Optimistic Secure Real-Time Concurrency Control Using Multiple Data Versions

Byeong-Soo Jeong Daeho Kim Sungyoung Lee
Dept. of Computer Engineering
Kyung Hee University
Seoul, Korea
{jeong, tonkey, sylee}@nms.kyunghee.ac.kr

Abstract

In many real time applications, security is an important requirement, since the system maintains sensitive information to be shared by multiple users with different security levels. A secure real-time database system must satisfy not only logical data consistency but also the timing constraints and security requirements associated with transactions. Even though an optimistic concurrency control method outperforms locking based methods in firm real-time database systems, in which late transactions are immediately discarded, existing secure real-time concurrency control methods are mostly based on locking. In this paper, we propose a new optimistic concurrency control protocol for secure real-time database systems. We also compare the performance characteristics of our protocol with locking based methods while varying workloads. The results show that optimistic concurrency control performs well over a wide range of system loading and resource availability conditions.

1. Introduction

Depending on the application, database systems need to satisfy some additional requirements over and above logical data consistency. In real-time applications such as control systems, transactions have explicit timing constraints that they should meet while maintaining data consistency. In secure applications such as military applications, data and transactions are classified into different levels of security and the high security level information should be prevented from flowing into lower security levels. Because of these additional requirements, conventional concurrency control techniques cannot be used directly in such advanced database applications.

In real-time database systems, the deadline of a transaction is combined with its time-critical priority and system resources are scheduled in favor of the high priority transaction so that deadline-missing transactions are minimized. If we use conventional concurrency control mechanisms in a real-time database, priority inversion problems can occur due to shared data access [1]. In secure databases, a low security level transaction can be aborted or delayed by a high security level transaction due to shared data access. Thus, by aborting low-security level transactions in the predetermined manner, high-security level information can be indirectly transferred to the lower security level, in what is called a covert channel [8].

Recently, there is increasing need for supporting applications which have timing constraints while managing sensitive data in advanced database systems such as military command control systems and stock information systems. Thus, we need to integrate security requirements into real-time database systems. There has been considerable research and performance study on the concurrency control protocols for real-time databases and secure databases. But secure real-time concurrency control protocols are rarely presented. To our knowledge, SRT-2PL (Secure Real-Time Two-Phase Locking) [13] and PSMVL (Priority-driven Secure Multi-Version Locking Protocol) [14] are the only examples able to solve both real-time and secure requirements together.

As we can tell by their names, these protocols are based on locking protocols. Performance studies of concurrency control methods in real-time environments have shown that an optimistic approach can outperform

* This work was supported by International collaborative Research Program (TRP-9802-6) sponsored by Ministry of Information and Communication of Korea

locking based methods [5, 6]. A key reason for this result is that the optimistic method, due to its validation stage conflict resolution, ensures that eventually discarded transactions do not restart other transactions, unlike the locking approach in which soon-to-be-discarded transactions may restart other transactions. Even though an optimistic concurrency control method outperforms locking in firm real-time database systems, in which late transactions are immediately discarded, existing secure real-time concurrency control methods are mostly based on locking.

In this paper, we propose a new concurrency control protocol based on an optimistic method for secure real-time database systems. The proposed protocol solves the conflicts between real-time constraints and security requirements by maintaining multiple data versions and ensures serializability by appropriately marking conflicting transactions and letting them continuously proceed with the correct data versions. Our protocol eliminates the priority inversion and covert channel problems. The schedules produced by our protocol also ensure serializability. We devise data structures and several rules that can maintain multiple data versions efficiently. We also compare the performance characteristics of our protocol with locking based methods from the viewpoint of deadline-missing transactions and the degree of wasted restarts under varying workload conditions.

The rest of the paper is organized as follows. In section 2, some related work is reviewed. In section 3, we briefly describe a secure real-time database model and present our secure real-time protocol. In section 4, we discuss the logical correctness of the proposed protocol. The results of the simulation experiment are described by comparing it with locking based methods in section 5. Finally, we conclude our study in section 6.

2. Related Research

In the early stage, real-time concurrency control protocols and multi-level secure concurrency control protocols have been studied independently. The former has been researched on the basis of locking strategies or optimistic methods, including priority based scheduling approaches. In the case of using locking strategies, typical examples are Wait Promote, High Priority, and Conditional Restart algorithms [1]. These algorithms aim to schedule transactions without causing priority inversion problems. OPT-SACRIFICE, OPT-WAIT, and WAIT-50 [5] are several alternatives that exploit optimistic techniques in a real-time database environment. The optimistic protocols use priority information in the resolution of data conflicts, that is, they resolve data conflicts always in favor of higher priority transactions. In a transaction's validation stage, if there are conflicting higher priority transactions, it should be restarted (sacrificed) or blocked (waited) until higher priority transactions commit. WAIT-50 is a hybrid algorithm that controls the amount of waiting based on transaction conflict states.

