

# The Reduced Ceiling Protocol for Concurrency Control in Real-time Databases with Mixed Transactions

Kam-yiu Lam<sup>1</sup>, Tei-Wei Kuo<sup>2</sup>, Wai-Hung Tsang<sup>1</sup> and Gary C.K. Law<sup>1</sup>

Department of Computer Science<sup>1</sup>  
City University of Hong Kong  
83 Tat Chee Avenue, Kowloon  
HONG KONG  
email: [cskylam@cityu.edu.hk](mailto:cskylam@cityu.edu.hk)

Department of Computer Science and  
Information Engineering<sup>2</sup>  
National Chung Cheng University  
Chiayi, 621 Taiwan, ROC  
email: [ktw@cs.ccu.edu.tw](mailto:ktw@cs.ccu.edu.tw)

## Abstract

*This paper proposes a real-time concurrency control protocol called Reduced Ceiling Protocol (RCP) for real-time database systems (RTDBS) consisting of hard and soft real-time transactions. The schedulability of hard real-time transactions can be improved by bounding the blocking time from soft real-time transactions. Different concurrency control strategies are proposed to resolve data conflicts between different combinations of hard and soft real-time transactions, and the properties of the RCP schedules are shown. In the RCP, methodologies are proposed to reduce the number of aborts for soft real-time transactions due to data conflicts with hard real-time transactions. Simulation experiments have been performed to study the performance of the RCP as compared with the optimistic concurrency control with wait 50 (OCC wait-50) under different workloads of soft real-time transactions, various ratios of read/write operations, and deadline constraints. It has been found that the RCP can not only guarantee the performance of hard real-time transactions but also reduce the number of deadlines missed of the soft real-time transactions under all situations.*

Keywords: Real-time database systems, concurrency control protocols, conflict resolution, soft and hard real-time transactions

## 1 Introduction

In the past decade, the research on *real-time database systems (RTDBS)* has received a lot of interest [1,2,3]. Researchers have proposed various concurrency control protocols to meet transaction deadlines and, at the same time, to maintain database consistency [4,5]. Scheduling algorithms, such as the earliest deadline first (EDF) and the rate monotonic algorithm [6] were used to satisfy the timing constraints of real-time transactions, and concurrency control protocols, such as two phase locking [7] and optimistic concurrency control (OCC) [7], were used to guarantee database consistency. In particular, the well-known priority ceiling protocol (PCP), which is for the scheduling of tasks in hard real-time systems [6,8], has been extended for the scheduling of hard real-time transactions [9]. Sha, Rajkumar, and Lehoczky [10] proposed the PCP in which processes can inherit the higher priority of a process they block. The priority ceiling of a

resource is the priority of the highest priority process which may use the resource. A process's resource request is blocked if its priority is not higher than the priority ceiling of any resource which has been grabbed by another process but has not yet been released. With the PCP, the deadlines of all the hard real-time transactions can be guaranteed for a given workload and system configuration. Besides the concurrency control protocols proposed for hard real-time transaction systems, various protocols were proposed for soft and firm RTDBS, e.g., [4,11,12]. They used different heuristics to resolve data conflicts between transactions with different priorities. Although these protocols may not guarantee transaction deadlines, they can largely reduce the number of deadline violations.

Although various concurrency control protocols were proposed in past researches [4,5,10,13], most of them were merely designed for RTDBS with either hard or soft real-time transactions. They are not suitable to a large class of real-time database applications, which consists of both hard and soft real-time transactions. There is an increasing demand in processing hard and soft real-time transactions in real-time database systems. In a computer integrated manufacturing system, hard real-time transactions are responsible for controlling the production processes and the movement of robots. Soft real-time transactions are for other less critical tasks, e.g., higher-level management jobs. In [14], a detailed example of a mixed real-time database system is also illustrated for a medical information system. Hard real-time transactions are for monitoring the physical status of a critical patient from various sensor devices, such as the blood pressure, the heart beat rate, and the body temperature. Soft real-time transactions are for other conventional record keeping tasks or the retrieving of patient data. Two other examples of mixed real-time database systems are air traffic control systems [15] and steel plant production [16].

Although the protocols for hard real-time transactions, such as the PCP, can guarantee the deadlines of hard real-time transactions, they may not be used for the RTDBS having soft real-time transactions due to the restrictive assumptions of the protocols. When the protocols are applied to a RTDBS with a mixed transaction set (consisting of both hard and soft real-time transactions), the performance of soft real-time transactions may become very poor because of the conservative nature of the resource allocation policies and the dynamic data requirements of soft real-time transactions. Although researchers have proposed various concurrency control protocols for RTDBS, there is little work on the concurrency control in which *the deadlines of hard real-time transactions must be guaranteed, and the number of deadline violations for soft (and firm) real-time transactions must be minimized*. The paper aims at proposing a concurrency control protocol with a simple and efficient conflict resolution mechanism for the RTDBS which consists of both hard and soft real-time transactions.

The only difference between soft and firm real-time transactions is on the consequence of deadlines. In general, when a firm real-time transaction misses its deadline, it becomes useless (or even harmful as suggested by some researchers). Thus, a firm real-time transaction has to be aborted immediately after its deadline violation. On the contrary, it is still useful to complete a soft real-time transaction after it misses its deadline, although its usefulness may be much reduced. In this paper, we do not distinguish soft and firm real-time transactions as they are only different in the ways that the system handles them when they miss their deadlines. In the discussions of the RCP protocol, we use the term "soft real-

time transaction" as it is more common and general than firm real-time transactions in terms of scheduling. It should be noted that the RCP can also be applied for systems with firm and hard real-time transactions without any modifications, and all the properties remain.

Due to the different requirements for hard and soft real-time transactions, a single strategy for concurrency control is not sufficient for the RTDBS with both hard and soft real-time transactions. Different conflict resolution strategies must be used to resolve the data conflicts between different types of transactions since the importance of completing them is very different. In this paper, we propose a new concurrency control protocol, called the *Reduced Ceiling Protocol (RCP)*, which is an integration of the Priority Ceiling Protocol (PCP) and the optimistic concurrency control (OCC). Although the RCP is simple, the schedulability of hard real-time transactions is guaranteed by bounding the blocking time from soft real-time transactions. Mechanisms are also designed to reduce the number of unnecessary aborts for soft real-time transactions. We will consider different strategies for resolving different types of conflicts among hard and soft real-time transactions. The performance of the RCP is demonstrated by a series of simulation experiments.

An important variant of the PCP is the read/write PCP [9]. The read/write PCP and the PCP use the same principles for the concurrency control, except that the PCP only provides exclusive locks, and the read/write PCP provides read and write locks for read and write operations, respectively. Since the focus of the paper is not to study the PCP, we choose to use the PCP in the RCP in order to simplify the discussion and to focus this work on the mechanisms for resolving data access conflicts between soft and hard real-time transactions. It should be noted that all of the nice properties of the RCP will remain if the RCP adopts the read/write PCP instead of the PCP for the concurrency control of hard real-time transactions.

The remaining parts of the paper are organized as follows. Section 2 summarizes the related work. Section 3 defines the transaction model. Section 4 defines the Reduced Ceiling Protocol (RCP) and shows the important properties of the protocol. Section 5 reports the performance of the proposed protocol as compared with a real-time OCC protocol. Section 6 is the Conclusions.

## 2 Related Work

Real-time concurrency control of data access has received a lot of attention in the past decade [1,3,5,17,18]. Traditional concurrency control techniques cannot be directly applied to RTDBS because they do not consider the urgency of real-time transactions in data access. As an example, the Two Phase Locking (2PL) protocol [4,7] often introduces an unbounded number of priority inversions to higher priority transactions, where a priority inversion is a situation in which a higher priority transaction is blocked by a lower priority transaction due to data access conflict [9]. While many real-time concurrency control protocols proposed in past researches were based on traditional concurrency control protocols designed for non-real-time databases, they, in general, belong to two basic approaches: blocking or aborting [5].

The first approach is based on blocking to resolve data access conflicts and has been used in many lock-based protocols, e.g., [9,11]. Before a transaction accesses a data item, it must obtain a proper lock on the data item. Priority inheritance [10] is often used to adjust the priorities of transactions to bound the number of priority inversions for higher priority transactions, where priority inheritance is a mechanism that raises the priority of a lower priority transaction to the maximum priority of the transactions which are blocked by the lower priority transaction. To prevent the occurrence of deadlock and further limit the number of priority inversions, the idea of priority ceiling [10] was combined with the semantics of reads and writes, e.g., [9]. Although priority inheritance and ceiling were shown to be very effective in managing the number of priority inversions and guaranteeing the schedulability of hard real-time transactions, the data sets of all transactions must be known and fixed. Furthermore, because of the conservative nature of the definition of priority ceiling, transactions may suffer from unnecessary blocking due to high priority ceiling. The resulting concurrency of the system will be greatly affected. Another problem with the PCP is that it cannot be used for soft and firm RTDBS due to its restrictive assumptions. In RTDBS consisting of soft or firm real-time transactions, it is difficult to predict the data sets and the sizes of the transactions due to the dynamic properties of the transactions.

