with an increase in priority levels starting from two priority levels. When the priority level is one, all the non-real-time and soft real-time transactions have the same priority. The response time of non-real-time is smaller than two priority levels since they are now less affected by the soft real-time transactions.

When we compare the results of different priority levels with EDF (the last point of each curve), it is a surprise to see that when the number of available priority levels is greater than 8, the performance of the protocols is slightly better than that under a “perfect” real-time scheduler (using EDF) as shown in Figure 10 and Figure 11. This is due to higher restart overhead for resolving data conflicts. When the number of priority levels is already large,

further increase in number of priority levels will not have any significant effect on reducing the degree of interleaving of transaction executions. Instead, it will increase the number of restarted transactions since now more transactions have different priorities. The result is more transaction restarts and greater degree of resource contention. In Figure 12 and Figure 13, we can see that the number of transaction restarts under EDF is slightly greater than when the number priority levels is 128.

To further confirm the argument, we repeat the experiments using a larger inter-arrival time of soft real-time transactions, 8ms, and a larger useless rate, 1.5. Consistent with the previous results, the performance of the protocols is slightly better than EDF when the number of priority levels is 128 as shown in Figure 14 and Figure 15. Furthermore, the differences in the performance between IM-DU and other protocols are much greater comparing with Figures 10 and 11.

From the above results, we can see that the priority scheduling can have serious impacts on the performance of the concurrency control strategies. Although transaction restart is good for real-time scheduling, as the probability of priority inversion will be smaller, the side effect is a heavier system workload especially when the workload is already heavy. Thus, in the design of real-time concurrency control protocols, it is important to identify the cost for meeting the timing requirement of higher-priority transactions and the data contention probability. We find that in some cases, using a “low resolution” for deadlines (i.e., mapping deadlines into priority levels greater than 8) sometimes improve the system performance because some transaction restarts are avoided and it is not necessary to use a ‘perfect’ real-time scheduler.

6.4 Summary of Performance Results

In this sub-section, we summarize the performance results discussed in the last sub-sections.

I. In a Real-time Environment

- (1) A single concurrency control strategy, i.e., 2PL and OCC, is not good enough for MSRTDBS.
- (2) Although OCC is good to soft real-time transactions, it is not suitable for a system with heavy workload since the restart overhead is high.
- (3) A simple integration of OCC (for soft real-time transaction) and 2PL (for non-real-time transactions) may be even worse than using a pure OCC or 2PL method when the workload is heavy since the blocking of non-real-time transactions may introduce more transaction restarts. Reducing the restart overhead can greatly improve its performance (i.e., in IM-DU).
- (4) If the life-spans of soft real-time transactions are longer, the cost of transaction restarts will be more expensive and further degrade the performance of the protocols.
- (5) Comparing with other tested protocols, IM-DU gives the best performance to soft real-time transactions and at the same time can minimize the impact on non-real-time transactions. The improvement with IM-DU is greater if the life-spans of the soft real-time transactions are longer.

II. In a Non-Real-time Environment

- (1) The impact of non-real-time scheduling on the optimistic method is greater than lock-based methods since the cost of transaction restart will be more expensive in a non-real-time environment.
- (2) IM-DU gives the best performance to both soft real-time and non-real-time transactions comparing with other tested protocols and it is less affected by the non-real-time scheduling.

III. In an Incomplete Real-time Environment

- (1) The number of priority levels in the system plays the key role on the performance of a real-time concurrency control protocol.
- (2) The performance under EDREL is better than the pure deadline-driven priority assignment policy, EDF, when the number of priority levels is not very small and when the workload is heavy and the restart overhead is high. It is because EDF generates more transaction restarts comparing with EDREL. This suggests that using a “low resolution” scheduler may also be more suitable for MSRTDBS.
- (3) Although supporting real-time data access is important in a MSRTDBS, the performance of a system may be better even if some non-real-time data accesses are allowed since its performance could be highly sensitive to transaction restart overheads.

7 Conclusions and Future Works

Real-time concurrency control has been an active research topic in past decades. Various concurrency control protocols have been proposed for different types of real-time database systems. With the increasing demand of response time requirements, many advanced database applications now must provide real-time performance to certain services. A *mixed soft real-time database system (MSRTDBS)* is a database system consisting of both soft-real-time and non-real-time transactions. Since MSRTDBS captures the needs of many advanced database applications in supporting soft real-time transactions in traditional database systems, the design of proper concurrency control mechanisms for MSRTDBS is of paramount importance. The techniques proposed for the concurrency control of soft (or firm) real-time transactions may not be suitable to MSRTDBS due to the existence of non-real-time transactions because their data-conflict resolution mechanisms may significantly affect the performance of non-real-time transactions.