Multi-level secure concurrency control protocol, whose research is mainly based on the Bell-LaPadula [2] security model, aims to maintain noninterference among transactions with different security levels. In a secure database, problems occur when there are data conflicts between different security level transactions. If we allow a low security level transaction to be aborted or blocked by a high security level transaction during the resolution of data conflicts, a covert channel can be made. The secure concurrency control protocols have been researched mainly on the basis of using multiple data versions or a single version. In the case of using multiple data versions, when different security level transactions request same data item simultaneously, they use different versions to avoid interference [8]. A major concern of these protocols is how to select a correct version in order to ensure serializability and how to maintain multiple versions considering storage overhead. In the case of using a single version, a low level transaction should not be blocked or aborted by a high level transaction. The secure 2-phase locking protocol proposed by Son [16] tries to simulate execution of basic 2PL without the blocking of low level transactions by high level transactions. Blocking occurs when a low level transaction (say $\boldsymbol{T_L}$) asks for a write operation on a data item that is already read-locked by a high level transaction (say $\boldsymbol{T_H}$). The secure 2PL protocol allocates a virtual lock to $\boldsymbol{T_L}$ and allows it to execute the write operation, without delay, on the local area. Once the lock is released by $\boldsymbol{T_H}$, the virtual lock is changed to a real lock, and an actual write operation is executed on the database.

Recently, some integrated protocols that satisfy real-time constraints and security requirements together have been proposed. Mukkamaala and Son [13] have introduced SRT-2PL, which prevents covert channels by maintaining the property of noninterference among transactions with different security levels. Since a covert channel arises when a high level transaction aborts or delays a low level transaction, the SRT-2PL removes possible conflicts by separating transactions with an access class identical to the data object from higher access class transactions. This is accomplished by maintaining two copies (a primary copy and a secondary copy) of each data object. The primary copy of an object is accessed (read/write) by transactions at the same security level as the data object. The secondary copy is accessed (read) by transactions at a security level higher than the data object. The update of the primary copy is carried out on the secondary copy through the queue that contains each update of the primary copy in the same order of update.

Park et. al. [14] propose another secure real-time protocol based on a multi-version 2-phase locking protocol. It introduces a new concept of serializability, what is called F (First-read) serializability [15], which is more general than the definition of 1-copy serializability. F-serial eliminates the limitation that transactions must read the most recent versions and provides the boundary of each version that a transaction can read when the transaction does not read the most recent version. Thus, it can provide a higher degree of concurrency than MV2PL (Multi-Version 2-Phase Locking) which is based on 1-copy serial. The other works which need to be mentioned, are [4] and [10]. In [4], a novel dual approach is proposed that allows a real-time database system to simultaneously use different concurrency control mechanisms for guaranteeing security and for improving real-time performance. In [10], a new optimistic concurrency control algorithm is presented which can avoid unnecessary restarts by adjusting serialization order dynamically.

3. Optimistic Secure Real-Time Concurrency Control

Performance studies of concurrency control algorithms for conventional database systems have shown that locking protocols outperform optimistic techniques under most operating circumstances. However, in firm real-time database systems where late transactions are immediately discarded, an optimistic technique is considered as more advantageous from the viewpoint of real-time performance, i.e., meeting timing constraints instead of transaction's response time [5, 6]. Figure 1 shows one example of such scheduling advantage. Let's consider the following schedule of transactions, ***T1*** and ***T2*** that require 5 execution time units and have deadlines, time unit 4 and 6 respectively. In the case of lock-based protocols, ***T2*** cannot meet deadline (time 6) due to blocking of soon-to-be-discarded transaction, ***T1***. On the contrary, in optimistic protocol, ***T2*** can meet the deadline and successfully commit since ***T2***'s validation happens after ***T1***'s abortion. With such advantages in mind, we propose a new secure real-time concurrency control protocol based on optimistic approach.

