Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

On Improving the Schedulability of Real-Time Data-Intensive Applications: Preemptibility versus Non-Preemptibility*

Ming-Chung Liang, Shao-Juen Ho, and Tei-Wei Kuo

Advanced System Integration Laboratory
Department of Computer Science and Information Engineering
National Chung Cheng University, Chiayi, Taiwan 621, R.O.C.
{lmc83,hsj84,ktw}@cs.ccu.edu.tw

Abstract

This paper proposes a class of simple and efficient concurrency control algorithms based on the notion of compatibility. We consider a schedulability analysis model to better manage the schedulability of a transaction system. The schedulability of a transaction system is improved by aborting excessive blocking from lower priority transactions and, at the same time, blocking excessive aborting from highly priority transactions. We consider the criticality of transactions and provide a uniform mechanism to tune up the tolerable blocking time of transactions. The strengths of this work is demonstrated by improving the worst-case schedulability of an avionics example [16] and a satellite control system [5].

KEY WORDS AND PHRASES: real-time databases, concurrency control, transaction aborting, priority inversion.

1 Introduction

Real-time database systems must preserve data consistency and meet the timing constraints of transactions. A number of analytic and simulation studies on the performance of scheduling algorithms that meet the specified deadline requirements have been documented in the literature [1, 2, 3, 6, 7, 8, 9, 11, 15, 18, 19, 22, 23, 25]. In these studies, some proposed new real-time concurrency control algorithms; some adopted real-time scheduling algorithms such as the Earliest-Deadline-First algorithm (EDF) [10] to meet the timing requirements of transactions; and some adopted conventional concurrency control algorithms such as the two-phase locking protocol (2PL) to coordinate the data accesses of transactions.

The technical question is to determine which real-time scheduling algorithm can be used with which conventional concurrency control algorithm without adverse results. This paper proposes a class of concurrency control algorithms based on the notion of compatibility. We consider a schedulability analysis model to better manage the schedulability of a whole transaction system. The schedulability of higher priority transactions is improved by aborting lower priority transactions that may introduce excessive blocking to higher priority transactions. The schedulability of lower priority transactions is retained by controlling their aborting cost from higher priority transactions. Lower priority transactions are allowed to increase the length of their critical sections or use an implicit locking scheme based on the idea of preemption level [2] to reduce any interference from higher priority transactions. We also provide a uniform mechanism to consider the criticality of a transaction and to tune up the tolerable blocking time of transactions. The strengths of the work is demonstrated by improving the worst-case schedulability of an avionics example [16] and a satellite control system [5].

The rest of the paper is organized as follows. Section 2 introduces the modulized scheduling approach (MSA) and related real-time concurrency control algorithms. We also prove some important properties such as serializability preservation and maximum priority inversion time. Section 3 derives theorems and provides an analytic procedure to balance aborting cost and blocking cost. Some case studies are also provided. Section 4 further extends the MSA approach and proposes a uniform mechanism to consider the criticality of transactions. Section 5 is the conclusion.

2 Real-Time Data Synchronization: Modulized Scheduling Approach (MSA)

2.1 Compatibility Model

The basic idea of the modulized scheduling approach (MSA) is to integrate real-time scheduling algorithms, conventional concurrency control algorithms, and simple aborting algorithms based on the notion of compatibility. A real-time scheduling algorithm is *compatible with* a concurrency control algorithm if the real-time scheduling algorithm allows transactions to access data objects according to the synchronization requirements of the concurrency control algorithm. As an example, a real-time scheduling algorithm such as the Earliest-Deadline-First algorithm, that schedules transactions/processes with

*Supported in part by a research grant from the National Science Council under Grant NSC85-2213-E-194-008

the most urgent deadline, is compatible with a concurrency control algorithm like 2PL, which forbids any transaction to obtain any new locks on data objects if the transaction has released any locks. A real-time scheduling algorithm that forbids transactions/processes to hold more than one resource or data object at a time is apparently incompatible with 2PL.

As the astute reader can expect, most well-known real-time scheduling algorithms such as EDF [10] and conventional concurrency control algorithms such as 2PL are compatible with each other. The main difficulty lies on the compatibility problem of an aborting algorithm with a real-time scheduling algorithm and a concurrency control algorithm. It is because an aborting algorithm may change the priority structure of transactions such as that adopting priority inheritance [24] or violate the synchronization requirements imposed by a concurrency control algorithm such as serializability. An aborting algorithm must specify these concerns if it is to be used with or to be compatible with a real-time scheduling algorithm and/or a concurrency control algorithm. As an example, if an aborting algorithm does not reset the priority of a transaction that is inherited from the priority of an aborted transaction, then the aborting algorithm is not compatible with the Priority Ceiling Protocol (PCP) [24] where PCP depends heavily on the idea of priority inheritance. Furthermore, an aborting algorithm that aborts any transactions involved in certain resource competition must release all resources or data objects held by the transactions to be compatible with 2PL. If an aborting algorithm is, indeed, compatible with a real-time scheduling algorithm and/or a concurrency control algorithm, then the combination of these might preserve the properties of the two algorithms. The minimum requirements for an aborting algorithm to be compatible with a real-time scheduling algorithm and a concurrency control algorithm are to ensure that the aborting algorithm does not harm the priority assignment scheme, scheduling mechanism, and/or data synchronization requirements of the real-time scheduling algorithm and the concurrency control algorithm. We refer interested readers to [11] for the detailed definition of "compatibility".