The second approach is based on aborting to resolve data access conflicts. In the High Priority Two Phase Locking (HP-2PL) [4], when the priority of the lock requesting transaction is higher than the priority of the lock holding transaction, the lock holding transaction will be aborted. In this way, the problem of priority inversion is resolved. On the other hand, if the lock requesting transaction is a lower priority transaction, it will be blocked until the higher priority transaction releases the lock. Aborting is also used in the optimistic concurrency control protocols to resolve data access conflicts in RTDBS, e.g., [4,12,19,20,21]. With optimistic concurrency control, the resolution of data conflicts is often delayed until a transaction has finished all of its operations. A validation test is performed to ensure that the resulted schedule is serializable if the validating transaction is allowed to commit. Various validation algorithms such as wait-50 and sacrifice were proposed [21]. In particular, Haritsa, et al. [21] have shown the superiority of the optimistic concurrency control protocols over the locking-based protocol. Other previous work [12,19,22] also had similar findings. Although their results did not quantify the schedulability of critical transactions against the aborting cost of a transaction system, it was noted that aborted transactions were often restarted, and an abundance of system resources were wasted in servicing aborted transactions. In [12,19], in order to reduce the amount of resources wasted on the aborted transactions, the concept of dynamic adjustment of serializability order was introduced to reduce the number of transaction aborts. In [20], the tradeoff between aborting and blocking cost is considered. A series of abort-oriented protocols was introduced with on-line and off-line analysis of aborting effects on the schedulability of transaction systems.

In recent years, issues on the concurrency control of mixed transactions in RTDBS have received growing attention as it is more practical for many real-time applications. In [14], a framework for a mixed RTDBS model has been defined in which transactions with different characteristics and criticality are classified into different classes. Different principles are used to schedule different transaction classes. However, in their framework, it has been assumed that the hard real-time transactions (class I transactions as defined in their work) will not have any data conflicts with other class I transactions and with soft real-time transactions. In [23], an adaptive algorithm is suggested to schedule multi-class

transactions in a RTDBS. However, it assumes that the transactions have no hard real-time deadlines, and the proposed algorithm cannot guarantee the deadlines of the transactions. In [13], the dynamic adjustment concept has been extended for the scheduling of transactions with different priorities. However, the protocol still cannot guarantee the deadlines of the critical transactions. Thus, it is still not suitable for RTDBS in which the schedulability of hard real-time transactions must be guaranteed.

In [24], a two-level concurrency control framework is proposed for concurrency control in a real-time database system with a mix of soft real-time and non-real-time transactions. In the framework, a master concurrency controller is proposed to detect the possible inter-class data conflicts. Any intra-class data conflicts amongst transactions in the same class are detected and resolved by individual schedulers. Data conflicts between soft real-time and non-real-time transactions are resolved by performing a serialization checking on the time-stamps of transactions. The framework provides an excellent architecture in extending conventional non-real-time database system to mixed real-time database system by adding the master concurrency controller and new schedulers for real-time concurrency control. Different concurrency control protocols can be incorporated into the systems by replacing each individual scheduler.

The scheduling of mixed tasks in real-time systems also has received a lot of interest in the recent years. In [25], the deferrable server algorithm is used to schedule completely preemptable periodic and aperiodic tasks. A slack stealing algorithm is proposed in [26] to schedule soft real-time aperiodic tasks in a real-time system in which hard real-time periodic tasks are scheduled using a fixed-priority algorithm. In [27], two different optimal algorithms are proposed to schedule soft real-time aperiodic tasks in the systems with hard real-time periodic tasks in which dynamic priorities are assigned to the soft real-time tasks. Although these algorithms are suitable for mixed real-time systems, when they are applied in RTDBS with mixed transactions, the data consistency problem has to be solved.

### 3 Transaction Model

We are interested in uniprocessor RTDBS consisting of *hard real-time* and *soft real-time* transactions. Hard real-time transactions are periodic transactions. Soft real-time transactions are sporadic transactions. It is assumed that each (hard or soft) transaction is defined as a sequence of operations and assigned a criticality and a priority. The system distinguishes hard real-time transactions from soft real-time transactions by their criticality levels. Hard real-time transactions will be assigned a higher criticality level.

The processing of an operation in a transaction requires the access of one or more data items in the database. The priority of a soft real-time transaction is defined according to its deadline such as that based on the Earliest Deadline First (EDF) scheduling. The transaction, which has a closer deadline, will be assigned to a higher priority. The priority of a hard real-time transaction is defined explicitly by application engineers based on the characteristics of the transactions such as their periods. A transaction with a shorter period may be assigned to a higher priority. Suppose that the priorities of hard real-time transactions are always higher than the priorities of all of the soft real-time transactions. A transaction will be

assigned to use the CPU if it is ready and has the highest priority among all of the ready transactions, and it is not blocked due to data conflict. Any violation of the timing constraints of hard real-time transactions may result in a catastrophe, while missing the timing constraints of soft real-time transactions will only result in the degradation of system performance.

When a soft real-time transaction is executing in its validation and write phases, it is assigned a priority higher than all the hard and soft real-time transactions (Please see Section 4.1.2 and Lemmas 1 and 2 in Section 4.2). Such a two-level priority assignment scheme helps bound the blocking time of hard real-time transactions from soft real-time transactions. Note that the proposed priority assignment scheme and the *Reduced Ceiling Protocol* (RCP) defined in the following section aim at providing a simple and efficient conflict resolution mechanism between hard and soft real-time transactions. Optimization techniques similar to slack stealing [26,28] can be used to further improve the performance of the RCP. However, in the application of slack stealing in a RTDBS, special considerations have to be paid to ensure that the final execution schedule is serializable [7].

Since hard real-time transactions are highly critical, it is assumed that the timing constraints, read sets, and write sets of hard real-time transactions are known before the system operates. The read set and write set of a transaction are the set of data items which may be read and written by the transaction respectively. Although the deadlines of soft real-time transactions are known upon their arrivals, it is assumed that, due to the dynamic nature of the soft real-time transactions, there is no way to determine which data items a soft real-time transaction may access until the corresponding operation occurs. Furthermore, the execution time requirements and the number of operations in a soft real-time transaction cannot be predicted. Their lengths may change during their execution<sup>1</sup>.

## 4 The Reduced Ceiling Protocol (RCP)

### 4.1 Overview of Reduced Ceiling Protocol (RCP)

The goal of the *Reduced Ceiling Protocol* (RCP) is to guarantee the schedulability of hard real-time transactions and then to minimize the number of deadline violations of soft real-time transactions. Methods are designed in the RCP to reduce the number of aborts for soft real-time transactions due to data conflicts with hard real-time transactions. The RCP adopts different strategies to resolve data conflicts between different combinations of hard and soft real-time transactions, as shown in Table 1.

Lock Holder	Lock Request	
	Hard RT-trans	Soft RT-trans
Hard RT-trans	PCP	Blocked
Soft RT-trans	Granted	OCC

Table 1: Concurrency control strategies in the RCP

The schedulability of hard real-time transactions is ensured by reserving data items based on the PCP principles. The conflict resolution strategy amongst soft real-time transactions follows the optimistic approach. In order to minimize the blocking of hard real-time transactions by soft real-time transactions, hard real-time transactions are allowed to “preempt” soft real-time transactions over conflicts in data accesses. In order to reduce the number of deadlines missed of soft real-time transactions, they are allowed to hold their locks even though the locks are requested by hard real-time transactions if they are in the validation phase or in the write phase.

#### 4.1.1 Concurrency Control of Hard Real-time Transactions

Before a hard real-time transaction accesses a data item, it has to acquire the corresponding lock on the data item. It is required to lock and access data items in a two phase locking (2PL) fashion [7], i.e., a transaction may not request any new lock after it releases any obtained lock.

In the RCP, lock conflicts among hard real-time transactions are resolved using the strategies similar to that of the PCP [9] except that it has to go through a checking phase before it commits. The purpose of the checking is to resolve its data conflicts with soft real-time transactions and will be discussed in more details in Section 4.1.3.

In the RCP, similar to the PCP, each lock is defined with a *priority ceiling*. The priority ceiling of a lock is defined as the highest priority of the hard real-time transaction which may access the lock. The lock request of a hard real-time transaction will be denied, i.e., blocked, if its priority is not higher than the priority ceiling of all the locks currently being held by other hard real-time transactions. If the priority of the transaction is higher, then the lock request is granted. The lock operations of soft real-time transactions will not block any hard real-time transactions as the setting of locks for a soft real-time transaction will not affect the priority ceiling of the lock. The schedulability of hard real-time transactions should be verified in an off-line fashion. Lemma 3 and Theorem 3 in Section 4.2 can be used to guarantee the schedulability of hard real-time transactions.

#### 4.1.2 Concurrency Control of Soft Real-time Transactions

Soft real-time transactions are scheduled by an optimistic approach. The reason for using an optimistic approach to schedule soft real-time transactions is that many researchers [12,19,21,22] have found the superiority of optimistic approaches over lock-based approaches in scheduling soft real-time transactions. Furthermore, the performance of the RCP with an optimistic approach can be further improved by the application of dynamic serialization adjustment in the optimistic approach [13,19].

In the RCP, the execution of a soft real-time transaction is divided into three phases: (1) the read phase, (2) the validation phase, and (3) the write phase. During the read phase, a soft real-time transaction reads data items from the database but writes data items into its private work space, where a delay write procedure is adopted. Any data conflict

---

<sup>1</sup> One of the reasons is the triggering of transactions. In triggering of transaction, the length of a transaction will be suddenly increased.

among soft real-time transactions is ignored at the read phase. Data conflicts are resolved when any one of the conflicting transactions enters the validation phase.