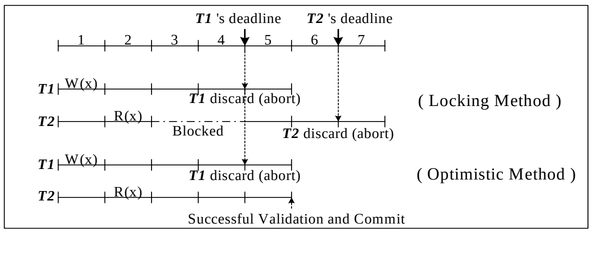

Figure 1. Transaction Scheduling

In our secure real-time database model, each transaction (and data) has its own security level and priority value assigned by considering the transaction's deadline (we assume that the priority value is unique among transactions). A basic principle of our protocol is that if data conflict occurs in the transaction's validation phase, conflict is resolved to meet real-time and security requirements. In order to meet the real-time requirement and avoid covert channels, lower priority transaction and high security level transactions should be restarted (aborted) or delayed (waited) during conflict resolution. Data conflict, which is considered in the forward validation, arises from the data set written by the validating transaction and the data set read by currently running transactions. When we solve real-time and security requirements in single scheme, the problem occurs in the case that validating transaction has low priority value and low security level and conflicting active transaction has high priority value and high security level (after now, we call this case as LL-HH conflict). If we abort a validating transaction for a real-time requirement, it can cause a security violation. Aborting active transactions violates the real-time requirement. This case happens when a low priority transaction, which has written on a data item already read (i.e., read-down) by high priority transactions, enters the validation phase. Our proposed protocol solves these conflicts between real-time and security requirements by using multiple versions of data objects.

In the proposed protocol, when a data conflict of such a type happens, we commit the validating transaction and also let the active transaction continue to progress by selecting correct versions of the data object in order to ensure serializable execution. In the transaction's validation phase, if such a case happens, the conflicting active transactions are marked with a timestamp, MTS(Marked Time Stamp) at this validation point and inserted into an MTL (Marked Transaction List) with the transaction identifier and timestamp value. After a transaction is marked, a read operation of the marked transaction is executed on the correct data version by comparing this timestamp (MTS) with that of the data version. In what follows, we now turn to describe our protocol in more detail by presenting several rules and data structures.

In the proposed protocol, transaction read/write operations are performed according to the following rules.

[***Rule 1***] A transaction's validation is processed sequentially (one at a time) and at this time a unique timestamp value is given. The timestamp value is the logical time value that indicates a logical order of transaction validation.

[***Rule 2***] At first, every write operation is performed in the transaction's local workspace. After the validation is successfully finished, the write operation is reflected in the database. At this time, a new data version is generated and the data WTS (Write Time Stamp) is recorded as the timestamp of the validation point.

Table 1. Sample History

Time / Trans.	…	10	11	12	13	14	15	16	17	18	…
T_1		$r(x)$			$w(x)$		$r(y)$		$r(z)$	**V**	…
T_2			$r(x)$	$w(y)$		$w(z)$		**V**	…		

$P(T_1) > P(T_2)$
$L(T_1) = L(x) > L(T_2) = L(y) = L(z)$
V : validation

$P(T_i)$: T_i's priority
$L(T_i)$: T_i's security level
$L(x)$: Security level of data x

[***Rule 3***] In the transaction's validation phase, if the LL-HH case happens, active transactions are marked with the timestamp value of the validation point, we call it an MTS (Marked Time Stamp), and inserted into the MTL (Marked Transaction List) with MTS value and transaction identifier.

[***Rule 4***] Read operations of marked transactions use the most recent version of data $\boldsymbol{x}$ where WTS($\boldsymbol{x}$) < MTS($\boldsymbol{T}$).

[***Rule 5***] Read operations of unmarked transactions always use the most recent data versions.

[***Rule 6***] At the time of transactions' read-downs, which means read operations by transactions at a security level higher than the data objects, a transaction identifier is inserted into the RDTL (Read-Down Transaction List) of the corresponding data version.

[***Rule 7***] When the marked transaction is committed or restarted (aborted), the corresponding node of that transaction in the MTL is deleted.

[***Rule 8***] When the transaction is committed or restarted (aborted), the corresponding transaction identifier is deleted from the RDTL of the data version if it has performed read-down operations on that version.

[***Rule 9***] In the case where the MTL changes (the case where a marked transaction is committed or restarted), all data versions but one where the WTS value is smaller than the smallest MTS value in the MTL are deleted.

[***Rule 10***] If the RDTL of the data version is empty and it does not violate [***Rule 9***], the corresponding data versions are deleted.

In our protocol, the execution of a transaction consists of three phases: read, validation, and write phase. The read phase process can be described by the following procedure.

```
read_phase($T_a$)
{
    foreach $D_i$ ($i = 1, 2, ..., m$) in RS($T_a$) {
        if ($T_a$ is marked)
            select the version of $D_i$ where WTS($D_i$) < MTS($T_a$)
            read that version
        else    read the most recent version of $D_i$
    }
    foreach $D_j$ ($j = 1, 2, ..., n$) in WS($T_a$)
        write $D_j$ in local workspace
}
```

After the read phase, a transaction goes through the validation as in the forward validation procedure. If data conflicts occur during the validation, conflict resolution is executed as follows.

```
conflict_resoution($T_v$)
{   // P(T): Priority of Transaction, T
    // L(T): Security Level of Transaction, T
    case 1: $P(T_v) > P(T_a)$ and $L(T_v) = L(T_a)$
            abort($T_a$);    commit($T_v$);
    case 2: $P(T_v) > P(T_a)$ and $L(T_v) < L(T_a)$
            abort($T_a$);    commit($T_v$);
    case 3: $P(T_v) < P(T_a)$ and $L(T_v) = L(T_a)$
            abort($T_v$);
    case 4: $P(T_v) < P(T_a)$ and $L(T_v) < L(T_a)$
            mark($T_a$);    commit($T_v$);
}
```

Once the validation of a transaction is successful, the transaction's updates are copied from the local workspace into the database in the write phase. At this time, a new version of the data object is generated with the WTS as the timestamp value of the validation point.