For the purpose of this paper, we will use the terms "resource" and "data object" interchangeably. (A data object is one kind of resource.) We will also use the terms "process" and "transaction" synonymously.

Now, we will state our notation.

Notation:

- $\tau_{i,j}$ denotes the j_{th} instance of transaction τ_i. p_i and c_i are the period and worst-case computation time of transaction τ_i, respectively. If transaction τ_i is aperiodic, p_i is the minimal separation time between its consecutive requests. When there is no ambiguity, we use the terms "transaction and "transaction instance" interchangeably.

- $R_{i,j}$ denotes the j_{th} request of transaction τ_i. A transaction instance $\tau_{i,j}$ is initiated for each request of transaction τ_i. Once transaction instance $\tau_{i,j}$ is aborted, $\tau_{i,j}$ may be restarted or terminated as required by the selected scheduling algorithm.

2.2 Modulized Scheduling Approach

We assume that a transaction system consists of a fixed set of transactions. Transactions are classified as abortable or non-abortable in an off-line fashion. (Please see Section 3.) Before a transaction can access a data object, the transaction must first obtain a lock on the semaphore that guards the data object. When a transaction terminates (commits or is aborted), it must release all of its locks. A critical section of a transaction is a code segament between the locking operation and its corresponding unlocking operation of the corresponding semaphore. We assume in this paper that critical sections are properly nested[1].

Transactions are required to adopt a delayed write procedure. For each data object updated by a transaction, the update is done in the local area of the transaction and the actual write of the data object is delayed until the commit time of the transaction. As a result, transactions do not release locks of semaphores until they commit or are aborted. The delay write procedure eases the aborting process and avoids cascading aborting.

Protocol	Bounded Priority Inversion	Blocked at Most Once	Deadlock Avoidance
NPCS	Yes	Yes	Yes
HLP	Yes	Yes	Yes
PCP	Yes	Yes	Yes

Table 1: Real-Time Scheduling Algorithms: Non-Preemptible Critical Section (NPCS), Highest Locker's Priority (HLP), and Priority Ceiling Protocol (PCP).

Our scheduling algorithms are the integration of three compatible components: real-time scheduling algorithms in Table 1[2], the two-phase locking protocol (2PL), and a simple aborting mechanism. We are interested in the context of uniprocessor priority-driven preemptive scheduling, and every transaction has a fixed priority.

Real-Time Scheduling Algorithms:

The non-preemptible critical section algorithm (NPCS) is an extension of the kernelized monitor algorithm in [17]. The NPCS algorithm, is a priority-driven scheduling algorithm, where the critical sections of every transaction instance are non-preemptible. The highest locker's priority algorithm (HLP) documented in [20] is a refinement of the NPCS algorithm. Instead of making all critical sections non-preemptible, when a transaction instance is executing within a critical section, the transaction instance runs at a priority equal to the highest priority of all transaction instances which may access the corresponding

[1]It is one of the assumptions of PCP to handle the priority inversion problem.

[2]A similar table appears in [14, 20]

semaphore. The *priority ceiling protocol* (PCP) in [24] further refines the HLP algorithm, 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 no larger than the priority ceiling of any resource which has been grabbed by another process but has not yet been released. The priority ceiling of a semaphore is equal to the highest priority of transactions which may use the semaphore. Note that PCP can only be used with fixed priority assignments, and the complexity of PCP is the highest in Table 1. However, it allows the highest level of concurrency in a transaction system.

Concurrency Control: 2PL

Transactions are required to access semaphores in a 2PL fashion. For example, if NPCS is used to schedule real-time transactions, transactions tend to have a single non-preemptible critical section, in which semaphores are accessed in a 2PL fashion.

Aborting Mechanism:

Transactions are classified as abortable or non-abortable in an off-line fashion (Please see Section 3). When transaction instance τ requests a semaphore lock, and τ is blocked (under the chosen NPCS, HLP, or PCP algorithm) by some lower priority transaction instances, τ aborts all of the blocking lower priority transaction instances if the lower priority transaction instances are all abortable. If any one of the blocking lower priority transactions is non-abortable, τ is blocked and does not abort any transaction instance. Note that when a transaction instance is aborted, it releases every lock it owns and restarts immediately.