During the validation phase, data conflicts are checked against all executing soft real-time transactions with the validating transaction using a forward validation scheme [12]. They also check against conflicts with all hard real-time transactions. (The details of the checking with hard real-time transactions will be discussed in the next sub-section, Section 4.1.3.) If there is any data conflict between the validating soft real-time transaction and any executing soft real-time transaction, the resolution method will follow the OCC wait-50 principles. In the wait-50 scheme, if the number of conflicting transactions with higher priorities is greater than 50% of the total number of conflicting transactions, the validating transaction will be blocked until it is smaller than 50%. Otherwise, all conflicting transactions will be restarted even though some of their priorities are higher than the priority of the validating transaction. Please note that other priority conflict resolution methods for optimistic methods such as OCCTI and Sacrifice [11] can be used to resolve the data access conflicts among soft real-time transactions so that the priorities of the conflicting soft real-time transactions will be considered in resolving the data conflict. In here, we assume it uses OCC wait-50 as we compare the RCP with the OCC wait-50 in the performance evaluation experiments.

Once a soft real-time transaction has passed its validation phase, it enters the write phase, and the actual writing of data items takes effect on the database. Since a soft real-time transaction is near completion when it is in the validation and write phases, it is assigned a priority higher than all of the hard and soft real-time transactions. The purpose is to finish it earlier so that the probability of data conflict with any other transactions will be lower. The assignment of a higher priority to the validation and write phases of a soft real-time transaction does introduce priority inversion to hard real-time transactions. However, as shown in Section 4.2, only one soft real-time transaction will be in the validation and write phases at any time, and the maximum number of blocking for a hard real-time transaction is limited to one soft real-time transaction.

In order to detect data conflicts between soft and hard real-time transactions, every soft real-time transaction is required to set a shared lock on a data item before it accesses the data item in its read phase. The discussion on detecting the lock conflicts will be discussion in details in the Section 4.3. Note that the locks of soft real-time transactions will not block any other soft real-time transactions. Also, the setting of shared locks will not affect the priority ceiling and the scheduling mechanism of hard real-time transactions. The shared locks will be released after the validation test. In the next sub-section, the interactivities between hard and soft real-time transactions will be discussed in more details.

#### **4.1.3 Concurrency Control between Hard and Soft Real-time Transactions**

In the RCP, hard real-time transactions are given preferences in using the data items as they are much more critical. When a soft real-time transaction requests a lock which is already locked by a hard real-time transaction, the request is blocked until the hard real-time transaction releases the lock. The reason for denying the lock request is that, even if the soft real-time transaction is allowed to set the lock and continue its execution, it is very likely that it cannot pass

its validation test and will be aborted due to the conflict with the hard real-time transaction.

If a hard real-time transaction requests a lock on a data item which is already locked by a soft real-time transaction, the lock request is granted, and the priority ceiling is set accordingly. However, the execution of the soft real-time transaction will not be affected at this time. The checking of data conflicts between the hard and soft real-time transactions is done at the validation phase of the soft real-time transactions or at the commit time of the hard real-time transaction whichever is earlier. When a validating soft real-time transaction discovers any data conflict with a hard real-time transaction, it will be aborted immediately.

When a hard real-time transaction wants to commit after it has completed all its operations, it will go through a checking phase. In the checking phase, all of its data items will be examined again to see whether there are any soft real-time transactions accessing the same data items. The system will abort all soft real-time transactions which are reading the data items that are write-accessed by the committing hard real-time transaction. However, if a soft real-time transaction is writing the same data item, the soft real-time transaction can continue its execution. It is because the *Thomas Write Rule* [7] is used to resolve write/write conflicts of transactions scheduled by the RCP. Since soft real-time transactions adopt a delay write principle for write operations, the serialization order of a write/write conflicting soft real-time transaction in a serialization graph [7] will always be after the hard real-time transaction if the hard real-time transaction commits first.

Please note that it is possible that a soft real-time transaction repeatedly waits in the queue for different conflicting hard real-time transactions. Such potential “indefinite postponement” for soft real-time transactions is the price paid for the deadline guarantee of hard real-time transactions. Although no aging mechanism is adopted for the priority adjustment of soft real-time transactions, due to the management of priority inversion time for hard real-time transactions, we must emphasize that one of the purposes in designing the RCP is to minimize the impacts of hard real-time transaction executions on soft real-time transaction executions, and the results of this approach are shown in the performance evaluation.

## 4.2 Properties of the RCP

The section is meant to show important properties of the RCP.

**Theorem 1** All the RCP schedules are serializable.

**Proof.** The correctness of this proof directly follows from the facts: (1) The serializability order among hard real-time transactions is guaranteed by the two phase locking mechanism [7]. (2) The serializability order between hard real-time transactions and soft real-time transactions is maintained by aborting or blocking of soft real-time transactions if they have data conflicts with hard real-time transactions. (3) The serializability order among soft real-time transactions is maintained at the validation phase of soft real-time transactions [7]. When a soft real-time transaction enters the validation phase, all conflicting soft real-time transactions have to be aborted in order to maintain the schedule to be serializable. When a hard real-time transaction wants to commit, a soft real-time transaction will be aborted by the hard real-time transaction if the soft real-time transaction is in conflict with the hard real-time transaction, and the soft real-time transaction is performed

before the hard real-time transaction. [7]. Q.E.D.

**Lemma 1** There can be at most one soft real-time transaction executing in its validation and write phases at any time.

**Proof.** Since soft real-time transactions have a lower priority than hard real-time transactions except in their validation and write phases, a soft real-time transaction is not possible to execute, and enter its validation and write phases unless no hard real-time transaction is active. Also, when a soft real-time transaction enters its validation and write phases, it runs at the highest priority in the system and will not suffer from any preemption from hard or soft real-time transactions. Therefore, there can be at most one soft real-time transaction executing in its validation and write phases at any time. Q.E.D.

**Lemma 2** At most one validation and write phase of a soft real-time transaction may block the execution of one request of a hard real-time transaction.

**Proof.** Lemma 1 shows that there can be only one soft real-time transaction executing in its validation and write phases at any time. Also, when the soft real-time transaction leaves its validation and write phases, active hard real-time transactions will start executing and will not suffer from any further blocking from soft real-time transactions. Q.E.D.

It is highly important to allow at most one soft real-time transaction in the validating phase and the write phase to block the execution of one request of a hard real-time transaction; otherwise, hard real-time transactions may be blocked for a long time and have no schedulability guarantee. Note that the schedulability of hard real-time transactions must be well justified. In order to guarantee that at most one soft real-time transaction can block the execution of one request of a hard real-time transaction, it is essential to show that there can be only one soft real-time transaction executing in its validation phase and write phase at any time. Theoretically, we can allow two soft real-time transactions with mutually exclusive data sets in the validating phase and write phase at the same time without violating any serializability criteria, but the two soft real-time transactions may both block a hard real-time transaction. Such blocking by two soft real-time transactions may double the blocking time, i.e., the priority inversion time, of hard real-time transactions and damage the schedulability of hard real-time transactions.

**Theorem 2** The RCP is deadlock-free.

**Proof.** Since the PCP and optimistic concurrency control do not generate deadlocks [7,10], only deadlocks involving hard and soft real-time transactions may exist under the RCP. Also, the only chance that a soft real-time transaction may block a hard real-time transaction is in its validation and write phases. Since a soft real-time transaction running in its validation and write phases does not need to request any more data items, there should not exist any circular waiting chain for a deadlock which includes hard and soft real-time transactions at the same time. In other words, the RCP is deadlock-free. Q.E.D.

**Lemma 3** The maximum number of priority inversion per request of a hard real-time transaction under the RCP is one.

**Proof.** Lemma 2 shows that each request of a hard real-time transaction may experience at most one time of “priority inversion” from soft real-time transactions because of their execution in the validation and write phases. Furthermore, since

hard real-time transactions are scheduled by the PCP, the maximum number of priority inversion from hard real-time transactions per request of a hard real-time transaction is equal to one according to the properties of the PCP. We claim that no request of a hard real-time transaction should experience both one priority inversion from some hard real-time transaction and another one priority inversion from some other soft real-time transaction. The correctness of the claim comes from the following observation: the reason that a request of a hard real-time transaction may block a request of another hard real-time transaction is because the former arrives at the system earlier than the latter. If a request of a soft real-time transaction also blocks the later hard real-time transaction request, the soft real-time transaction request must enter its validation and write phases before the later hard real-time transaction request arrives at the system. In fact, the soft real-time transaction request must also enter its validation and write phases before the former hard real-time transaction request arrives at the system (Please see the proof of Lemma 1). In other words, the soft real-time transaction request should block the former hard real-time transaction request instead of the later. A contradiction exists. Q.E.D.

Let a RTDBS consist of a set  $T_h = \{\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_n\}$  of hard real-time transactions and a dynamic set  $T_s = \{\mathbf{t}'_1, \mathbf{t}'_2, \dots, \mathbf{t}'_m\}$  of soft real-time transactions.  $c_i$  and  $p_i$  are the computation time and period of a hard real-time transaction  $\mathbf{t}_i$ , respectively. Suppose that hard real-time transactions in  $T_h$  are numbered in inverse order of their periods, and their priorities are assigned in the rate monotonic priority order. Lemma 3 implies that the maximum blocking time of a request of a hard real-time transaction is equal to the maximum of the time interval for a validation phase plus a write phase of soft real-time transactions and the priority inversion time of the hard real-time transaction under the PCP.

**Theorem 3** Let  $b_i$  be the blocking time of hard real-time transaction  $\mathbf{t}_i$  under the RCP.  $\mathbf{t}_i$  is schedulable if  $\sum(c_j/p_j) + (b_i/p_i) \leq n(2^{(1/n)} - 1)$ .