In this paper, we have explored the performance of lock-based and optimistic protocols for MSRTDBS. Based on their characteristics, we propose new integrated methods IM, IM-ED and IM-DU, which possess the benefits of both the lock-based and optimistic approaches. The purpose of the protocols proposed in this paper is to support real-time scheduling for soft real-time transactions and, at the same time, to maintain a good performance for non-real-time transactions, e.g., in terms of the mean response time. Extensive simulation experiments have been conducted to demonstrate the performance of the proposed protocols, as compared to MHP-2PL, MOCC-W50 and

MCC, under real-time, non-real-time, and incomplete real-time environments. The experiments are based on a stock trading system, which is a typical RTDBS application with mixed soft real-time and non-real-time transactions. It has been shown that IM-DU gives the best overall performance in most of the situations as it wisely combines the benefits of the optimistic approach and lock-based approach. However, the poor performance of IM and IM-ED suggested that a simple integration of 2PL and OCC approaches may even degrade the system performance since the restart probability of non-real-time transactions will be higher due to the blocking by soft real-time transactions. The major performance problem of MOCC-W50 is due to a large number of transaction restarts and a significant amount of workload wasted on restarted transactions. The performance problem of MHP-2PL is caused by the high probability of useless restarts and useless wait. It may result in poor performance, compared to the optimistic protocols, when the system workload is not heavy.

A critical implementation issue is that most MSRTDBS may not be provided with a “perfect” real-time scheduler, which is required by many previously proposed real-time concurrency control protocols. Under a non-real-time environment (or an “incomplete” real-time environment), the performance of the protocols proposed in the previous research may be undermined. We have explored the performance of the proposed protocols in an incomplete real-time environment where the system can only support a limited number of priority levels. An important observation is that increasing the priority levels (which are used to support real-time transactions) may even make the performance of the whole system, including soft real-time transactions, worse, due to large number of restarted transactions. This observation indicates that in the implementation of real-time concurrency control protocols, it is not necessary to use a “perfect” real-time scheduler. A general-purpose operating system, which can support several priority levels, may give a similar or even a better performance.

We expect that MSRTDBS will become increasingly common in large mobile applications that require mobile data access. We will extend our protocols to provide concurrency control services for a mobile MSRTDBS. Finally, we will further study how the workloads of soft real-time and non-real-time transactions can be dynamically balanced in order to attain a better overall performance. For example, we can reduce the admission rate of soft real-time transactions if the system cannot meet the statistical timing requirements of non-real-time transactions; similarly, the number of non-real-time transactions can be limited if the deadline miss rate of soft-real-time transactions exceeds a certain tolerance level. We believe that the importance of MSRTDBS will be increasingly prominent, and its unique characteristics are worth for further study.

References

- [AG89] Abbott, R. and H. Garcia-Molina, “Scheduling Real-time Transactions with Disk Resident Data”, in Proceedings of the 15th VLDB Conference, pp. 356-396, 1989.
- [AG92] Abbott, R., & Garcia-Molina, H., “ Scheduling Real-time Transactions: A Performance Evaluation”, ACM Transactions on Database Systems, vol. 17, no. 3, pp. 513-560, 1992.

- [AGK94] 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.
- [AGK95] 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.
- [B96] Bestavros, A., "Advances in Real-time Database System Research", ACM SIGMOD Record, vol. 25, no. 1, 1996.
- [BN96] Bestavros, A. and S. Nagy, "Value-Cognitive Admission Control for RTDBS, in Proceedings of Real-Time Systems Symposium, 1996.
- [BHG87] Bernstein, P.A., Hadzilacos, V., & Goodman, N., Concurrency Control and Recovery in Database Systems, Addison-Wesley, Reading, Mass., 1987.
- [BLS97] Real-time Database Systems: Issues and Applications, edited by A. Bestavros, K.J. Lin, and Sang H. Son, Kluwer Academic Press, 1997.
- [CSKT91] Chen, S., J.A. Stankovic, J.F. Kurose and D. Towsley, "Performance Evaluation of Two Disk Scheduling Algorithms for Real-time Systems", Journal of Real-time Systems, vol. 3, pp. 307-336, 1991.
- [DMKV96] 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.
- [DSK00] A. Datta, S.H. Son, and V. Kumar, "Is a Bird in the Hand Worth More than Two in the Bush? Limitations of Priority Cognizance in Conflict Resolution for Firm Real-Time Database Systems", IEEE Transactions on Computers, vol. 49, no.5, 2000.
- [G95] Gallmeister, B.O., Programming for the Real World POSIX.4, O'Reilly & Associates, Inc., 1995.
- [HCL90] Haritsa, J.R., Carey, M.J., and Livny, M., On Being Optimistic about Real-Time Constraints, Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 331-343, 1990.
- [HLC91] Haritsa, J.R., Carey M.J. & Livny, M., "Earliest Deadline Scheduling for Real-Time Database Systems", in Proceedings of Real-Time Systems Symposium, pp. 232-242, 1991.
- [HCL92] Haritsa, J.R., Carey M.J. & Livny, M., "Data Access Scheduling in Firm Real-time Database Systems", Journal of Real-Time Systems, vol. 4, no. 3, pp. 203-242, 1992.
- [HR00] J. R. Haritsa, and K. Ramamritham, "Real-Time Databases in the New Millenium", Real-Time Systems, vol. 14, no. 3, 2000.
- [HS99] Haritsa J. and S. Seshadri, "Real-Time Index Concurrency Control", IEEE Transactions on Knowledge and Data Engineering, 2000.
- [HSR92] Huang, J. 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.
- [KR99] Konana, P., and Ram S., "Semantic-Based Transaction Processing for Real-Time Databases: The Case of Automated Stock Trading", INFORMS Journal on Computing, vol. 11, no. 3, pp. 299-315, 1999.
- [KG93] 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.
- [KLACL99] Kao, B., Kam-yiu Lam, B. Adelberg, R. Cheng and T. Lee, "Updates and View Maintenance in Soft Real-Time Database Systems", in Proceedings of 1999 Conference on Information and Knowledge Management,

Kansas City, Nov 1999.