A scheduling algorithm consists of any real-time scheduling algorithm in Table 1, 2PL, and the above aborting mechanism is called a MSA algorithm. The set of MSA algorithms can be properly enlarged by considering other compatible real-time scheduling algorithms and concurrency control mechanisms. For example, the Basic Aborting Protocol, Table-Driven Aborting Protocol, and Dynamic Aborting Protocol are also a MSA algorithm or its variations [12, 13].

Example 1 *A MSA Schedule: PCP + 2PL + Aborting*

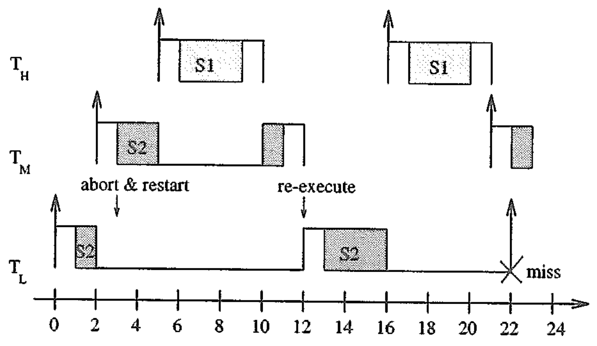

Figure 1: A MSA Schedule

We illustrate a MSA schedule by an example. Suppose there are three transactions τ_H, τ_M, and τ_L in a single processor environment. Let τ_H, τ_M, and τ_L have computation requirements 5, 5, and 7, respectively, and periods 11, 19, and 22, respectively. Suppose τ_H is the only transaction using semaphore S_1, and transactions τ_M and τ_L share semaphore S_2. Let τ_H, τ_M, and τ_L have priorities 3, 2, and 1, respectively (3 is the highest priority level). Suppose transaction τ_L is abortable, and transactions τ_H and τ_M are non-abortable. According to the MSA algorithm, semaphores S_1 and S_2 have priority ceilings as high as the priority levels of transactions τ_H and τ_M, respectively.

As shown in Figure 1, transaction τ_L locks semaphore S_2 and runs at its assigned priority level at time 1. When τ_M arrives at time 2, τ_M preempts τ_L. At time 3, τ_M tries to lock semaphore S_2 and the lock request results in the aborting and restarting of transaction τ_L because τ_L holds a semaphore (S_2) with a priority ceiling no less than the priority of τ_M, and τ_L is abortable. Since the lock request of semaphore S_2 is granted, τ_M proceeds with its execution, but it is later preempted by transaction τ_H at time 5. At time 6, the lock request of semaphore S_1 issued by τ_H is granted because the priority of τ_H is higher than the pririty ceiling of semaphore S_2, that is owned by transaction τ_M at that time. At time 10, τ_H commits and releases its lock on semaphore S_1, and τ_M resumes its execution. At time 12, τ_M commits, and τ_L regains CPU and re-executes as a new transaction. However, transactions τ_H and τ_M arrive again at times 16 and 21, respectively; τ_L receives only 4 units of computation time before its deadline at time 22., but τ_L needs 7 units of computation time. τ_L misses its deadline at time 22!

This example demonstrates one of the goals in designing MSA that a higher priority transaction may abort lower priority transactions to meet its timing constraints, and the schedulability improvement of a higher priority transaction is at the cost of the possible schedulability degradation of lower priority transactions. □

2.3 Properties

We shall prove the following theorems to illustrate the idea of compatibility (See Section 2). Some definitions and proofs of PCP from [24] are listed as needed. Note that jobs in the definitions and theorems from [24] are transaction instances in this paper. When NPCS, HLP, or PCP is used as a real-time scheduling algorithm in a MSA algorithm, the MSA algorithm is named as MSA(NPCS), MSA(HLP), or MSA(PCP), respectively[3]. With limited space, detailed proofs are removed in this section.

Lemma 1 *MSA prevents deadlocks.*

Proof. The correctness of this proof directly follows from the fact that NPCS, HLP, and PCP all prevent deadlocks. □

Theorem 1 *Schedules generated by MSA are logically correct (based on serializability).*

[3]MSA(PCP) is also named as the Basic Aborting Protocol in [13].

Proof. Directly follows from the properties of the delayed write procedure and 2PL.

