PSMVL : A Concurrency Control Protocol for Real-Time Secure Database Systems

Chan-jung Park and Seog Park

Abstract

The applications for real-time database systems must satisfy timing constraints. Typically the timing constraints are expressed in the form of deadlines which are represented by priorities to be used by schedulers. In many real-time applications, since the system maintains sensitive information to be shared by multiple users with different levels of security clearance, security is another important requirement. As more advanced database systems are being used in applications that need to support timeliness while managing sensitive information, protocols that satisfy both requirements need to be developed. In this paper, we propose a new priority-driven secure multiversion locking(PSMVL) protocol for real-time secure database systems. The schedules produced by PSMVL are proven to be one-copy serializable. We have also shown that the protocol eliminates covert channels and priority inversions. The details of the protocol, including the compatibility matrix and the version selection algorithm are presented. The results of the performance comparisons of our protocol with other protocols are described.

I. Introduction

A multilevel secure database management system (MLS/DBMS) is a transaction processing system that each user who accesses the system has a clearance level and each data item maintained by the system has a unique classification level[9, 10]. In order to control all the accesses to the database, mandatory access control (MAC) mechanisms are adopted in MLS/DBMS. With MAC mechanisms, the sensitive data can be protected by permitting accesses by only the users whose security levels are higher than or equal to the levels of data. In order for MLS/DBMS to be correct, it has to meet security requirements in addition to satisfying logical data consistency. The most important requirements for multilevel security are the eliminations of both covert channels between transactions of different levels and the starvations of high-level transactions[9, 10]. In principle, MLS database systems should be used for any system that contains sensitive data[14].

Meanwhile, in real-time database management systems (RTDBMSs), transactions have explicit timing constraints such as deadlines[8, 13]. The time criticalness(priority) of a transaction usually derives from both its timeliness requirement and its importance. RTDBMS must satisfy timing constraints associated with transactions and maintain data consistency. There are increasing needs for supporting applications which have timing constraints while managing sensitive data in advanced database systems. To support such applications, we must integrate real-time transaction processing techniques into MLS/DBMS, namely MLS/RT DBMS[15]. Since MLS/RT DBMS needs to support both MLS and RT requirements, it is easy to see that protocols for MLS/RT DBMS could be more complicated than those for MLS/DBMS or RTDBMS.

There are several on-going research projects on concurrency control protocols for RTDBMS and MLS/DBMS. However, the protocols for MLS/RT DBMS are rarely presented. Recently, SRT-2PL(Secure Real-Time Two Phase Locking) protocol[12] for MLS/RT DBMSs was proposed. In the protocol, a data manager maintains a primary copy and a secondary copy for each data item to satisfy two requirements. In addition, the data manager also maintains a single queue which contains the updates that have been performed on the primary copy but yet to be performed on the secondary copy for ensuring serializability. The primary copy of a data item is used for the read and write operations of the same level transactions, while the secondary copy of the data item is used for the read operations of high level transactions. However, there still exists the priority inversion problem[1] because of the superposition operation of the queue on the secondary copy. That means when a high-level high priority

Manuscript received March 14, 1997; accepted August 16, 1997.

C. J. Park is with Department of Computer Science, Sogang University, Korea.
S. Park is with Department of Computer Science of the College of Engineering, Sogang University, Korea.

1) A *priority inversion* occurs when a high-priority transaction is delayed by a low-priority transaction. It is not desirable in real-time database systems.

transaction T requests a read operation on a low level data item x, if other transaction reads the secondary copy of x, then T is blocked.

In this paper, we propose a priority-driven secure multiversion locking protocol, called PSMVL, for MLS/RT DBMSs. The proposed protocol ensures that high-priority transactions are not blocked due to low-priority transactions for timing constraints, while low-level transactions are not interfered by high-level transactions to avoid covert channels.

The protocols based on multiversions require more amount of storage than those based on a single version. However, the proposed protocol is based on multiversion scheme for some reasons. First, disk prices have come down dramatically, the disk space needed to store multiple versions is cheaper. Second, the concurrency control protocol which maintains a single version of each data item, such as 2PL-HP[2] and OPT-wait[7] or OPT-sacrifice [7], cannot avoid the starvations of high level transactions, because low level transactions should neither be delayed nor be aborted to prevent covert channels. And the protocol cannot eliminate the starvations of low priority transactions, since low priority transactions can be delayed or aborted by high priority transactions. Third, the protocols which maintain two versions of each of data items can partly resolve the above starvation problems. However, due to the limited number of versions, when a high priority transaction at a high level conflicts with a low priority transaction at a low level on the same data item, the protocols sacrifice one of the requirements. Therefore, multiversion scheme is considered appropriate to satisfy all the requirements for MLS/RT DBMSs. In addition, since our protocol increases the degree of concurrency due to multiversions. We have shown that the histories[2] produced by the protocol are one-copy serializable[6].