[KS96] Kim, Y-K, and Son, S.H., "Supporting Predictability in Real-Time Database Systems", Proceedings of Real-Time Technology and Applications Symposium, Brookline, Massachusetts, pp. 38-48, 1996.

[KWL99] Tei-Wei Kuo, Chih-Hung Wei & Kam-yiu Lam, "Real-Time Data Access Control on B-tree Index Structure", in Proceedings of IEEE International Conference on Data Engineering, Sydney, Mar 1999.

[L94] Lee, J., Concurrency Control Algorithms for Real-time Database Systems, Ph.D. Thesis, Department of Computer Science, University of Virginia, 1994.

[LKTL00] Kam-yiu Lam, Tei-Wei Kuo, Tsang N W H and Law G C K, "The Reduced Ceiling Protocol for Concurrency Control in Real-time Databases with Mixed Transactions", The Computer Journal, vol. 43, no. 1, January 2000, pp. 65-80.

[LKTL] Kam-yiu Lam, Tei-Wei Kuo, Wai-Hung Tsang and Gary C.K Law, "Concurrency Control in Mobile Distributed Real-time Database Systems", Information Systems, vol. 25, no. 4, June 2000, pp. 261-322.

[LL76] Liu, C.L., and Layland, J.W., "Scheduling Algorithms for Multi-programming in a Hard-Real-Time Environment", Journal of ACM, vol. 20, no. 1, pp. 41-64, 1976.

[LLK98] Lee, C.S., Lam, K.Y. and Kao, B., "Priority Scheduling of Transactions in Distributed Real-time Databases", Journal of Real-time Systems, vol. 16, pp. 31-62, 1998.

[ORP96] O'Neil, P., K. Ramamritham and C. Pu, "A Two-Phase Approach to Predictably Scheduling Real-time Transactions", in Performance of Concurrency Control Mechanisms in Centralized Database Systems, edited by V. Kumar, Prentice Hall, New Jersey, 1996.

[OS95] Ozsoyoglu, G. and R. Snodgrass, "Temporal and Real-Time Databases: A Survey", IEEE Transactions on Knowledge and Data Engineering, vol. 7, no. 4, pp. 513-532, 1995.

[R93] Ramamritham, K., "Real-time Databases", International Journal of Distributed and Parallel Databases, vol. 1, no. 2, 1993.

[SLC91] Son, S. H., Lin Y. & Cook, R.P., Concurrency Control in Real-Time Database Systems, in Foundations of Real-Time Computing Scheduling and Resource Management, edited by A. M. van Tilborg & G. M. Koob, Kluwer Academic Publisher, 1991.

[SKS96] N.R. Soparkar, Henry F. Korth and A. Silberschatz, Time-Constrained Transaction Management Real-Time Constraints in Database Transaction Systems, Kluwer Academic Publishers, 1996.

[SRL87] 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.

[SRSC91] Sha, L., Rajkumar, R., Son S.H. & Chang, C.H., "A Real-time Locking Protocol", IEEE Transactions on Computers, vol. 40, no. 7, pp. 793-800, 1991.

[TSH96] Thomas, S., S. Seshadri and J.R. Haritsa, "Integrating Standard Transactions in Real-Time Database Systems", Information Systems, vol. 21, no. 1, 1996.

[UB93] Ulusoy, O. and G.G. Belford, "Real-time Transaction Scheduling in Database Systems", Information Systems, vol. 18, no. 8, pp. 559-580, 1993.

[U98] Ulusoy, O. "Real-Time Data Management for Mobile Computing", Proceedings of International Workshop on Issues and Applications of Database Technology (IADT'98), Berlin, Germany, 1998.

[UB98] Ulusoy O. & A. Buchmann, "A Real-Time Concurrency Control Protocol for Main-Memory Database Systems", *Information Systems*, vol. 23, no. 2, pp. 109-125, 1998.

[YWLS92] Yu, P.S., Wu, K.L., Lin, K.J. & Son, S.H., "On Real-time Databases: Concurrency Control and Scheduling", *Proceedings of IEEE*, vol. 82, no. 1, pp. 140-157, 1994.