In order to describe our protocol more clearly, we explain step-by-step operations according to the above procedures for the sample history in Table 1. Figure 2 shows data structures used in the protocol and how it changes while executing transactions. In Figure 2, data structures enclosed by a solid line represent the initial state of data versions and MTL, and the update of data versions and MTL is represented by a dotted line. In the sample history of Table 1, until the validation of T_2 begins, transactions' read/write operations are performed without any interference. $r(x)$ and $r(y)$ of T_1 read the most recent data version, $x=30$ and $y=20$ respectively. $r(y)$ of T_2 also reads $y=20$. At this moment, since $r(y)$ of T_1 is a read-down operation, T_1's *ID* is inserted in the RDTL of data version ($y=20$). $w(x)$, $w(y)$, and $w(z)$ of T_1 and T_2 are

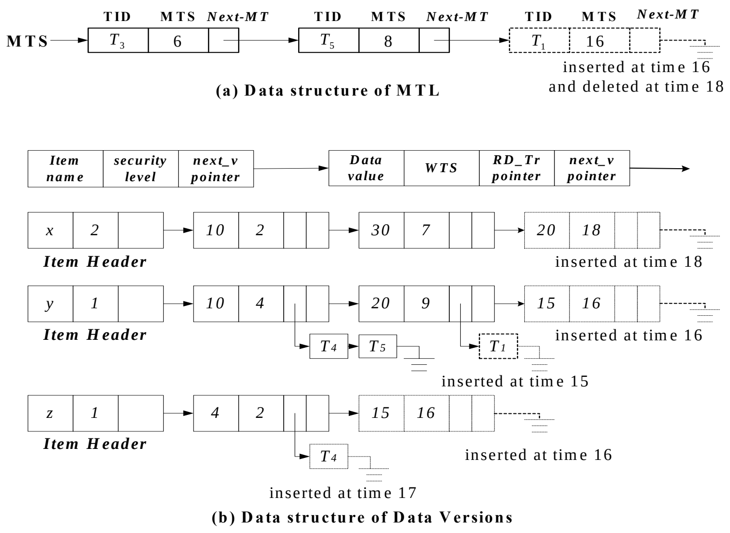

Figure 2. Data Structures

executed on each transaction's local workspaces.

During the validation of T_2 at time 16 in Table 1, LL-HH conflict happens due to data *y*. Therefore, T_1 is marked (See Figure 2 (a)) according to the above rules and after this, T_2 is committed successfully reflecting *w(y)* and *w(z)* to the database (at this moment, new versions of data *y* and *z* are generated). At time 17, since T_1 is marked by T_2, *r(z)* of T_1 selects data version, *z=4* (where WTS(*z*) < MTS(T_1)) instead of using data version, *z=10* which is the most recent version of *z*. After the successful validation of T_1, T_1 is committed. In consequence of T_1's commitment, a new version of *x* is generated and the node of T_1 in MTL is deleted. If the smallest MTS value in the MTL is changed, the version management procedure is executed according to [***Rule 9***]. For example, if T_3 is deleted from MTL (deletion of T_3 occurs when T_3 is committed or aborted), the version management procedure deletes the first version of *x* (value=*10*, WTS=*2*) since x already has a version (value=*30*, WTS=*7*) whose WTS value is smaller than the smallest MTS value, *8* in MTL.

4. The Correctness of the Protocol

In this section, we prove the correctness of the proposed protocol. To prove logical correctness (i.e., serializability), we use the simple definitions of a history and a serialization graph. In order to show that our protocol satisfies real-time and security requirements, we briefly state the results less formally in [**Lemma 3**] and [L**emma 4**].

[**Lemma 1**] All committed transactions at a single security level are serializable.

Proof : In the case where every transaction has the same security level, the execution of transactions is the same as in the existing real-time optimistic protocol of [5] since the LL-HH conflict does not occur. Thus, we know that all committed transactions are serializable as in [5]. The serial order of committed transactions is the same as the validation order of transactions.

[**Lemma 2**] All committed transactions (independent of their security level) are serializable.