**Proof.** The correctness of this proof follows directly from the formula for the achievable utilization factor of the rate monotonic algorithm [9], where the blocking time of  $\mathbf{t}_i$  is modeled as the extra computation time for  $\mathbf{t}_i$ . Q.E.D.

Although soft real-time transactions are often unpredictable and their sizes can be much longer than those of hard real-time transactions, the blocking time of a hard real-time transaction by a soft real-time transaction is bounded. The reason is that only validating soft real-time transactions (i.e., soft real-time transactions executing in their validation and write phases) can block a hard real-time transaction. At the validation phase and the write phase, a soft real-time transaction is no longer unpredictable because all of its operations have completed (and its access pattern is known). Thus, the system will know which data objects the validating soft real-time transaction wants to update, and the time required for completing the validation phase and write phase is predictable. This information of soft real-time transactions can be used to bound the blocking time of the hard real-time transactions. Note that this paper aims at mixed scheduling of real-time transactions in real-time main-memory database systems.

If the time for the validation and write phases of a soft real-time transaction is too long and can cause the conflicting hard real-time transaction to miss its deadline, a simple extension of our protocol is to abort the validating soft real-time transaction if it is in the validation phase. However, once a soft real-time transaction has entered the write phase,

a mutually exclusive critical section must be enforced to maintain the integrity of the database and the hard real-time transaction must be blocked. Thus, in order to bound the blocking time, a checkup will be done before a soft real-time transaction enters its write phase to determine whether it should be aborted or it will be allowed to enter its write phase. With this extension, the main framework and design goal of the protocol remains and the only incurred cost is the possible deadline violations of soft real-time transactions due to aborting.

For the offline schedulability test of hard real-time transactions, it may not be necessary to use the worst-case execution time of both the validation and write phases of the soft real-time transactions. Instead, the worst-case time for the write phase (or other well-chosen time value) may be used. Different time values will affect the result of the schedulability test and the number of soft real-time transactions to be aborted. If the time required for the write phase of a soft real-time transaction is larger than the maximum tolerable blocking time of any hard real-time transaction, the soft real-time transaction can be considered for aborting when it wants to enter its write phase. However, once the soft real-time transaction enters its write phase, some hard real-time transactions may take the risk of missing their deadlines unless an efficient rollback mechanism is provided to recover the database from partially completed write phase.

### 4.3 Implementation Considerations

For the simplicity of system implementation, a lock table is maintained for the data items in the database to resolve the data conflicts between different types of transactions. Three types of locks are defined. They are pre-lock (P-lock), validation lock (V-lock) and exclusive lock (E-lock). The P-locks and V-locks are for soft real-time transactions and the E-locks are for hard real-time transactions. Since a soft real-time transaction may read or write a data item, there are two types of pre-locks. They are the pre-read-locks (PR-locks) and the pre-write-locks (PW-locks). E-locks and V-locks are exclusive locks. The compatibility of the locks is shown in table 2.

Lock Requester	Lock Holder			
	PR-lock	PW-lock	V-lock	E-lock
PR-lock	✓	✓	✗	✗
PW-lock	✓	✓	✗	✗
V-lock	✓	✓	✗	✗
E-lock	✓	✓	✗	✗

Table 2: 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 will set a P-lock on a data item before it accesses the data item. If it wants to read the data item, it will set a PR-lock. If it wants to write the data item, it will set a PW-lock. All the P-locks are compatible with each other. The lock request will be denied if a V-lock or an E-lock is already set on the same data item by another transaction. If there is a V-lock, 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. So, the lock requesting soft real-time transaction is not allowed to access the data item as validation phase and write phase have to be performed in a critical section [7]. If there is an E-lock on the data item, it means that a hard real-time transaction is accessing the data item. According to the RCP, soft real-time transactions are not allowed to access the data item and are blocked.

When a soft real-time transaction finishes its read phase, it will enter the validation phase and its priority will be raised up to be higher than that of the hard real-time transactions. In the validation phase, it will convert all its PR-locks and PW-locks to V-locks. Once the soft real-time transaction has converted all its P-locks to V-locks, the system will check data conflicts for the validating transaction against other soft and hard real-time transactions according to the principles defined in the RCP. The lock table will be examined for P-locks and E-locks. If there is an E-lock on the same data item as any of its V-locks, the validating transaction will be aborted. If the validating transaction has successfully converted all of its P-locks to V-locks and has passed the validation, then the soft real-time transaction enters the write phase. It will commit after releasing the V-locks at the end of the write phase.

A hard real-time transaction has to set an E-lock before it is allowed to access the data item. The condition for granting an E-lock follows the principles of the RCP, as defined in the Section 4.1.1. It will check with the priority ceiling of all of the E-locks in the system and also the V-lock on the same data item. At the commit time of a hard real-time transaction, it will release all of its E-locks. At the same time, it will check for PR-lock on the same data item. If a soft real-time transaction has set a PR-lock on any of its accessed data items, the soft real-time transaction will be aborted.

## 5 Performance Study

### 5.1 Real-time Database Model

The RTDBS model consists of six components. They are the transaction generators, the scheduler, the CPU, the ready queue, the block queue and the database as shown in Figure 1.

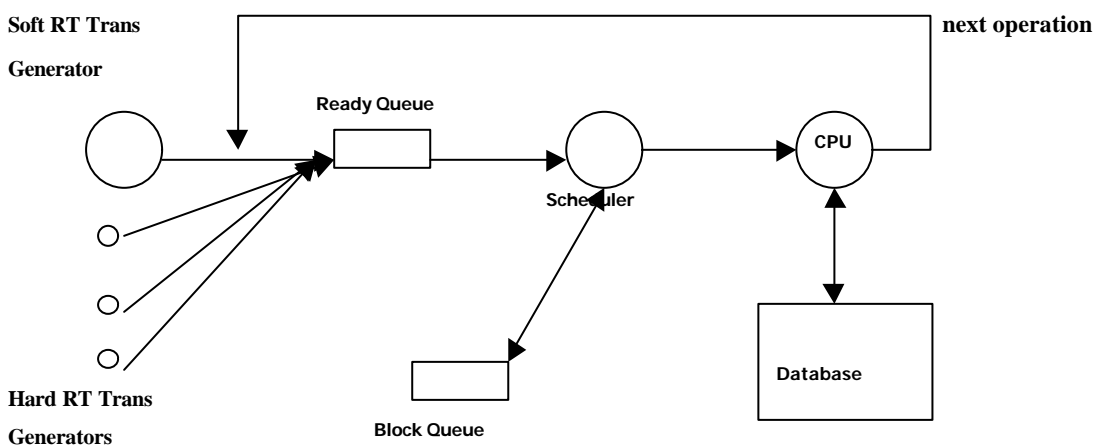


Figure 1: The RTDBS Model

There are two types of transaction generators in the system. One of the generators is responsible for the generation of soft real-time transactions following the Poisson distribution. The other generators generate hard real-time transactions following their own periods.

Each generated (hard or soft) transaction is assigned a deadline and a priority. It is assumed that the deadline of a hard real-time transaction is the sum of its arrival time and period to simplify the analysis. (Note that the protocol should work well even if the hard real-time transactions have pre-period deadlines. Although the number of deadline violations for hard real-time transactions with pre-period deadlines will increase, the impact of having pre-period deadlines on the RCP should be similar to that on the PCP.) The priorities of hard real-time transactions are defined according to the rate monotonic algorithm [6]. The hard real-time transaction comes from a shorter period generator, and thus will be assigned a higher priority. For the soft real-time transactions, a higher priority will be assigned to the transaction with a closer deadline according to the Earliest Deadline First (EDF) scheduling. Hard real-time transactions have higher priorities than the soft real-time transactions except when the soft real-time transactions are in their validation or write phases.

The operations of a soft real-time transaction are either read or write. The probability of an operation to be a write operation is defined by the read/write ratio (R/W Ratio). The processing of an operation requires the access of data items in the database and the use of the CPU. It is assumed that each operation requires the access of one data item. The required data items of the operations of the soft real-time transactions are uniformly distributed in the database. The required data items of the operations of the hard real-time transactions are predefined.

The scheduling of the CPU is performed by the scheduler. The transactions are queued in the ready queue for the CPU according to their priorities. At any time, the highest priority transaction runs the CPU, unless it is being blocked by other transactions due to lock conflict. Before a transaction (both soft and hard) is allowed to use the CPU, its deadline will be checked. If it has been missed, the transaction will be aborted immediately. Since main memory database systems can better support real-time database applications, it is assumed that the database is in the main memory [4].

Before a transaction accesses a data item, it has to set the lock corresponding to the data item in an appropriate mode. Based on the adopted concurrency control protocol, the scheduler determines whether the lock requesting transaction can be processed, be blocked in the block queue, or be aborted. If a transaction has to be aborted, all its seized locks have to be released, and it may restart its execution from beginning if its deadline has not expired. After the completion of the commit procedure, the transaction will release all its seized locks.

## 5.2 Simulation Experiments Settings

Two simulation models of the above RTDBS model have been implemented using the simulation tool OpNet [29]. One of the simulation models uses the RCP for concurrency control while the other uses a modified version of the OCC wait-50 [5,19]. We call it Mixed OCC (MOCC). The reason of comparing the RCP with a protocol based on OCC wait-50

instead of HP-2PL [4] is that the performance of the OCC wait-50 has been shown to be better than HP-2PL [12,19,21,22]. The reason of not comparing the RCP with the PCP is that the PCP cannot be implemented in a RTDBS with mixed transactions (having unknown access data sets) due to its restrictive assumptions. Note that the data sets of soft real-time transactions are unknown at their arrivals.