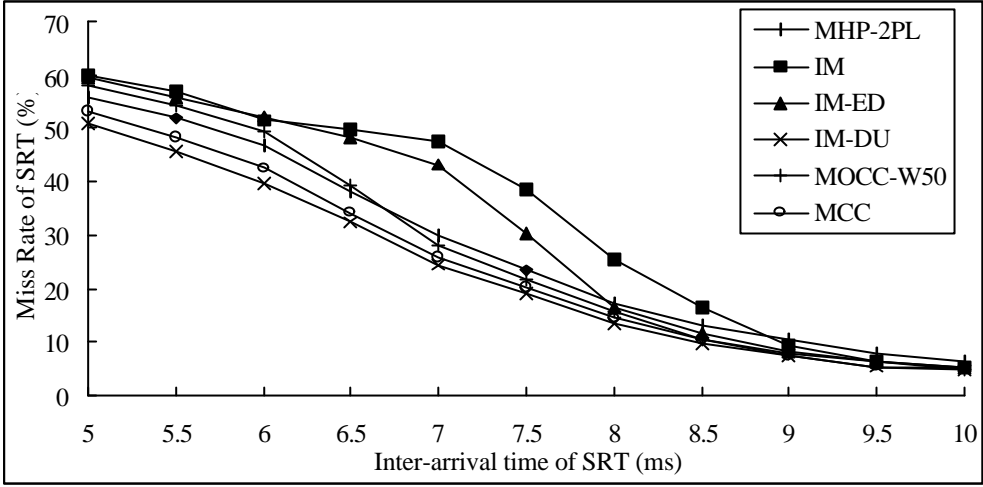


Figure 2. Impact of Inter-arrival time of SRT on Miss Rate under EDF (Inter-arrival time of NRT = 100ms, useless rate = 1.3)

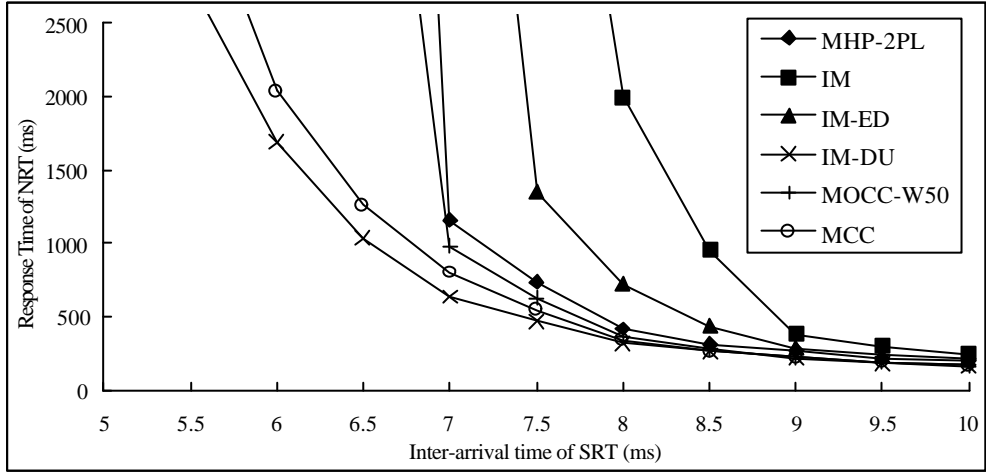


Figure 3. Impact of Inter-arrival time of SRT on Response Time of NRT under EDF (Inter-arrival time of NRT = 100m, useless rate = 1.3)

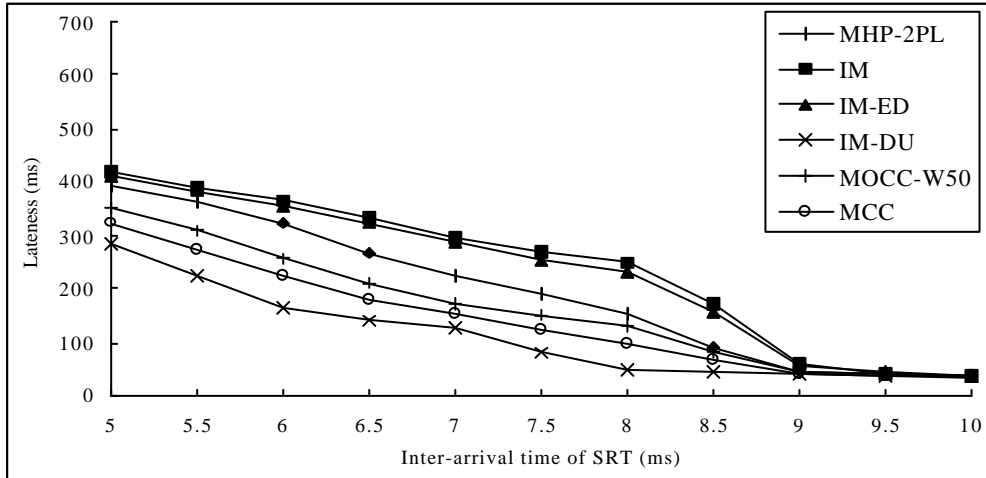


Figure 4. Impact of Inter-arrival time of SRT on Lateness under EDF (Inter-arrival time of NRT = 100ms, useless rate = 1.3)