Definition 1 *[24] Transitive blocking is said to occur if a transaction instance is blocked by another transaction instance which, in turn, is blocked by the other transaction instance.*

Theorem 2 *MSA prevents transitive blocking.*

Proof. The correctness of this proof directly follows from the fact that NPCS, HLP, and PCP all prevent transitive blocking. □

Theorem 3 *A transaction instance can be blocked for no more than one critical section of at most one lower priority transaction instance under MSA. In other words, a transaction instance can experience at most one time of priority inversion under MSA.*

Proof. Directly follows from the properties of NPCS, HLP, and PCP. □

Theorem 4 *A higher priority transaction instance can abort at most one lower priority transaction instance under MSA(PCP).*

Proof. This theorem can be proved by arguing the semaphore locking scheme and the definition of priority ceiling. □

3 Schedulability Analysis

3.1 Schedulability Model

Lehoczky, et al. [14] and the Rate Monotonic Theory [20] first provided a pseudo polynomial time algorithm to predict the schedulability of a transaction τ_i precisely. Let b_i, ab_i, d_i, p_i, and c_i be the worst case blocking time, aborting cost (time), deadline, period, and worst case computation requirement of a transaction τ_i, respectively. We can extend their results as follows:

Let the priorities of transactions be defined in a rate monotonic fashion (RMS), where RMS assigns transactions with priorities inversely proportional to the periods of the transactions. For transactions with priorities out of RMS order, their execution times are modeled as blocking times. We refer interested readers to [20]

Lemma 2 *Transaction τ_i in a set of periodic transactions scheduled by a MSA algorithm will always meet its deadline for all phases if there exists a pair $(k, m) \in R_i$ such that*

$$\sum_{j=1}^{i-1}(c_j\lceil\frac{mp_k}{p_j}\rceil) + c_i + b_i + ab_i \le mp_k$$

where $R_i = \{(k,m)|1 \le k \le i, m = 1, 2, \cdots, \lfloor\frac{p_i}{p_k}\rfloor\}$, and b_i and ab_i are the blocking time and aborting cost of τ_i, respectively.

Proof. The worst case aborting cost of τ_i is modeled as an extra computation time of τ_i. The correctness of the proof follows from lemmas in [20]. □

3.2 Preemptibility: Aborting Cost

We use the following theorems to estimate the aborting cost of transactions.

Definition 2 *The direct aborting cost charged to a lower priority transaction instance τ by a higher priority transaction instance τ' is the CPU time that has been consumed by τ when τ is aborted by τ'.*

Definition 3 *A-$cost_{i,j}$ denotes the maximum direct aborting cost possibly charged to an instance of transaction τ_j by an instance of transaction τ_i.*

Definition 4 *Given a set of transactions $\tau_1, \tau_2, \cdots, \tau_n$ listed in the non-decreasing order of their priorities, the maximum aborting cost charged to an instance of transaction τ_j by an instance of transaction τ_i is α-$cost_{i,j}$ = $\max(A\text{-}cost_{i,k})$, where $i < k \le j$.*

Theorem 5 *A request of a lower priority transaction can be aborted at most once by a higher priority transaction within a period of the higher priority transaction.*

Proof. This theorem can be proved by arguing the semaphore locking scheme and the definition of priority ceiling. □

Let $\Pi = \{\tau_1, \tau_2, \cdots, \tau_n\}$ be a set of periodic transactions listed in the non-increasing order of their priorities, and $HPC_i = \{\tau_1, \tau_2, \cdots, \tau_{i-1}\}$ is the set of transactions with a priority no less than that of τ_i.

Lemma 3 *The worst-case aborting cost charged to a request of a lower priority transaction τ_j by a higher priority transaction τ_i is $\lceil\frac{p_j}{p_i}\rceil \times \alpha$-$cost_{i,j}$ = $\lceil\frac{p_j}{p_i}\rceil \times \max_{j \ge k > i}(A\text{-}cost_{i,k})$.*

Proof. Follows directly from the definition of α-$cost_{i,j}$ and Theorem 5. □

Lemma 4 *The worst-case aborting cost for a request of transaction τ_j is $\sum_{\tau_i \in HPC_j}(\lceil\frac{p_j}{p_i}\rceil \times \alpha\text{-}cost_{i,j})$.*

Proof. Follows directly from Lemma 3 □

Lemma 5 *The worst-case aborting cost for a request of transaction τ_j between time 0 and time $t \le p_j$ is at most $\sum_{\tau_i \in HPC_j}(\lceil\frac{t}{p_i}\rceil \times \alpha\text{-}cost_{i,j})$.*

Proof. The statement can be proved in the same way as Lemma 4 does. □

3.3 Non-Preemptibility: Blocking Cost

Theorem 6 *A request of a transaction can be blocked for at most one critical section of at most one lower priority transaction instance under MSA. In other words, a transaction request can experience at most one time of priority inversion under MSA.*

Proof. The correctness of this proved can be derived from Theorem 3. □

3.4 Schedulability Analysis Procedure

The underlying idea of the analysis is that when a lower priority transaction may introduce excessive blocking to a higher priority transaction such that the higher priority transaction may miss its deadline in the worst case, the lower priority transaction is abortable.