The components of the real-time database systems and the transaction model have been adopted in many other research studies. The simulation program has been validated under different model parameter values, different simulation lengths, different types of workloads, and even when the system merely consists of read-only transactions (for both hard real-time and soft real-time transactions). Note that this workload reflects the characteristics of a control system which rarely has any data conflicts.

In the model with the MOCC, the executions of hard real-time transactions will follow the same way as the soft real-time transactions do. They have to go through three phases: execution phase, validation phase, and write phase. Hard real-time transactions are given preferences in using the data items. Whenever a validating soft real-time transaction finds that it has a data conflict with a hard real-time transaction, the soft real-time transaction will be aborted. Since validation of a transaction has to be performed in a critical section, a hard real-time transaction may be blocked by a soft real-time transaction which is in the validation phase.

The primary performance measure is the miss rate which is defined as the number of deadline missing transactions over the total number of transactions generated. Since there are two types of transactions in the system, two miss rates are defined. They are  $MR_h$  (miss rate of hard real-time transactions) and  $MR_s$  (miss rate of soft real-time transactions). Since missing the deadline of a hard real-time transaction can be catastrophic, the value of  $MR_h$  should be zero. In addition, we also measure the total abort rate ( $AR_T$ ) and mean block queue length. The total abort rate is defined as the number of transactions aborted due to data conflicts over the total number of transactions generated.

Similar to many other previous studies [4,11,12], the deadlines of soft real-time transactions are defined based on the expected execution time of the transactions such as:

$$\text{Deadline} = \text{expected execution time} \times (1 + \text{slack number})$$

where slack number is a random variable uniformly distributed between the slack range.

Table 2 summarizes the set of model parameters and their baseline setting:

<b>Model Parameters</b>	<b>Baseline Values</b>
Number of hard real-time transaction generators	5
Mean arrival rate of soft real-time transaction (MAR)	2 trans/sec

Transaction length of soft real-time transactions	3 to 15 operations uniformly distributed
Transaction length of hard real-time transactions	5 operations
Slack range	3 to 4
CPU time to process a data item	34 ms
CPU time to perform an update in the write phase	6ms
CPU time to record the access of a data item by soft real-time transactions at the read phase	1ms
CPU time to remove the record of the access of a data item by soft real-time transactions at the validation phase	1ms
R/W Ratio	0.5
Number of Data Items in the Database	200

Table 2: Model parameters and their baseline setting

In the baseline setting, the periods of the hard real-time transaction generators are defined to be 40sec, 20sec, 13.3sec, 10sec and 8sec. These represent a total workload of the hard real-time transactions of 15%. The parameter settings, such as the soft real-time transaction lengths, are mainly defined based on the settings adopted in the previous research results, e.g. [4,11,12]. Since little work has been done in mixed scheduling of hard and soft real-time transactions, some experiment parameters are determined according to the common characteristics of many real-time database applications, e.g., the mean time to process an operation and the time required to set a lock. Some of the parameters are set to reflect the purposes of the performance studies of the paper. For example, in order to evaluate the performance of the RCP under heavy workloads and high data contention situations, a heavy hard real-time workload and a small database are adopted in the simulation experiments.

The length of each simulation run is 5,000 seconds. When the arrival rate of soft real-time transactions is one soft real-time transaction per second, about 5,000 soft real-time transactions are processed. The simulation length is determined after a series of trial runs until the obtained results are stable. For each simulation run, we use a batch mean method to collect the statistics and it is divided into 10 batches. The length of each batch equals to 500 simulation time units. The statistics from the first batch is discarded in order to remove the initial bias. The final results are obtained from the mean values of the results from the second to the last batch. For the miss rate of soft real-time transactions, we have found that the confidence levels for 95% is within  $\pm 0.1\%$  of the mean values.

## 5.3 Simulation Results

### 5.3.1 Impact of Workload

Figures 2 to 4 depict the results for MOCC and RCP, respectively, under different workloads of soft real-time transactions (different MAR) and at a hard real-time transaction workload equal to 15% utilization of the system. As shown

in Figure 2, the miss rates of both hard and soft real-time transactions are consistently much lower under RCP than that under MOCC, especially when the MAR is high. The smaller  $MR_s$  for RCP is due to smaller abort rates,  $AR_T$  as observed in Figure 3. As depicted in Figure 3,  $AR_T$  is higher under MOCC than that under RCP. It is because all the data conflicts between the hard real-time transactions and the soft real-time transactions are resolved by aborting the soft real-time transactions in MOCC. The amount of resources wasted on aborted transaction is high due to late detection of data conflicts in the optimistic approach. Furthermore, the poor performance of the soft real-time transactions in MOCC is also due to the abort of hard real-time transactions. A hard real-time transaction may be aborted due to data conflicts with other hard real-time transactions. They will restart from their beginning. Thus, the hard real-time transaction workload is higher, and the scheduling of the soft real-time transactions is greatly affected.

On the other hand, some of the data conflicts between hard and soft real-time transactions are resolved by blocking in RCP. Concurrent accesses on the same data items by a soft real-time transaction and a hard real-time transaction may be allowed if both of them want to write on the data item and the hard real-time transaction commits first. So,  $AR_T$  for RCP is smaller as shown in Figure 3. However, when the workload is very heavy (as shown in Figure 3), the abort rate of the MOCC is a little bit smaller than the RCP. It is because more transactions are aborted due to deadline missing before they access data items under the MOCC than those under the RCP.

An important observation from Figure 2 is that when the MAR is high, the  $MR_h$  for RCP remains zero. Thus all the hard real-time transactions can be guaranteed. However, the  $MR_h$  for MOCC increases with an increase in workload. This is due to the abort of hard real-time transactions. The aborted hard real-time transactions will have a higher probability to miss deadlines as they may not have sufficient slack for re-execution. About 8% of hard real-time transactions miss their deadlines when the MAR is 2.5 tran/sec. Although the percentage is not high, it is highly undesirable as any deadline missing of a hard real-time transaction can result in disasters. In RCP, data conflicts between hard real-time transactions are resolved by blocking. Although this will result in a greater number of blocked transactions, as shown in Figure 4, the performance of the hard real-time transactions will not be affected as they have sufficient time to complete their execution.

Figures 5 to 6 show the results when the workload of the hard real-time transactions is doubled (30%) by reducing the generation periods of the hard real-time transactions. The results are consistent with those in Figure 2 to Figure 3. The performance of RCP is consistently better than that of MOCC. The differences in their performance are even greater as the conflict probability between the soft and hard real-time transactions is higher due to a higher hard real-time transaction workload. Similar to Figure 2, both  $MR_s$  and  $MR_h$  increase when the MAR increases in MOCC, as depicted in Figure 5. However, the  $MR_h$  remains zero even under a high MAR in RCP. The poor performance of MOCC as compared with RCP is also due to the abort of hard real-time transactions. However, as can be seen from Figure 6, it is surprise to see that the total abort rate,  $AR_T$ , for RCP is even higher than that of MOCC under a heavy workload. It is because, under a heavy workload, many soft real-time transactions miss their deadlines and are aborted before they reach the validation phase. So,

the number of abort due to data conflict is smaller.

Figure 7 shows the miss rates for RCP and MOCC under different hard real-time transaction workloads when the soft real-time transaction workload is set to be 2.0 trans/sec. RCP2.0 S and RCP2.0 H denote the miss rates of soft and hard real-time transactions for RCP, respectively; MOCC2.0 S and MOCC2.0 H denote the miss rates of soft and hard real-time transactions for MOCC, respectively. It can be observed that, under a situation with only soft real-time transactions, i.e., the number of hard real-time transactions is set to be zero, the performance of RCP is similar to that of MOCC, as both of them use the optimistic approach to resolve the data conflicts between soft real-time transactions (Please see the miss rates when “Hard Tx Utilization” is equal to zero). However, when the hard real-time transaction workload increases, the performance of RCP becomes better than MOCC. Both the  $MR_s$  and  $MR_h$  for the MOCC increase sharply with the hard real-time transaction workload. However, the  $MR_s$  and  $MR_h$  for RCP are much smaller even under heavy hard real-time transaction workload.

### 5.3.2 Read/Write Probability

Figure 8 depicts the miss rates,  $MR_s$  and  $MR_h$ , under different Read/Write Ratios (R/W ratios) when the hard real-time transaction workload is 15% and at the MAR of 1.4 tran/sec and 1.7 tran/sec respectively. (RCP-s 1.4 denotes the miss rates of soft real-time transactions for RCP at the MAR of 1.4 tran/sec.) Increasing the proportion of read operations (larger R/W ratio) in a soft real-time transaction, the  $MR_s$  for both RCP and MOCC decrease gradually due to smaller conflict probability. The performance of RCP is consistently better than MOCC under different R/W ratios and different soft real-time transaction workloads. The differences in their performance become smaller at greater percentages of read operations due to smaller data conflict probability. For a greater write operation probability (smaller R/W ratio), more transactions will be aborted under MOCC than that under RCP because write/write conflicts between hard and soft real-time transactions may be allowed (if the hard real-time transactions can finish earlier). Similar to Figure 2, an important observation from Figure 8 is that the  $MR_h$  for RCP remains zero even under a heavier workload (MAR = 1.7) and a higher conflict probability (all operations are write). But the performance of the hard real-time transactions in MOCC is not so good. Their miss rates are consistently greater than zero for different ratios of read/write operations and soft real-time transaction workloads.