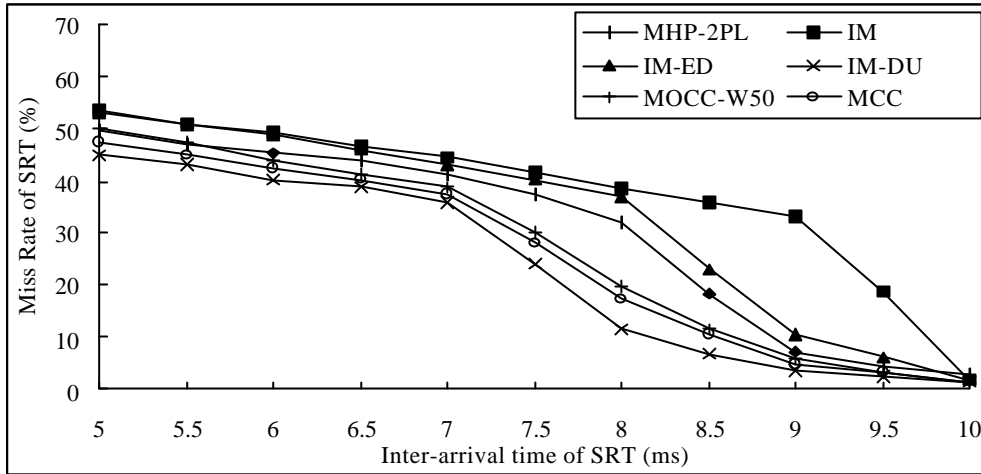


Figure 5. Impact of Inter-arrival time of SRT on Miss Rate under EDF (Inter-arrival time of NRT = 100ms, useless rate = 1.5)

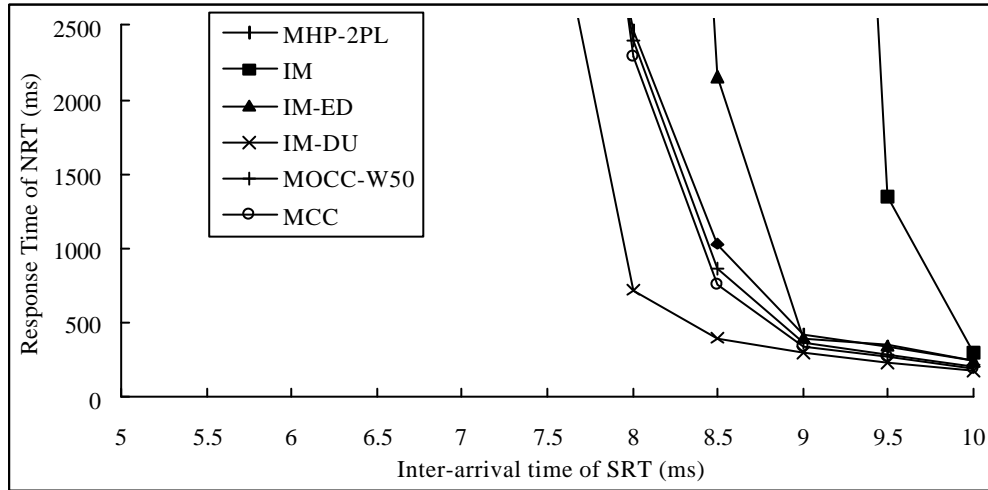


Figure 6. Impact of Inter-arrival time of SRT on Response Time of NRT under EDF (Inter-arrival time of NRT = 100m, useless rate = 1.5)

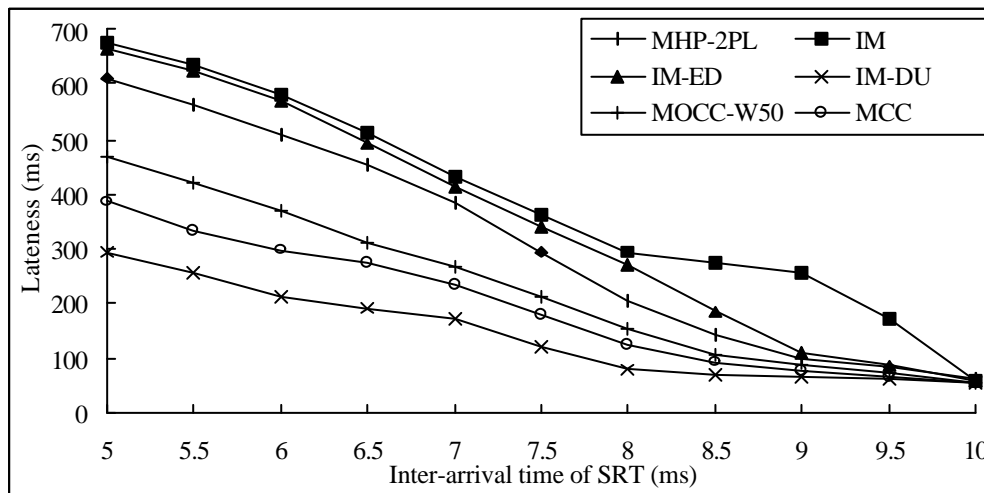


Figure 7. Impact of Inter-arrival time of SRT on Lateness under EDF (Inter-arrival Time of NRT = 100ms, useless rate = 1.5)