Let b_i, ab_i, d_i, p_i, and c_i be the worst case blocking time, aborting cost, deadline, period, and worst case computation requirement of a transaction τ_i, respectively. Also let $\Pi = \{\tau_1, \tau_2, \cdots, \tau_n\}$ be a set of periodic transactions listed in the non-decreasing order of their priorities. $HPC_i = \{\tau_1, \tau_2, \cdots, \tau_{i-1}\}$ is a set of transactions with a priority no less than that of τ_i.

Schedulability Analysis Procedure:

Let $R_i = \{(k,m) | 1 \leq k \leq i, m = 1, 2, \cdots, \lfloor \frac{p_i}{p_k} \rfloor\}$, and $SP_i = \{mp_k | (k,m) \in R_i\}$.

Lemma 2 shows that the maximum blocking time that transaction τ_i can tolerate is $MB_i = max_{t \in SP_i}[t - \sum_{j \in HPC_i}(c_j \lceil \frac{mp_k}{p_j} \rceil) - c_i - ab_i]$. Based on Lemma 5, the maximum tolerable blocking time of transaction τ_i is $MB_i = max_{t \in SP_i}[t - \sum_{j \in HPC_i}(c_j \lceil \frac{mp_k}{p_j} \rceil) - c_i - \sum_{j \in HPC_i}(\lceil \frac{t}{P_i} \rceil \alpha\text{-}cost_{j,i}]$.

Initially, all transactions are non-abortable. The maximum tolerable blocking time MB_i of transaction τ_i is calculated as defined.

1. i = 1
2. If i > n then stop
3. If a transaction τ_j $(j > i)$ has a critical section that may lock a semaphore with a priority ceiling no less than transaction τ_i, and the length of the critical section is larger than MB_i, transaction τ_j becomes abortable. (See Theorem 6.)
4. go to 2

3.5 Case Study

Two transaction systems based on an avionics example [16] and a satellite control system [5] are studied to demonstrate the strengths of the work by improving the worst-case schedulability of the transaction systems. MSA(PCP) is used in the analysis.

3.5.1 Generic Avionics Platform

The avionics platform example has 18 periodic transactions and 9 data objects [16]. Table 2 shows that MSA(PCP) can successfully schedule the first six most critical transactions. However, PCP with semaphores locked in a 2PL fashion fails to schedule transaction Weapon_Release because of its stringent 5ms jitter requirement and excessive blocking from lower priority transactions that may access data object "DB" such as transaction Display_Graphic. Note that the cumulative processor utilization of transactions with priorities no less than that of transaction Weapon_Release is 6.6%; only the most critical transaction, i.e., Timer_Interrupt, is schedulable.

3.5.2 Olympus AOCS

The Olympus AOCS [5] has 10 periodic transactions and 4 sporadic transactions. Transactions are assigned priorities according to the deadline monotonic priority assignment that assigns transactions priorities inversely proportional to their deadlines. Table 3 shows that, by making transactions Process_IRES_data and Control_Law abortable, MSA(PCP) can schedule the first eight most critical transactions. However, all of the transactions except Bus_Interrupt remain unschedulable by the pure locking PCP protocol due to excessive blocking from transactions Process_IRES_data and Control_Law.

4 Extensions

Under the modulized scheduling approach (MSA), a lower priority transaction is abortable if it potentially introduces excessive blocking to some higher priority transactions. As a result, lower priority transactions tend to have a higher chance to miss their deadlines. In this section, we further extend the MSA approach in two ways: (1) The critical sections of a transactions may be lengthened to reduce the preemption cost from higher priority transactions. (2) Each transaction is assigned a fixed preemption level to reflect its criticality or to increase its maximum tolerable blocking time.

4.1 Increased Non-Preemptibility: Enlarged Critical Section

A critical section of a transaction can be lengthened by moving the corresponding semaphore locking operation toward the beginning of the transaction or moving the corresponding semaphore unlocking operation toward to the commit operation of the transaction. We argue that moving a semaphore unlocking operation toward to the commit operation of a transaction is more effective in improving the schedulability of a transaction because whether a transaction misses its deadline is determined by the time when the transaction executes its last instruction, i.e., the commit operation. Note that lengthening critical sections only increases the blocking time of higher priority transactions. Theorems in Sections 2 and 3 remain valid.

As the astute readers may notice, the schedulability of a transaction can be best improved if the transaction only unlock semaphores at its commit time. If critical sections are properly nested (as requested by PCP), and we take this approach to the extreme, then a transaction consists of two code segments; one is a critical section with all of the semaphores needed by the transaction locked in the beginning of the critical section and unlocked at the ending of the critical section. When a transaction begins its execution, it first runs a non-critical-section code segment and, then, a critical section code segment. If a critical section (or the critical section of a transaction) is non-abortable, we can revise Lemma 2 and have a more "optimistic" theorem to justify the schedulability of a transaction.