### 5.3.3 Mean Value of Slack Number

The mean value of the slack number (slack mean) determines the tightness of the deadlines of the soft real-time transactions. Figure 9 shows the miss rates,  $MR_s$  and  $MR_h$ , under different mean values of slack number. Consistent with our intuition, increasing the mean values decreases the miss rates of the soft real-time transactions due to looser deadline constraints. When the mean value is small (e.g., 2.5), the differences in the performance between MOCC and RCP are small. It is because, for a very tight deadline constraint, even with RCP, a large number of soft real-time transactions will miss their deadlines due to the tight deadline constraints. Increasing the mean values increases the differences in the performance between MOCC and RCP. When the slack factor is very large, a soft real-time transaction can complete

before its deadline even it has been aborted.

Figure 10 shows the  $AR_T$  for RCP and MOCC under different mean values of slack number. It is a surprise to see that the  $AR_T$  gradually increases with the mean value of slack number in both RCP and MOCC. It can be explained as the following: When the mean value is very small, soft real-time transactions expire their deadlines before they enter their validation phases. Thus, the number of aborts due to data conflicts will be smaller. Increasing the mean value increases the number of transactions entering the validation phase and thus results in a greater number of transaction aborts due to data conflicts. The same argument also explains why the miss rates of RCP are much smaller than that of the MOCC, but the differences in their abort rate are not so large.

The difference between the upper and lower values of the slack bound is called slack range. Figure 11 depicts the change in slack range on the miss rates for RCP and MOCC. Although the mean value is kept to be a constant, the  $MR_s$  increases gradually with an increase in slack range for both RCP and MOCC. When the slack range is larger, the conflict probability amongst the soft real-time transactions will be higher because the probability of having a soft real-time transaction with a deadline constraint tighter than the previously arrived soft real-time transaction will be greater. This will result in more CPU preemption and will increase the number of concurrently executing transactions. It will result in more data conflicts. Higher probability of data conflicts can be observed by a greater total abort rate in Figure 12 in which the total abort rates increases gradually with the slack range.

## 6 Conclusions

Although the research on the design of concurrency control protocols for real-time database systems (RTDBS) has received a lot of attention, most of the proposed protocols are for RTDBS with a single type of transactions. They are not suitable to a large class of real-time database applications, e.g., integrated computer manufacturing systems, which consist of both hard and soft real-time transactions. Different kinds of transactions have different properties, system requirements, and importance of completing them before their deadlines. The system should ensure that all hard real-time transactions can be completed before their deadlines, and the deadline missing of soft real-time transactions can be minimized.

In this paper, we propose a new concurrency control protocol, called Reduced Ceiling Priority (RCP), for RTDBS with mixed transactions. Due to the different criticality levels of hard and soft real-time transactions, different strategies are adopted for resolving the data access conflicts between different types of transactions. In addition, the principle of the Thomas Write Rule is employed to further reduce the number of aborts of soft real-time transactions due to the data access conflicts with hard real-time transactions. In the RCP, write/write conflicts between the hard and soft real-time transactions will be allowed if the hard real-time transaction can commit first. In this paper, we have shown the important characteristics of the RCP. Similar to the PCP, it is deadlock free and can guarantee the performance of the hard real-time transactions. A series of simulation experiments have been performed to compare the performance of RCP with MOCC, a modification of OCC wait-50, in a RTDBS with hard and soft real-time transactions. The performance results have shown

that the RCP can guarantee the performance of the hard real-time transactions even under a higher hard real-time workload and a greater conflict probability. At the same time, the performance of the soft real-time transactions under the RCP is much better than under the MOCC protocol at different hard and soft real-time workloads, and different deadline constraints. It is because RCP results in a smaller number of transaction aborts.

In the RCP, the OCC wait-50 is employed for resolving data conflicts between the soft real-time transactions. Other directions for future work are to study the performance of the RCP for a mixed real-time database systems with multiprocessors and the real-time database systems consist of hard, soft and non-real-time transactions. To further control the probability of data conflicts among soft real-time transactions, an admission control scheme may be used. When the workload of the soft real-time transactions is higher than a certain threshold, new transactions will not be allowed to enter the system for processing. Since the purpose of the paper is to propose a protocol for resolving data access conflicts in RTDBS and is not to extend the PCP, we choose to adopt the PCP for the concurrency control of hard real-time transactions in the RCP, instead of the read/write PCP. The RCP can be easily extended with the read/write PCP, and all nice properties of the RCP will remain.

## References

- [1] Bestarovs, A. (1996) Advances in Real-time Database System Research. *ACM SIGMOD Record*, 25(1), 3-8.
- [2] Bestavros, A., Lin, K.J., and Son, S.H. (1997) *Real-time Database Systems: Issues and Applications*. Kluwer Academic Publishers, Boston.
- [3] Yu, P.S., Wu, K.L., Lin, K.J. and Son, S.H. (1994) On Real-time Databases: Concurrency Control and Scheduling. *Proceedings of IEEE*, 82(1), 140-157.
- [4] Abbott, R., and Garcia-Molina, H. (1992) Scheduling Real-time Transactions: A Performance Evaluation. *ACM Transactions on Database Systems*, 17(3), 513-560.
- [5] Kao, B. and Garcia-Molina, H., (1993) An Overview of Real-Time Database Systems. *Advances in Real-Time Computing*. Edited by W.A. Halang and A.D. Stoyenko, Springer-Verlag.
- [6] Liu, C.L., and Layland, J.W. (1976) Scheduling Algorithms for Multi-programming in a Hard-Real-Time Environment. *Journal of ACM*, 20(4), 41-64.
- [7] Bernstein, P.A., Hadzilacos, V., and Goodman, N. (1987) *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, Mass.
- [8] Stankovic, J., Spuri, M. and Natale, M.D. (1995.) Implications of Classical Scheduling Results for Real-time Systems. *IEEE Computer*, 28(6),16-25.
- [9] Sha, L., Rajkumar, R., Son S.H. and Chang, C.H. (1991) A Real-time Locking Protocol. *IEEE Transactions on Computers*, 40(7), 793-800.
- [10] Sha, L., Rajkumar, R., and Lehoczky, J.P. (1990) Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9), 1175-1186.
- [11] Haritsa, J.R., Carey M.J. and Livny, M. (1992) Data Access Scheduling in Firm Real-time Database Systems. *Journal of Real-time Systems*, 4(3), 203-242.
- [12] Lee, J. (1994) *Concurrency Control Algorithms for Real-time Database Systems*. Ph.D. Thesis, Department of Computer Science, University of Virginia.
- [13] Son, S. H., Lin Y. and Cook, R.P. (1991) 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 Publishers.
- [14] Kim, Y-K and Son, S.H., (1996) Supporting Predictability in Real-Time Database Systems. In *Proceedings of Real-Time Technology and Applications Symposium*, Brookline, Massachusetts, pp. 38-48.
- [15] Peng, C.S., Lin K.J. and Ng, Tony P. (1997) A Performance Study of the Semantic-Based Concurrency Control Protocol in Air Traffic Control Systems. *Real-Time Database and Information Systems*. Edited by A. Bestavros and V. Fay-Wolfe, Kluwer Academic Publishers, Boston.

- [16] Shimakawa, H., Ido, G., Takada, H., Asano, Y., and Takegaki, M. (1997) Active Transactions Integrated with Real-Time Transactions According to Data Freshness. In *Proceedings of 1997 IEEE Real-Time Technology and Applications Symposium*, pp. 49-59.
- [17] Kuo, T.-W. and Mok, A.K. (1993) SSP: a Semantics-Based Protocol for Real-Time Data Access. In *Proceedings of IEEE 14th Real-Time Systems Symposium*, pp. 76-86.
- [18] Konana, P. and Ram, S. (1999) Semantic-Based Transaction Processing for Real-Time Databases: A Case of Automated Stock Trading. *INFORMS Journal on Computing*, 11(3), 299-315.
- [19] Lin, Y. and Son, S. H. (1990) Concurrency Control in Real-Time Databases by Dynamic Adjustment of Serialization Order. In *Proceedings of IEEE 11th Real-Time Systems Symposium*, pp. 104-112.
- [20] Liang, M.C., Kuo, T.W., and Shu, L. (1996) BAP: A Class of Abort-Oriented Protocols Based on the Notion of Compatibility. In *Proceeding of the Third International Workshop on Real-Time Computing Systems and Applications*, pp. 118-127.
- [21] Haritsa, J.R., Carey, M.J., and Livny, M. (1990) On Being Optimistic about Real-Time Constraints. In *Proceedings of the 9<sup>th</sup> ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 331-343.
- [22] Chiu, A., Kao B., and Lam, K. (1998) An Analysis of Lock-Based and Optimistic Concurrency Control Protocols in Multiprocessor Databases. *Journal of Systems and Software*, 42(3), 273-286.
- [23] Pang, H., Carey, M.J., and Livny, M. (1995) Multiclass Query Scheduling in Real-Time Database Systems. *IEEE Transactions on Knowledge and Data Engineering*, 7(4), 533-551.
- [24] Thomas, S. Seshadri, S. and Haritsa, J.R. (1996) Integrating Standard Transactions in Real-Time Database Systems”, *Information Systems*, 21(1), 3-28
- [25] Strosnider, J.K., Lehoczky, J.P. and Sha, L. (1995) The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-time Environment. *IEEE Transactions on Computers*, 44, 1405-1419.
- [26] Thuel, S.R. and Lehoczky, J.P. (1994) Algorithms for Scheduling Hard Aperiodic Tasks in Fixed-Priority Systems Using Slack Stealing. In *Proceedings of IEEE 15th Real-Time Systems Symposium*, pp. 22-35.
- [27] Tia, T., Liu, Jane W.S., and Shankar, M. (1996) Algorithms and Optimality of Scheduling Soft Aperiodic Requests in Fixed-Priority Preemptive Systems. *Journal of Real-time Systems*, 10, 23-43.
- [28] Joseph, M and Pandya, P. (1986) Finding Response Times in a Real-Time Systems. *The Computer Journal*, 29(5), 390-395.
- [29] *OpNet Modeling Manual*. (1996) Release 2.5 MIL 3, Inc., Washington, DC.