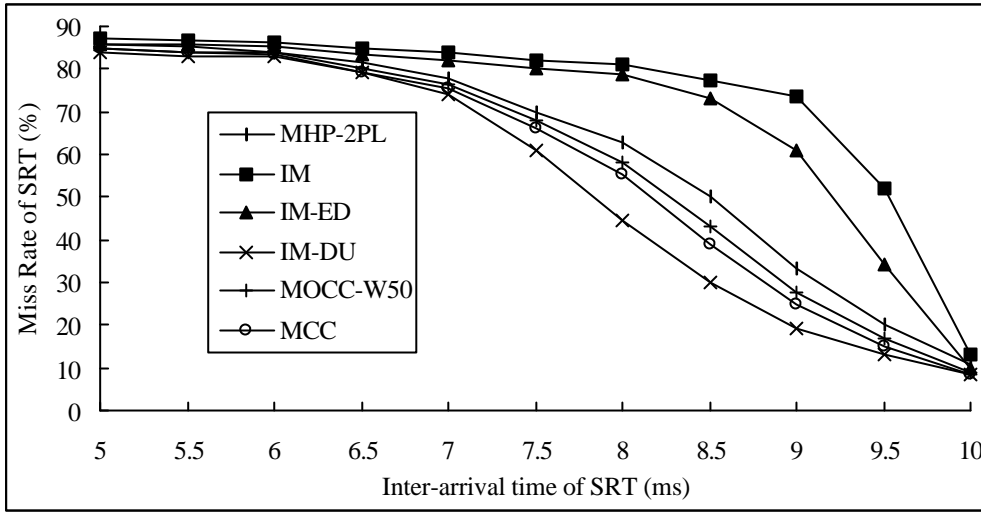


Figure 8. Impact of Inter-arrival time of SRT on Miss Rate under FCFS (Inter-arrival time of NRT = 100ms, useless rate = 1.5)

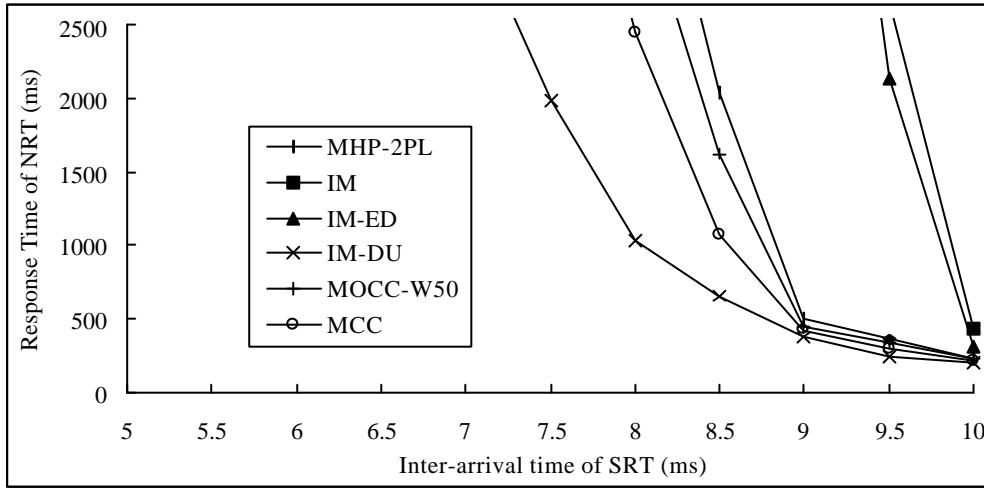


Figure 9. Impact of Inter-arrival time of SRT on Response Time of NRT under FCFS (Inter-arrival time of NRT = 100m, useless rate = 1.5)