Proof : By [Lemma 1], all committed transactions at a single level are serializable. Thus, we only need to prove that the committed transactions with different security levels are also serializable. Consider levels **L** and **L′** where **L > L′.** Since transactions at each level are serialized (by [Lemma 1]), let $\boldsymbol{T_1}$ **->**$\boldsymbol{T_2}$ **->... ->**$\boldsymbol{T_{m-1}}$ **->**$\boldsymbol{T_m}$ be the serialization order at level **L** and $\boldsymbol{t_1}$ **->**$\boldsymbol{t_2}$ **->... ->**$\boldsymbol{t_{n-1}}$ **->**$\boldsymbol{t_n}$ be the order at level **L′** of committed transactions. Suppose, by way of contradiction, that there is no serialization order among all these transactions. Then there is a cycle in the corresponding serialization graph so that one of the following two cases is possible (Let $\boldsymbol{T_i}$ and $\boldsymbol{T_k}$ be high security level transactions and $\boldsymbol{t_j}$ and $\boldsymbol{t_k}$ be low security level transactions):

Case 1) $\boldsymbol{T_i}$ **->**$\boldsymbol{t_j}$ **->**$\boldsymbol{t_k}$ **->**$\boldsymbol{T_i}$: Since only read-down operations are allowed for higher level transactions on lower level data objects, $\boldsymbol{T_i}$ **->**$\boldsymbol{t_j}$ implies that there is a data object $\boldsymbol{x}$ (at level **L′**) that was read (read-down) by $\boldsymbol{T_i}$ and modified later by $\boldsymbol{t_j}$. In our protocol, this can happen in two ways, (**I**) $\boldsymbol{T_i}$ is committed before $\boldsymbol{t_j}$ through a successful validation and (**II**) T_i is committed after t_j's commitment. In the case of (**II**), T_i is marked during the validation of t_j. Similarly, $t_k \rightarrow T_i$ implies that there is a data object y (at level **L′**) that was written by t_k and later read by T_i. In the case of (**I**), since T_i's timestamp (at the validation point) is smaller than t_j's timestamp and the timestamp of t_j is smaller than that of t_k (by $t_j \rightarrow t_k$), $t_k \rightarrow T_i$ can not happen. Also in the case of (**II**), T_i's read operation uses only the data version whose WTS value is smaller than MTS(T_i) and the WTS value of the data version which is written by t_k, is always larger than MTS(T_i) (since timestamp(t_k) > timestamp(t_j) and timestamp(t_j) = MTS(T_i)). Thus, neither can $t_k \rightarrow T_i$ happen in the case of (**II**). Therefore, the cyclic serialization graph of $T_i \rightarrow t_j \rightarrow t_k \rightarrow T_i$ cannot occur in our protocol.

Case 2) $t_i \rightarrow T_j \rightarrow T_k \rightarrow t_i$: This case also can be similarly contradicted as in the above case. In the cyclic serialization graph, $t_i \rightarrow T_j$ implies that there is a data object x (at level **L′**) that was modified by t_i and later was read by T_j. This means that t_i is committed before T_j. Similarly, $T_k \rightarrow t_i$ implies that there is a data object y (at level **L′**) that was read by T_k and later modified by t_i. Figure 3 shows the schedule of this case. In the cycle, $T_j \rightarrow T_k$ implies that T_k reads a data object z that was modified by T_j. But, as in Figure 3, T_k is marked at the time of t_i's validation and after T_k reads only the data version whose WTS value is smaller than the timestamp of t_i. Thus, $T_j \rightarrow T_k$ cannot occur in our protocol. Therefore, the cyclic serialization graph of $t_i \rightarrow T_j \rightarrow T_k \rightarrow t_i$ also cannot happen.

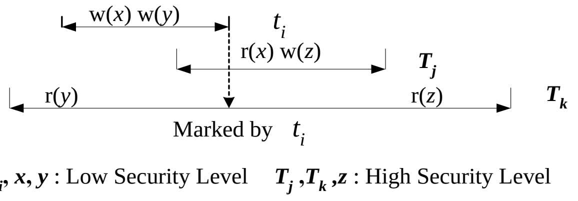
w(x) w(y)
t_i
r(x) w(z)
T_j
r(y)
r(z)
T_k
Marked by t_i
t_i, x, y : Low Security Level
T_j, T_k, z : High Security Level

Figure 3. Example of the Schedule

The same proof can be extended even when transactions from more than two levels are considered. Thus, all committed transactions by the proposed protocol are serializable. ■

[**Lemma 3**] In our protocol, low security level transactions are not delayed (Delay Secure) or aborted (Recovery Secure) by data conflicts with higher security level transactions. Data updates by higher security level transactions are not seen by lower security level transactions (Value Secure): (the necessary and sufficient conditions for a secure database).

Proof : Since our protocol is based on an optimistic method, transaction's delay, which is caused by blocking in locking protocol, does not occur. Also by [***Rule 3***] and [***Rule 4***], transaction abort (restart) by higher security level transactions does not occur. Neither does transaction write-down occur because of the Bell-LaPadula security model. ■

[**Lemma 4**] High priority transactions are neither aborted (restarted) nor delayed (blocked) by low-priority transactions due to data contention on low security level data objects.