\*The work reported in this paper was supported in part by a research grant from the National Science Council, Taiwan, under Grant NSC87-2213-E-194-002.

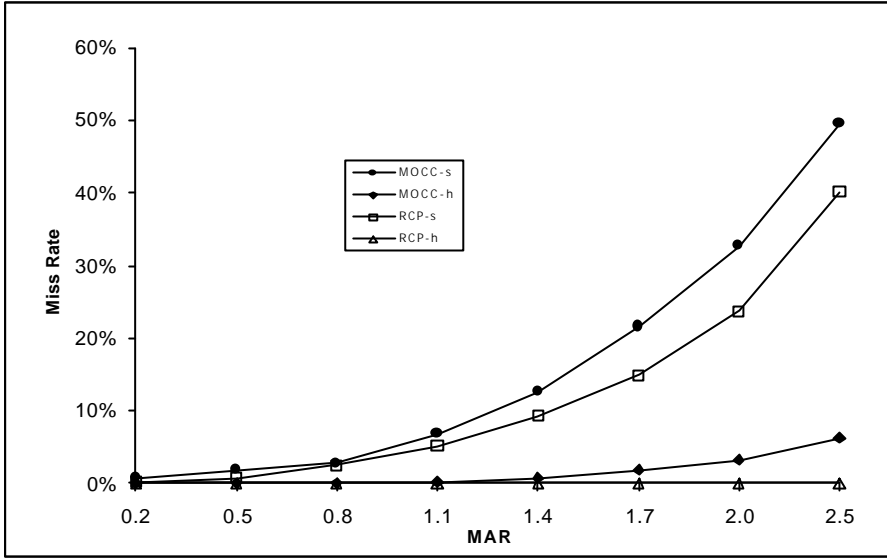


Figure 2: Miss rate of transactions when hard real-time transaction workload is 15 %

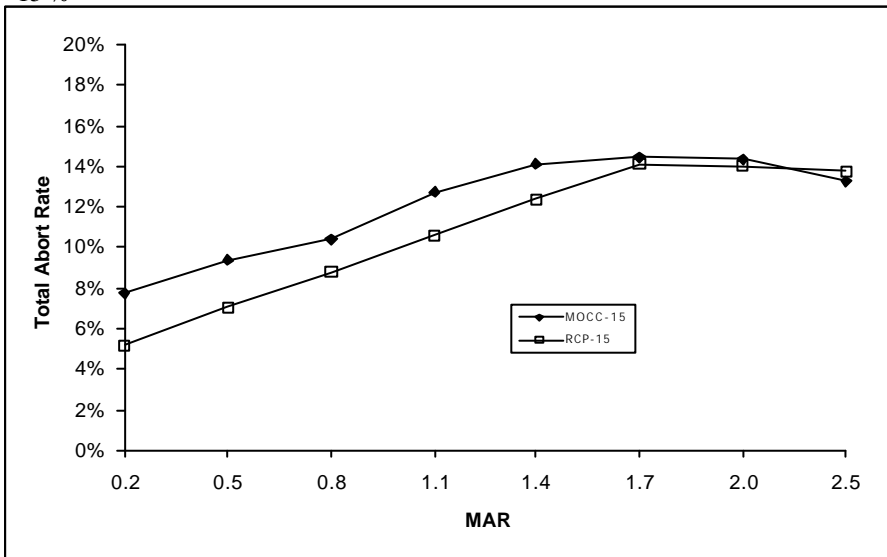


Figure 3: Total abort rate when hard real-time transaction is 15%

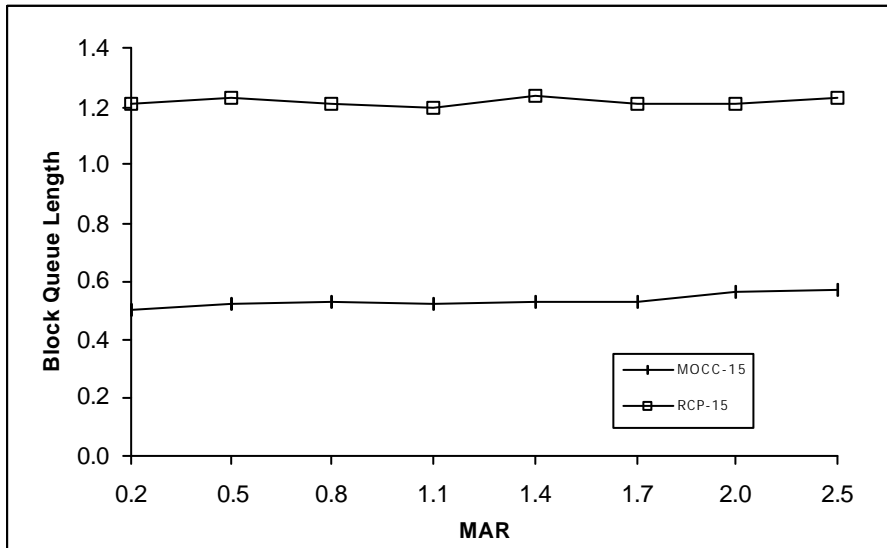


Figure 4: Mean block queue length when hard real-time transaction workload is 15%

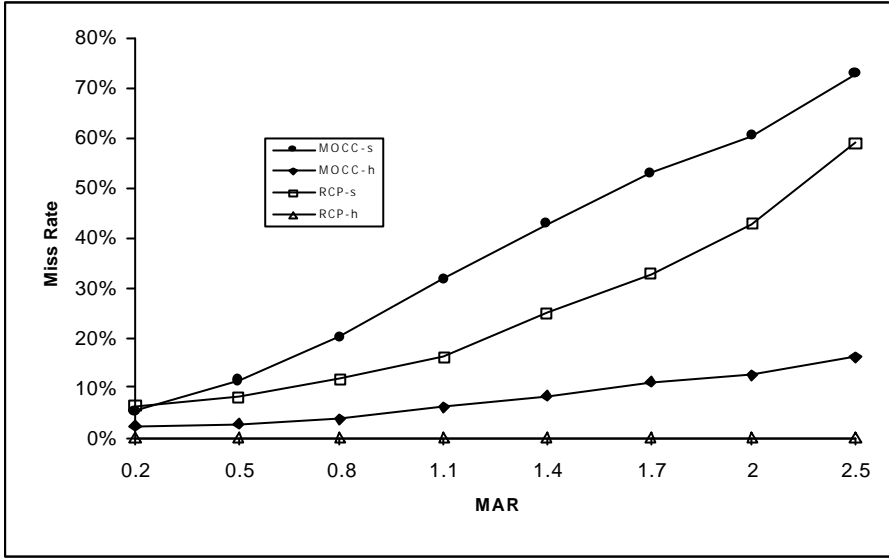


Figure 5: Miss rate of transactions when hard real-time transaction workload is 30%

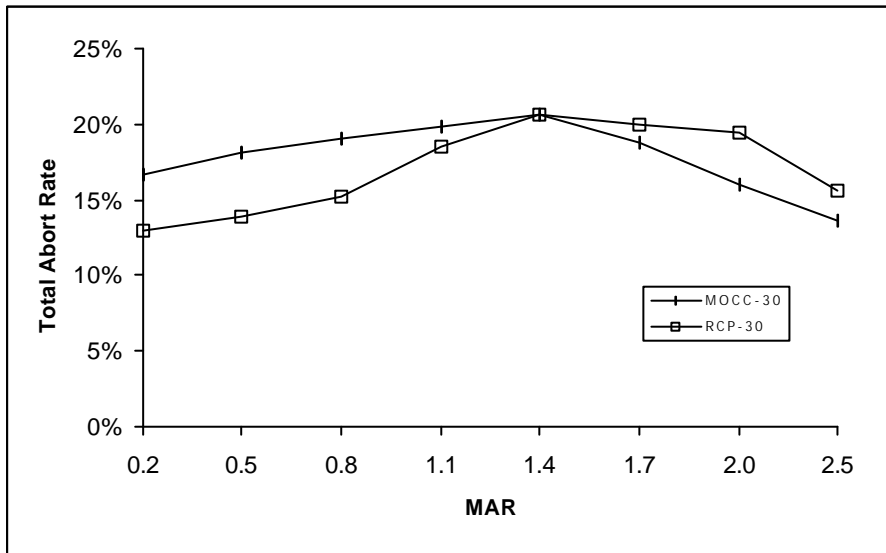


Figure 6: Total abort rate when hard real-time transaction workload is 30%

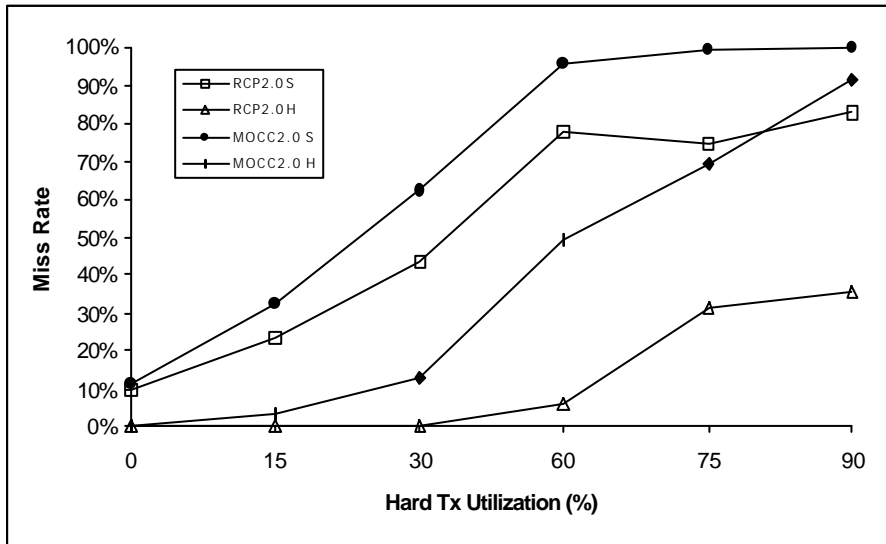


Figure 7: Miss rate of transaction at different hard real-time transaction workloads