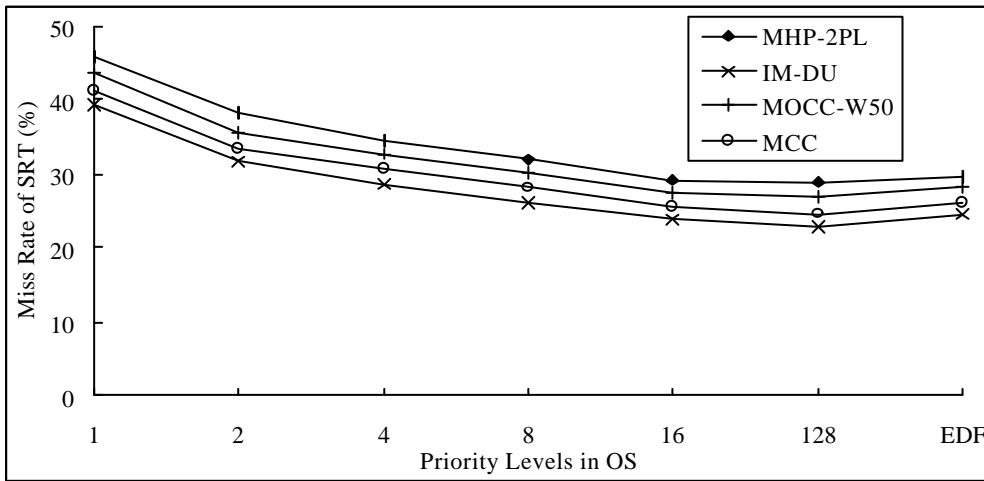


Figure 10. Impact of Priority Levels in OS on Miss Rate of SRT under EDREL (Inter-arrival time of SRT = 7ms, Inter-arrival time of NRT = 100ms, useless rate = 1.3)

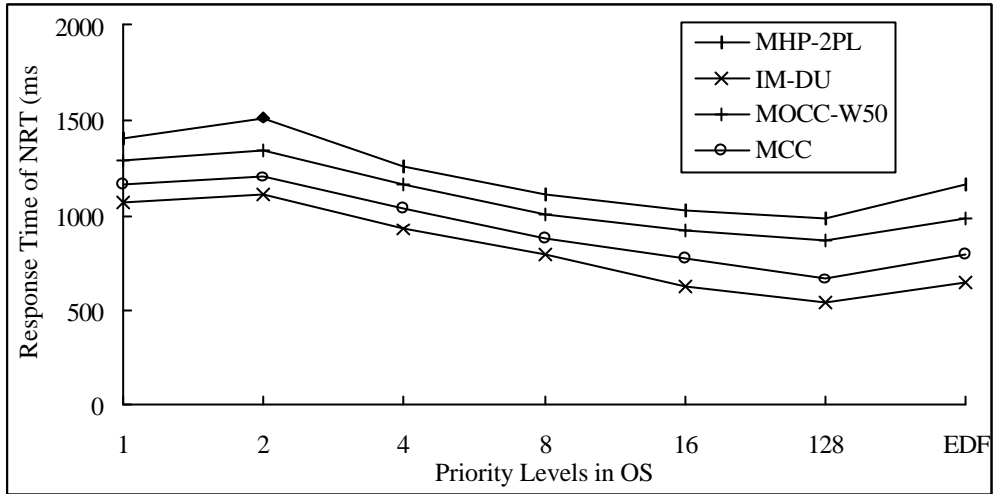


Figure 11. Impact of Priority Levels in OS on Response Time of NRT under EDREL (Inter-arrival time of SRT = 7ms, Inter-arrival time of NRT = 100ms, useless rate = 1.3)

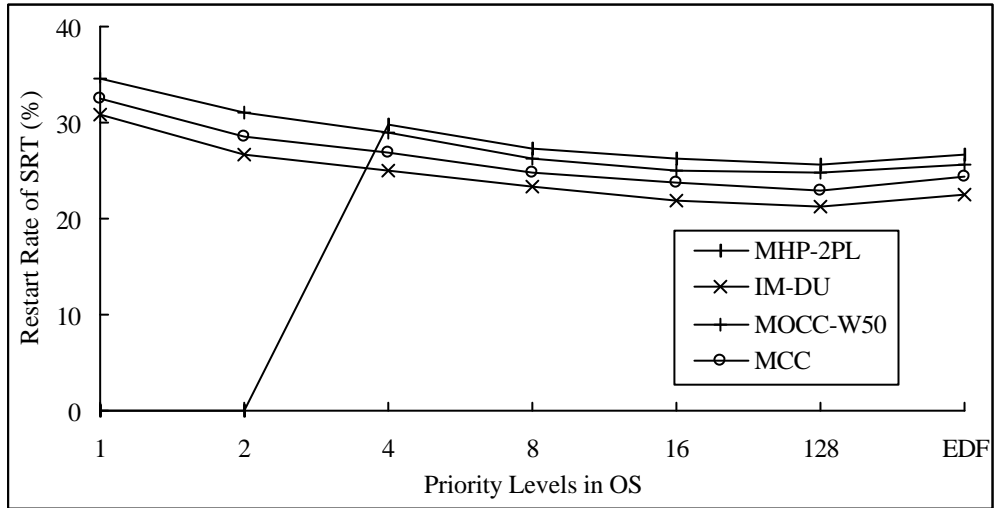


Figure 12. Impact of Priority Levels in OS on Restart Rate of SRT under EDREL (Inter-arrival time of SRT = 7ms, Inter-arrival time of NRT = 100ms, useless rate = 1.3)