Lemma 6 *Let transaction τ_i consist of a non-critical-section code segment and a critical-section code segment. Suppose that the non-critical-section and critical-section code segments need c_i^1 and c_i^2*

units of computation time, respectively, and the non-critical-section executes first. Transaction τ_i can be scheduled by a MSA algorithm for all phases if there exists a pair $(k, m) \in R_i$ such that

$$\sum_{j \in HPC_i} (c_j \lceil \frac{mp_k}{p_j} \rceil) + 1 + c_i^1 + b_i \leq mp_k,$$

and

$$\sum_{j \in HPC_i} (c_j \lceil \frac{mp_k}{p_j} \rceil) + c_i + b_i \leq d_i$$

Proof. The first condition makes sure that the critical-section code segment of transaction τ_i is scheduled before the scheduling point (k, m), and the second condition ensures that transaction τ_i receives enough computation time before its deadline. The rest of the proof follows directly from the correctness of Lemma 2. □

4.2 Preemption Level: Criticality and Schedulability Refinement

Real-time scheduling algorithms often assign transactions priorities based on some criteria irrelevant to the criticality/importance of the transactions. As a result, critical but lower priority transactions may miss deadline and cause the failure of the whole system. Suppose that each transaction is given a preemption level proportional to its criticality. Transaction scheduling in a MSA algorithm can be further restricted in the following way.

A transaction τ_i can preempt the running transaction if it also satisfies the following two conditions:

1. The premption level of τ_i is larger than the system premption level, where the system preemption level is the highest preemption level of transactions which were scheduled and have not terminated.

2. If MSA(PCP) is considered, the premption level of τ_i is larger than the system priority ceiling (as defined in [24])[4].

These conditions help critical but lower priority transactions to retain CPU and semaphores without sacrificing the integrity of the original MSA algorithms. Theorems related to serializability, deadlock avoidance, priority inversion, and schedulability analysis remain valid. However, Lemma 2 can be further relaxed if the schedulability of a non-abortable transaction is considered.

Lemma 7 *A non-abortable transaction τ_i scheduled by a MSA algorithm for all phases if there exist two pairs (k_1, m_1) and $(k_2, m_2) \in R_i$ and $m_2 p_{k_2} \geq m_1 p_{k_1}$ such that*

$$\textstyle\sum_{j \in HPC_i} (c_j \lceil \frac{m_1 p_{k_1}}{p_j} \rceil) + 1 + b_i \leq m_1 p_{k_1},$$

and

$$\textstyle\sum_{j \in HPC_i - HLC_i} (c_j \lceil \frac{m_1 p_{k_1}}{p_j} \rceil) +$$

$$\textstyle\sum_{j \in HLC_i} (c_j \lceil \frac{m_2 p_{k_2}}{p_j} \rceil) + c_i + b_j \leq m_2 p_{k_2},$$

where HPC_i is the set of transactions which have a higher priority than that of τ_i, and HLC_i is the set of transactions which have both a higher priority and a higher preemption level than those of τ_i, respectively.

Proof. The first condition makes sure that transaction τ_i is scheduled before the scheduling point (k_1, m_1), and the second condition ensures that transaction τ_i receives enough computation time before its deadline. Note that transactions belonging to the set $HPC_i - HLC_i$ can no longer preempt transaction τ_i after τ_i is first scheduled. The rest of the proof follows directly from the correctness of Lemma 2. □

Note that the tolerable blocking time of transaction τ_i can be easily adjusted by changing its preemption level. However, the idea of preemption level does introduce another form of blocking.

5 Conclusion

This paper proposes a class of concurrency control algorithms based on the idea of compatibility. The algorithms are not only simple and intuitive but also preserve many important properties of their individual compatible components such as the maximum priority inversion number of the Priority Ceiling Protocol (PCP). We consider a schedulability analysis model to better manage the schedulability of a whole transaction system. The schedulability of a transaction system is improved by aborting excessive blocking from lower priority transactions and, at the same time, blocking excessive aborting from highly priority transactions. We also provide a uniform mechanism to tune up the tolerable blocking time of transactions and consider the criticality of transactions. The strengths of the work is demonstrated by improving the worst-case schedulability of an avionics example [16] and a satellite control system [5].

How to provide a compromise among the blocking, preemption, and aborting costs of transactions is of paramount importance to the schedulability of a transaction system. This work provides various ways to observe the schedulability of a tansaction system from these three aspects. One future research direction of this work is to investigate the application semantics of a transaction system and its implications on transaction abortings and restartings. We will also further exploit the concept of preemption level to multiple locking granularities and modes. We believe that more research in classifying and analyzing real-time concurrency control algorithms and transaction systems to derive proper benchmarks may prove to be very rewarding.