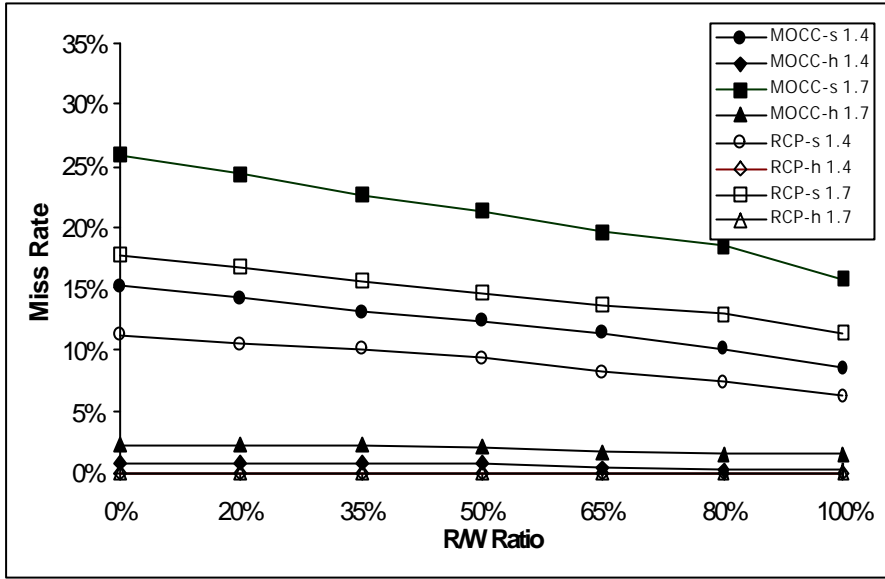


Figure 8: Miss rate of transaction at different read/write ratios

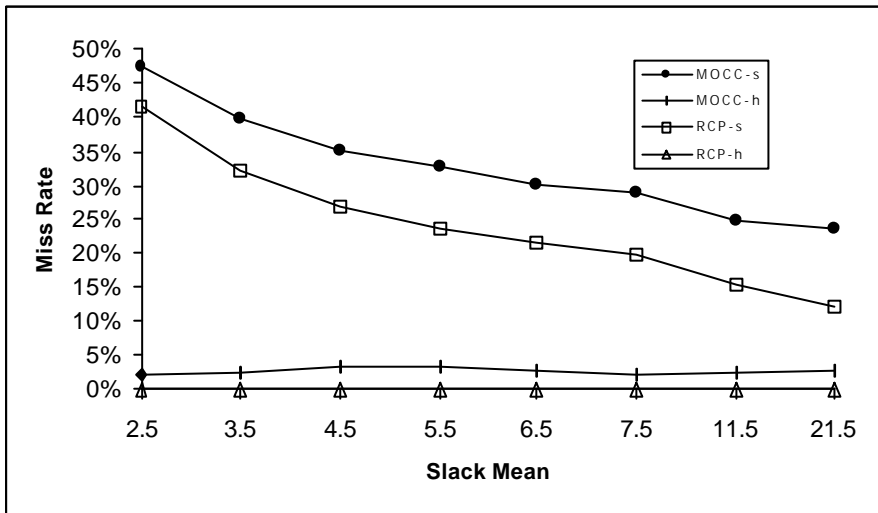


Figure 9: Miss rate of transactions in different mean value of slack factor

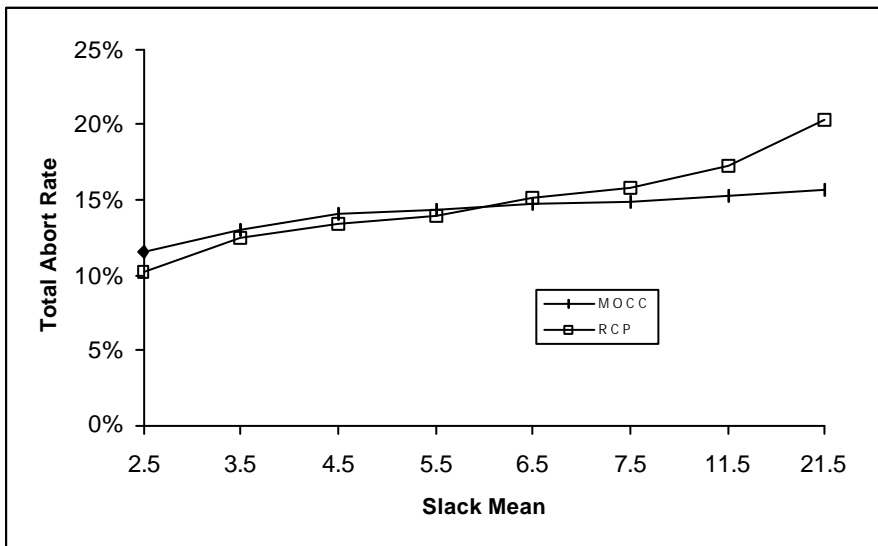


Figure 10: Total abort rate of transaction in different mean value of slack factor

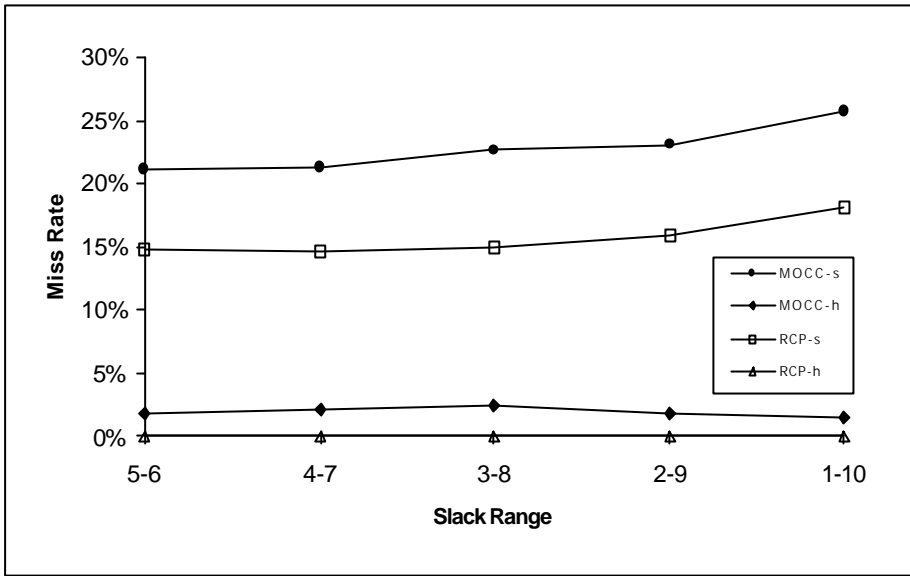


Figure 11: Miss rate of transactions in different slack ranges

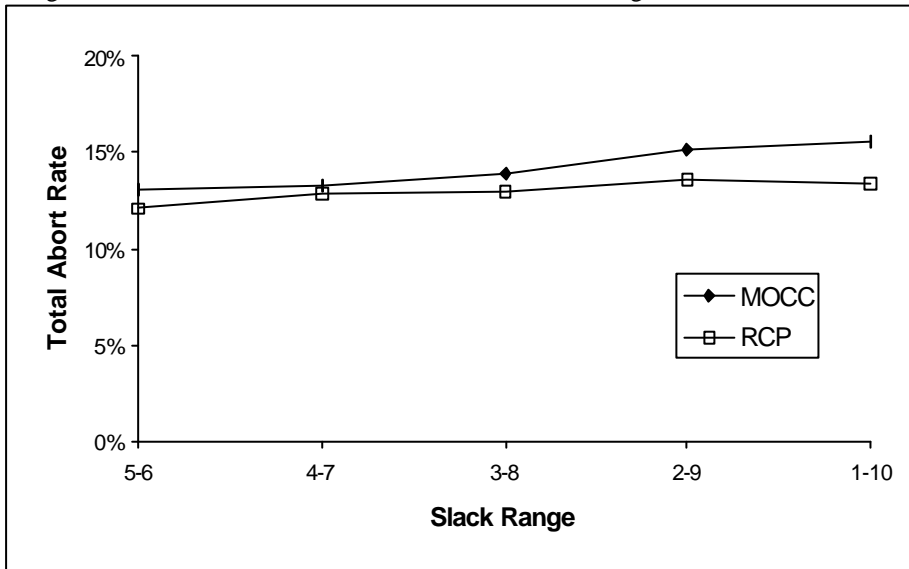


Figure 12: Total abort rate of transactions in different slack ranges