The rest of the paper is organized as follows. In Section 2, we present the security model of this paper and then introduce the features of transactions in RTDBMS. In Section 3, we classify the transactions according to their characteristics to discuss the conflicting natures of the requirements, and present the PSMVL protocol and the version selection algorithm. In Section 4, we prove the correctness of the protocol and show that it ensures serializability, security requirements, and no priority inversion. After the performance results of the protocol are presented in Section 5, we conclude the paper in Section 6.

II. Background

Let us the security level of a transaction T is denoted by $L(T)$ and the security level of a data item x is denoted by $L(x)$. When transactions access data items, the following security policies are adopted as ours[4].

(1) *Simple security property* for read operations: A transaction T is allowed to read a data item x if and only if $L(T) \geq L(x)$.

(2) *Restricted star property(☆-property)* for write operations: A transaction T is allowed to write into a data item x if and only if $L(T) = L(x)$.

The above two restrictions are intended that sensitive data are protected by permitting only the users whose security levels are higher than or equal to the levels of data. In other words, read/write operations at the same level and read operations at the lower level(read-down) are allowed.

A key feature of RTDBMS is that each transaction has timing constraints[1]. The concept of *value function* is adopted as the way of representing the timing constraints of real-time transactions. For each transaction, the output of the corresponding value function expresses the amount of profit that can be obtained by the completion of the transaction before its deadline. Since it is more advantageous to the system for transactions with the largest values to be completed before their deadlines, *high-priority* is given to the transactions that have a large output value. At run time, high-priority transactions should precede low-priority transactions.

In this paper, we adopt the priority assignment policy proposed in [16]. In that policy, each transaction has an initial priority and a start-timestamp. The initial priority of a transaction indicates the criticality of the transaction. The practical priority consists of the initial priority and the start-timestamp.

III. The PSMVL protocol

In this section, we examine the conflicts between real-time and security requirements, and then present the protocols and related rules.

Let T_i and T_j be transactions in a conflicting mode and let $P(T_i)$ and $L(T_i)$ be the priority of T_i and the security level of T_i, respectively. Then, there are three possible cases for the priorities of these transactions: (1) $P(T_i)=P(T_j)$, (2) $P(T_i)>P(T_j)$, (3) $P(T_i)<P(T_j)$. Since (2) and (3) are symmetric, without loss of generality we can consider just one, say (2). Therefore, we can assume that $P(T_i)$ is higher or equal to $P(T_j)$. In addition, there are three cases for the security levels of these transactions: (1) $L(T_i)=L(T_j)$, (2) $L(T_i)>L(T_j)$, (3) $L(T_i)<L(T_j)$.

Let P_{High}, P_{Low}, and P_{Eq} be the priorities and $P_{High}>P_{Low}$. Let L_{High}, L_{Low}, and L_{Eq} be the security levels and $L_{High}>L_{Low}$. In Table 1, L_{Eq} is used in the case that two transactions have the same security level. Table 1 shows all possible combinations of priority and security level pairs between T_i and T_j.

In the first case, the priorities and the security levels of the two transactions are the same. Therefore, the only concern is ensuring serializability. Security requirements and timing constraints

2) A *history* indicates the order in which the operations of transactions are executed relative to others.

Table 1. The priorities and the security levels between T_i and T_j.

Trans. / Cases	T_i		T_j	
	Priority	Security level	Priority	Security level
1	P_{Eq}	L_{Eq}	P_{Eq}	L_{Eq}
2	P_{Eq}	L_{Low}	P_{Eq}	L_{High}
3	P_{Eq}	L_{High}	P_{Eq}	L_{Low}
4	P_{High}	L_{Eq}	P_{Low}	L_{Eq}
5	P_{High}	L_{Low}	P_{Low}	L_{High}
6	P_{High}	L_{High}	P_{Low}	L_{Low}

can be ignored in this case. In the second and the third cases, the priorities of the two transactions are the same. Hence, timing requirements can be ignored and the low level transaction should not be delayed by the high level transaction. In the fourth case, the security levels of the two transactions are the same. The transactions must be scheduled so that they meet the timing constraints as well as the logical consistency. In this case, any protocol based on the multiversion scheme for RTDBMS can be used. In the fifth case, since $P(T_i)>P(T_j)$, T_i must be followed by T_j in order to prevent priority inversion. In addition, T_i can neither be delayed nor aborted by T_j to avoid covert channels. Both requirements can be satisfied by having T_i precede T_j.

The problem occurs in the sixth case where $P(T_i)>P(T_j)$ and $L(T_i)>L(T_j)$. Since $P(T_i)>P(T_j)$, T_i should not be blocked by T_j. On the other hand, T_j cannot be blocked by T_i in order to avoid covert channels. We call this kind of conflict *HH/LL-conflict*. It is a conflict between a high security level transaction with high-priority and a low security level transaction with low-priority. Ideally, T_i should precede T_j because of their priorities. We resolve the HH/LL-conflicts by using the proposed PSMVL protocol.