References

[1] R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," *Proceeding of the 14th VLDB Conference*, Los Angeles, CA, 1988, pp. 1-12.

[2] T.P. Baker, "A Stack-Based Resource Allocation Policy for Real Time Processes," *IEEE 11th Real-Time Systems Symposium*, December 4-7, 1990.

[4]The preemption levels of transactions must be defined carefully to satisfy this condition.

[3] A. Bestavros, "Timeliness Via Speculation for Real-Time Databases," *IEEE 15th Real-Time Systems Symposium*, 1994.

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

[5] A. Burns, A.J. Wellings, C.M. Bailey, E. Fyfe, "A Case Study in Hard Real-Time System Design and Implementation," YCS 190, Department of Computer Science, University of York, 1993.

[6] J. R. Haritsa, M. J. Carey, and M. Livny, "On Being Optimistic about Real-Time Constraints," *Preceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, April 1990, pp. 331-343.

[7] T.-W. Kuo and A.K. Mok, "SSP: a Semantics-Based Protocol for Real-Time Data Access," *IEEE 14th Real-Time Systems Symposium*, December 1993.

[8] T.-W. Kuo and A.K. Mok, "Using Data Similarity to Achieve Synchronization for Free," *IEEE 11th Workshop on Real-Time Operating Systems and Software*, May 1994.

[9] T.-W. Kuo and A.K. Mok, "Real-Time Database - Similarity Semantics and Resource Scheduling," *ACM SIGMOD Record*, March 1996.

[10] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *JACM*, Vol. 20, No. 1, January 1973, pp. 46-61.

[11] M.-C. Liang, "Real-Time Database - Abort-Oriented Concurrency Control," Master Thesis, Dept. of Computer Science and Information Engineering, National Chung Cheng University, June 1996.

[12] M.-C. Liang, S.-J. Ho, T.-W. Kuo, and L. Shu, "Abort-Oriented Concurency Control for Real-Time Data Access," *The Second Workshop on Real-Time and Media Systems,* July 1996, pp. 223-230.

[13] M.-C. Liang, T.-W. Kuo, and L. Shu, "BAP: A Class of Abort-Oriented Protocols Based on the Notion of Compatibility," *The Third International Workshop on Real-Time Computing systems and Applications,* October 1996, pp.118-127.

[14] J.P. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *IEEE 10th Real-Time Systems Symposium*, December 1989.

[15] Y. Lin and S. H. Son, "Concurrency Control in Real-Time Databases by Dynamic Adjustment of Serialization Order," *IEEE 11th Real-Time Systems Symposium*, December 4-7, 1990.

[16] C. D. Locke, D. R. Vogel, and T. J. Mesler, "Building a Predictable Avionics Platform in Ada: A Case Study", *IEEE 12th Real-Time Systems Symposium*, December 4-7, 1991.

[17] A. K. Mok, "The Design of Real-Time Programming Systems Based on Process Models," *IEEE Real-Time System Symposium*, 1984.

[18] C.-S. Peng and K.-J. Lin, "A Semantic-Based Concurrency Control Protocol for Real-Time Transactions," *IEEE 1996 Real-Time Technology and Applications Symposium*, 1996.

[19] X. Song and Jane W.-S. Liu, "Maintaining Temporal Consistency: Pessimistic vs Optimistic Concurrency Control," *IEEE Transactions on Knowledge and Data Engineering*, October 1995, pp. 787-796.

[20] L. Sha, "Distributed Real-Time System Design Using Generalized Rate Monotonic Theory," lecture note, Software Engineering Institute, CMU, 1992.

[21] B. Sprunt, L. Sha, and J.P. Lehoczky, "Scheduling Sporadic and Aperiodic Events in a Hard Real-Time Systems," *Technical Report* CMU-SEI-89-TR-11, CMU, April 1989. also appeared as "Aperiodic Task Scheduling for Hard Real-Time Systems," *The Journal of Real-Time Systems*, Vol. 1.

[22] L. Shu and M. Young, "A Mixed Locking/Abort Protocol for Hard Real-Time Systems," *IEEE 11th Workshop on Real-Time Operating Systems and Software*, May 1994, pp. 102-106.

[23] L. Shu, M. Young, and R. Rajkumar, "An Abort Ceiling Protocol for Controling Priority Inversion," *First International Workshop on Real-Time Computing Systems and Applications*, December 1994, pp. 202-206.

[24] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *Technical Report* CMU-CS-87-181, Dept. of Computer Science, CMU, November 1987. *IEEE Transactions on Computers*, Vol. 39, No. 9, September 1990.

[25] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Concurrency Control for Distributed Real-Time Databases," *ACM SIGMOD Record*, Vol. 17, No.1, March 1988, pp. 82-98.