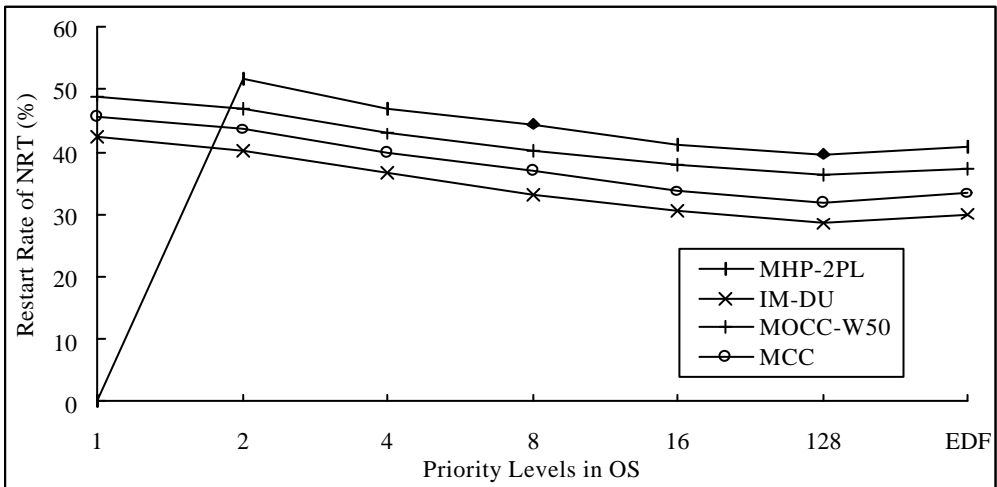


Figure 13. Impact of Priority Levels in OS on Restart Rate of NRT under EDREL (Inter-arrival time of SRT = 7ms, Inter-arrival time of NRT = 100ms, useless rate = 1.3)

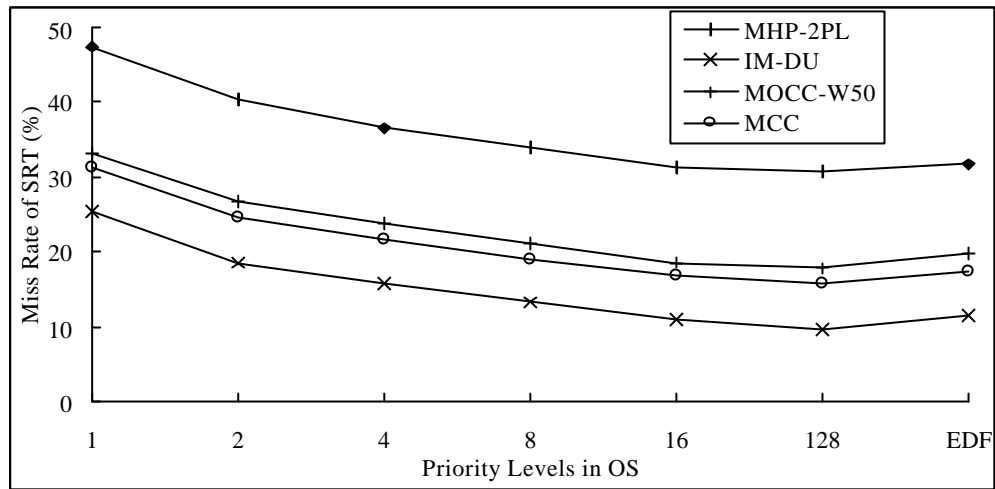


Figure 14. Impact of Priority Levels in OS on Miss Rate of SRT under EDREL
(Inter-arrival time of SRT = 8ms, Inter-arrival time of NRT = 100ms, useless rate = 1.5)

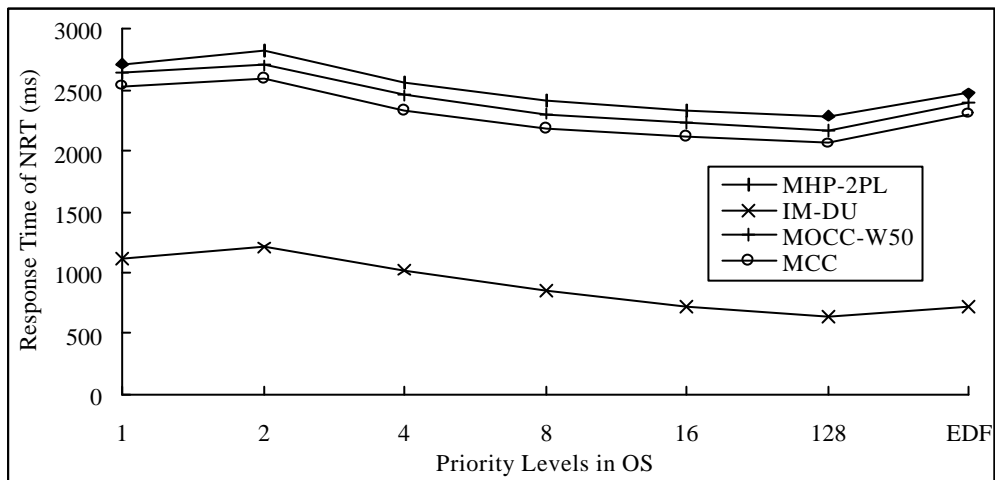


Figure 15. Impact of Priority Levels in OS on Response Time of NRT under EDREL
(Inter-arrival time of SRT = 8ms, Inter-arrival time of NRT = 100ms, useless rate = 1.5)