Proof : Same as the proof of [**Lemma 3**]. ■

5. Performance Evaluation

In this section, we describe the structure and details of the simulation model and experimental environment that were used to evaluate the performance of our protocol. We also present performance results from our experimental comparisons with locking based protocols.

5.1 Simulation Model

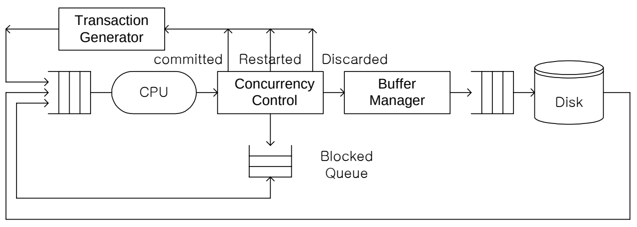

Figure 4. The Simulation Queueing Model

Table 2. Simulation Parameters

Parameters	Value
Database Size	100
Slack Value	1.5~4.5
Security Level	3
Transaction Size	5~20
CPR Time	3.0 ms
Disk Time	25 ms
Buffer Hit	0.7
Transaction Arrival Rate	5~30 tr./sec

Figure 4 illustrates the simulation queueing model designed for the experiment. Each transaction, which consists of a sequence of page read and write operations, is generated from the transaction generator and then inserted into the CPU queue. In our simulation, transactions arrive in a Poisson stream, i.e. their inter-arrival times are exponentially distributed. We varied the mean transaction arrival rate in order to experiment with different (heavy or light) transaction workloads. When a transaction in the CPU queue is selected by its priority which is calculated from deadline information, it must go through the given concurrency control protocol to obtain a data access permission. If data access is granted, the transaction

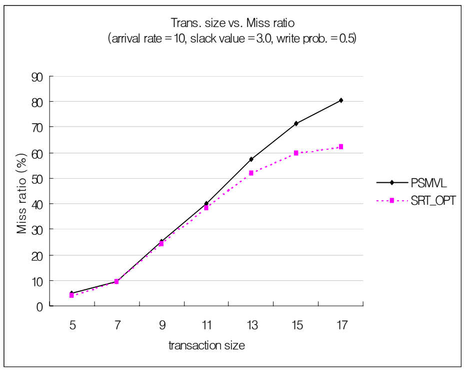

Figure 5. Miss Ratio vs. Arrival Rate (Write Probability = 0.5)

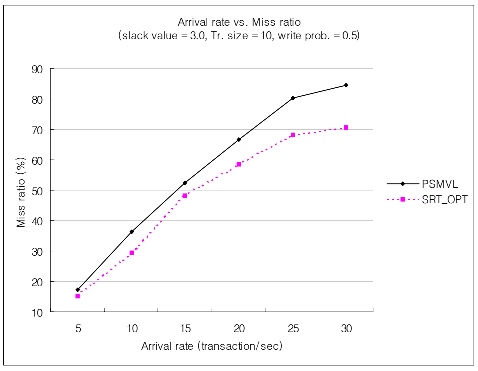

Figure 6. Miss Ratio vs. Arrival Rate (Write Probability = 0.1)

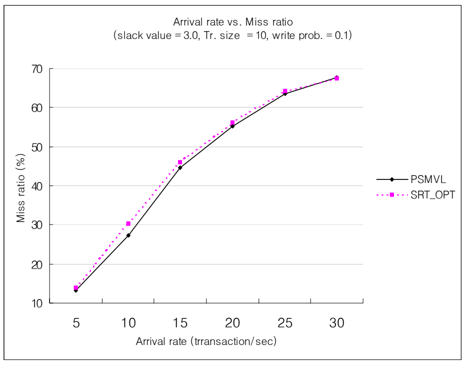

Figure 7. Miss Ratio vs. Transaction Size

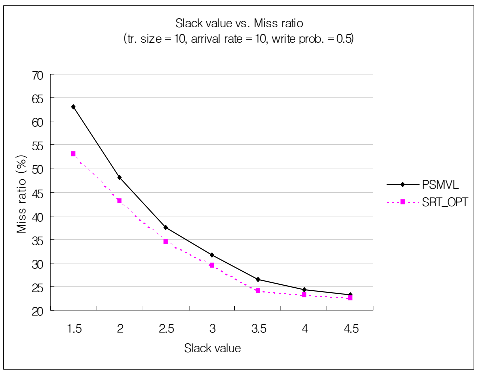

Figure 8. Miss Ratio vs. Slack Value

proceeds to perform the data operations, which consist of disk access and CPU computations. If the data access request is denied, the transaction will be placed into a blocked queue. A transaction, which is to be aborted by the protocol, can be restarted or discarded if it cannot meet the deadline. Transactions that have already missed their deadlines are immediately discarded from CPU queue.