Transaction #	Period (ms)	Exec (ms)	Abort ?	Abort Cost (ab_i) (ms)	Schedulability Test (Use best mp_k for MB_i)	Tolerable Blocking (MB_i)
Timer_Interrupt	1	0.051	No	0	0.051	0.949
Weapon_Release	200	3.01	No	0	$\lceil 5/1 \rceil \times 0.051 + 3.01$	1.735
Radar_Tracking_Filter	25	2.03	Yes	2.03	$\lceil 25/1 \rceil \times 0.051 + \lceil 25/200 \rceil \times 3.01 + \lceil 25/25 \rceil \times 2.03 + abort$	16.655
RWR_Contact_Mgmt	25	5.03	Yes	10.06	$\lceil 25/1 \rceil \times 0.051 + \lceil 25/200 \rceil \times 3.01 + \lceil 25/25 \rceil \times 2.03 + \lceil 25/25 \rceil \times 5.03 + abort$	3.595
Poll_Bus_Device	40	1	No	15.09	$\lceil 40/1 \rceil \times 0.051 + \lceil 40/200 \rceil \times 3.01 + \lceil 40/25 \rceil \times 2.03 + \lceil 40/25 \rceil \times 5.03 + \lceil 40/40 \rceil \times 1 + abort$	4.74
Weapon_Aim	50	3.02	No	15.09	$\lceil 50/1 \rceil \times 0.051 + \lceil 50/200 \rceil \times 3.01 + \lceil 50/25 \rceil \times 2.03 + \lceil 50/25 \rceil \times 5.03 + \lceil 50/40 \rceil \times 1 + \lceil 50/50 \rceil \times 3.02 + abort$	10.21

Table 2: Schedulability analysis of MSA(PCP) for the generic avionics example

Transaction	Period (ms)	Exec (ms)	Deadline (ms)	Abort ?	Aborting Cost (ab_i) (ms)	Schedulability Test (Use best mp_k for MB_i)	Tolerable Blocking (MB_i)
Bus_Interrupt	0.96	0.19	0.63	No	0	0.19	0.44
RTC	50	0.29	9	No	0	$\lceil 9/0.96 \rceil \times 0.19 + 0.29$	6.81
Read_Bus_IP	10	1.82	10	No	0	$\lceil 10/0.96 \rceil \times 0.19 + \lceil 10/50 \rceil \times 0.29 + 1.82$	5.8
Comand_Actuators	200	2.18	14	No	0	$\lceil 14/0.96 \rceil \times 0.19 + \lceil 14/50 \rceil \times 0.29 + \lceil 14/10 \rceil \times 1.82 + 2.18$	5.04
Request_DSS_Data	200	1.46	17	No	0	$\lceil 17/0.96 \rceil \times 0.19 + \lceil 17/50 \rceil \times 0.29 + \lceil 17/10 \rceil \times 1.82 + \lceil 17/200 \rceil \times 2.18 + 1.46$	6.01
Request_Wheel_Speeds	200	1.46	22	No	0	$\lceil 20/0.96 \rceil \times 0.19 + \lceil 20/50 \rceil \times 0.29 + \lceil 20/10 \rceil \times 1.82 + \lceil 20/200 \rceil \times 2.18 + \lceil 20/200 \rceil \times 1.46 + 1.46$	6.78
Request_IRES_data	100	1.46	24	No	0	$\lceil 24/0.96 \rceil \times 0.19 + \lceil 24/50 \rceil \times 0.29 + \lceil 24/10 \rceil \times 1.82 + \lceil 24/200 \rceil \times 2.18 + \lceil 24/200 \rceil \times 1.46 + \lceil 24/200 \rceil \times 1.46 + 1.46$	6.94
Telemetry_Response	62.5	3.24	30	No	0	$\lceil 30/0.96 \rceil \times 0.19 + \lceil 30/50 \rceil \times 0.29 + \lceil 30/10 \rceil \times 1.82 + \lceil 30/200 \rceil \times 2.18 + \lceil 30/200 \rceil \times 1.46 + \lceil 30/200 \rceil \times 1.46 + \lceil 30/100 \rceil \times 1.46 + 3.24$	8.37
Process_IRES_data	100	8.26	50	Yes	90.86	$\lceil 50/0.96 \rceil \times 0.19 + \lceil 50/50 \rceil \times 0.29 + \lceil 50/10 \rceil \times 1.82 + \lceil 50/200 \rceil \times 2.18 + \lceil 50/200 \rceil \times 1.46 + \lceil 50/200 \rceil \times 1.46 + \lceil 50/100 \rceil \times 1.46 + \lceil 50/62.5 \rceil \times 3.24 + 8.26 + abort$	miss

Table 3: Schedulability analysis of MSA(PCP) for the Olympus AOCS example