Table 2 summarizes the key parameters that are used in the simulation. The database itself is modeled as a collection of data pages in disks. The slack value is used to calculate the deadline according to the following formula: ***deadline = arrival_time + slack * transaction_size * execution_time***. This deadline information is also used to assign the priority of each transaction. We used the EDF (Earliest Deadline First) algorithm for the priority assignment of transactions, varying the slack value for the purpose of representing different real-time environments. The ***security_level*** of a transaction is randomly given at its generation and data objects are initially classified into different ***security_levels***. The ***transaction_size*** represents the number of read/write operations in a transaction and it is also varied in our simulation. The use of a database buffer pool is simulated using probability. When a transaction attempts to read a data page, the system determines whether the page is in memory or disk using the probability, ***Buffer_Hit***. The ***execution_time***, which is used to calculate the deadline, is obtained as follows: ***execution_time = CPU_time + (1 - Buffer_Hit) * Disk_time.***

5.2 Experiments and Results

For experiments intended to compare our optimistic protocol with locking protocols, we implemented PSMVL[14] which is a lock-based real-time secure protocol. Since it uses multiple versions of data in order to satisfy real-time and security requirements as we do, we can examine different behaviors between them. As mentioned in [10], it is difficult to compare the performance of an optimistic protocol with that of a locking protocol due to the significant difference in their implementation. In our simulation, we only consider the impact of data contention in two schemes while assuming that the processing overhead of each protocol is the same. The simulation program was written by CSIM library on the Sun UltraSparc workstation.

As for the primary performance metrics, we used ***Miss Ratio***, which is calculated as follows: ***Miss Ratio = 100 * (# of discarded transactions / # of transactions arrived)*** and ***Restart Ratio***, which measures the average ratio of transaction restarts. We measured these metrics while varying system workloads, transaction size, and deadline tightness (by using different slack values). Additionally, we measured other statistical information, including average transaction blocking time and unnecessary

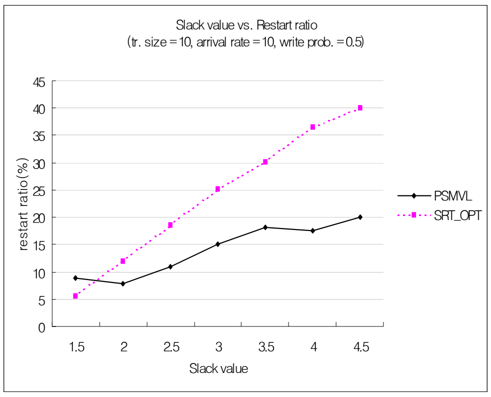

Figure 9. Restart Ratio vs. Arrival Rate

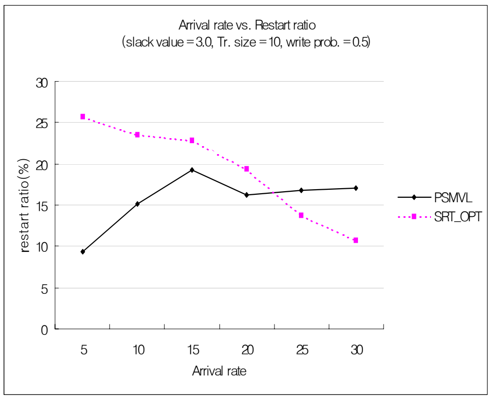

Figure 10. Restart Ratio vs. Slack Value

restart/blocking time caused by soon-to-be-discarded transactions.

Figures 5 and 6 (in the graph SRT_OPT represent our protocol, Secure Real-Time Optimistic protocol) show the miss ratios of two concurrency control schemes under different degrees of data contention (write probability = 0.1 and 0.5, respectively) while varying system workloads (by varying transaction arrival rate). As we can see from Figure 5, in the case where write probability = 0.1 (i.e., the operations in a transaction are mostly read) the performance of the two schemes is almost the same. The reason is that the two schemes behave similarly since there is not much data contention. However, in the case where data contention increases with high write probability and heavy workloads as in Figure 6, the performance difference becomes bigger. This is due to the fact that in the locking scheme many restarts (aborts) are made unnecessarily by soon-to-be-discarded transactions. The same reasoning can be applied in Figure 7. That is, if the transaction size increases, the possibility of data contention becomes larger. Figure 8 shows the miss ratio with different slack values (deadline tightness). From Figure 8, we can see that if the transaction has enough slack time, most transactions can meet the deadline in both schemes by restarting conflicting transactions.

Figures 9 and 10 show the restart ratio with different arrival rate and slack values, respectively. The restart ratio represents the average restart percentage of each transaction during the simulation. Actually, this metric does not give a meaningful comparison between the two schemes because the data conflicts are resolved differently in the two schemes. That is, in an optimistic approach, conflict resolution is accomplished only by restarting (aborting) one of the conflicting transactions. On the other hand, the locking based method resolves conflicts by restarting or sometimes by blocking a transaction. However, we can see that the restart ratio decreases as the system workload becomes heavier in Figure 8. The reason is that the chance of restarting becomes low as a heavier workload yields more deadline missing transactions. Figure 10 also shows similar reasoning. Enough slack time gives more chance of successful completion by restarting transactions more frequently.

6. Conclusion

In this paper, we have presented a new secure real-time concurrency control scheme based on an optimistic approach. Our protocol solves the conflict between real-time constraints and security requirements by maintaining multiple data versions, and it ensures serializability by appropriately marking conflicting transactions and letting them continuously proceed with the correct data versions. To evaluate its performance characteristics, we compared it with the existing locking based method, PSMVL. Even though we did not consider the protocol overhead of each scheme in detail, we could compare the effect of data contention between two schemes. Our experiments show that an optimistic approach gives better performance than the locking scheme in the case of high data contention because the forward validation of an optimistic method can reduce unnecessary restarts to a higher degree than the locking method can.

7. Reference

[1] R. K. Abbott and H. Garcia-Molina , "Scheduling Real-Time Transactions : A Performance Evaluation", In *ACM Transactions on Database Systems*, 17, pp513-560, 1992.

[2] D. E. Bell and L. J. LaPadula, "Secure Computer Systems : Unified Exposition and Multics Interpretation", *The Mitre Corp*., 1976.

[3] Rasikan David, Sang H. Son and Ravi Mukkamala, "Supporting Security Requirements in Multilevel Real-Time Databases", In *IEEE Symposium on Security and Privacy*, Oakland, CA, pp 199-210, 1995.

[4] B. George and J. Haritsa, "Secure Transaction Processing in Firm Real-Time Database System", In *Proceedings of ACM SIGMOD*, pp462-473, 1997.

[5] J. R. Haritsa, M. J. Carey, and M. Livny, "Dynamic Real-Time Optimistic Concurrency Control", In *11th IEEE*

Real-Time Systems Symposium, 1990.

[6] J. R. Haritsa, M. J. Carey, and M. Livny, "On Being Optimistic about Real-Time", In *Proceedings of the 1990 ACM PODS Symposium*, pp331-343, April 1990.

[7] Soek-Hee Hong, Myung-Ho Kim, Yoon-Joon Lee, "Methods of Concurrency Control in Real-Time Database", In *Korean Information Science Society Review*, Vol. 11, No. 1, pp 26-36, 1993.

[8] Thomas F. Keefe, W. T. Tsai, Jaideep Srivastava, "Database Concurrency Control in Multilevel Secure Database Management Systems", In *IEEE Transaction on knowledge and Data Engineering*, vol 5, no. 6, pp1039-1055, 1993.

[9] Young-kuk Kim and Sang H. Son, "Predictability and Consistency in Real-Time Database Systems", *Advances in Real-Time Systems*, S. H. Son (ed.), Prentice Hall, pp 509-531, 1995.

[10] J. Lee and S. H. Son, "Using Dynamic Adjustment of Serialization Order for Real-Time Database Systems", In *14th IEEE Real-Time Systems Symposium*, pp66-75, 1993.

[11] J. Mcdermott and S. Jajodia, "Orange Locking Channel-Free Database Concurrency Control via Locking", In *IFIP WG 11.3 Workshop in Database Security*, pp267-284, 1992.

[12] Ira S. Moskowitz and Myong H. Kang, "Covert Channels - Here to Stay?", In *Proceedings of COMPASS 94*, pp235-243, 1994.

[13] R. Mukkamala and S. H. Son, "A Secure Concurrency Control Protocol for Real-Time Databases", In *Annual IFIP WG 11.3 Conference of Database Security*, Rensselaerville, New York, Aug. pp 235-253, 1995.

[14] Chanjung Park, Seog Park, and Sang H. Son, "Priority-driven Secure Multi-version Locking Protocol for Real-Time Secure Database Systems", In *Proceedings of IFIP 11th Working Conference on Database Security*, August 1997.

[15] Chanjung Park and Seog Park, "Alternative Correctness Criteria for Multi-version Concurrency Control and a Locking Protocol via Freezing", In *International Database Engineering and Applications Symposium (IDEA '97)*, pp. 73-81, August 1997.

[16] S. H. Son and R. David, "Design and Analysis of a Secure Two-Phase Locking Protocol," In *18th International Computer Software and Applications Conference (COMPSAC'94)*, Taipei, Taiwan, pp 374-379, 1994.

[17] S. H. Son, R. David, and B. Thuraisingham, "An Adaptive Policy for Improved Timeliness in Secure Database Systems", In *Annual IFIP WG 11.3 Conference of Database Security*, pp223-233, 1995.