1. Compatibility Matrix

Like the multiversion two phase locking(MV2PL) protocol[5], PSMVL has three types of locks: read, write, and certify locks. The locks are governed by the compatibility matrix in Figure 1. Since no conflict occurs between read/write or write/write operations, the certify locks are needed in order to get the correct synchronization among transactions. The scheduler adopting PSMVL protocol acquires read and write locks before processing read and write operations, respectively. When a transaction is about to commit, the scheduler converts all of the transaction's write locks into certify locks.

We only consider the cases where lock requesters and lock holders have different security levels. As already mentioned, P_{High} and P_{Low} are priorities such that $P_{High}>P_{Low}$ while L_{High} and L_{Low} are security levels such that $L_{High}>L_{Low}$. Let $T_H(P_H, L_H)$ be a lock holder with priority P_H and security level L_H. Similarly, let $T_R(P_R, L_R)$ be a lock requester with priority P_R and security level L_R. The conflicts between all operations of lower level transactions and write or certify operations of higher level transactions cannot occur because of our security policy. There are four cases based on both the priorities and the security levels.

T_R \ T_H	R	W	C
R	Y	–	–
W	Y	–	–
C	Y^A	–	–

(a) $T_H(P_{Low}, L_{High})$ and $T_R(P_{High}, L_{Low})$; $T_H(P_{Eq}, L_{High})$ and $T_R(P_{Eq}, L_{Low})$

T_R \ T_H	R	W	C
R	Y	–	–
W	Y	–	–
C	Y^*	–	–

(b) $T_H(P_{High}, L_{High})$ and $T_R(P_{Low}, L_{Low})$

T_R \ T_H	R	W	C
R	Y	Y	Y^*
W	–	–	–
C	–	–	–

(c) $T_H(P_{Low}, L_{Low})$ and $T_R(P_{High}, L_{High})$

T_R \ T_H	R	W	C
R	Y	Y	N
W	–	–	–
C	–	–	–

(d) $T_H(P_{High}, L_{Low})$ and $T_R(P_{Low}, L_{High})$; $T_H(P_{Eq}, L_{Low})$ and $T_R(P_{Eq}, L_{High})$

R: Read W: Write C: Certify –: impossible
Y: Yes(shared lock) N: No(exclusive lock)
Y^*: yes due to security levels or priorities.
Y^A: yes but the lock holder is aborted

Fig. 1. The compatibility matrices for PSMVL.

In Figure 1 (a), $P(T_R)\geq P(T_H)$ and $L(T_R)<L(T_H)$. For priority and security reasons, T_R cannot be blocked. Therefore, T_H should be aborted. The abortion of T_H helps that T_H can read more recent data without violating any requirements.

In Figure 1 (b), $P(T_R)<P(T_H)$ and $L(T_R)<L(T_H)$. Under the BLP model, this situation can occur only when the operation of T_R is write while the operation of T_H is read-down. In this case, T_H cannot be blocked in order to avoid priority inversion and T_R cannot be delayed for security reasons. T_H is inserted into HH-list(x) where x is the data item T_R writes. HH-list(x) is used for keeping the orderings of priorities and is defined in the next section in detail.

In Figure 1 (c), $P(T_R)>P(T_H)$ and $L(T_R)>L(T_H)$. In this case, T_R cannot be blocked because of its priority and T_H should not be delayed in order to avoid covert channels. Since T_R and T_H cannot be blocked, T_R can share a lock and T_R is inserted into HH-list(x) where x is the data item T_H writes. In Figure 1 (d), $P(T_R)\leq P(T_H)$ and $L(T_R)>L(T_H)$. T_R cannot block T_H because of its priority and security level. Therefore, T_R is blocked until T_H commits.

2. Data Structures

Concurrency control protocols based on multiversion locking use 2PL for write/write synchronization and version selection for read/write synchronization[6]. When a transaction is about to choose a version of a data item, the most recent commit version

is commonly used. However, since there exist HH/LL-conflicts in the environment where RT and MLS requirements should be considered together, in order to resolve HH/LL-conflicts, the certify operation of a low level transaction with low priority in (b) of Figure 1 is permitted, and the read operation of a high level transaction with high priority is permitted in (c) of Figure 1. Therefore, additional rules for version selection are required.

In an MLS/RT DBMS, each data item has its own security level and each transaction has a priority and a security level. We define three data types such as *Data_itemT*, *VersionT*, and *Read_downT* for the management of versions. Data_itemT is a data type for each of data items, while VersionT is used for storing the versions of each data item. And Read_downT is a data type for the data items which are read by high level transactions.

In an MLS/RT DBMS, each data item has its own security level and each transaction has a priority and a security level. Each data item contains two fields: *level* and *version*. The *level* represents the security level of a data item and it must be trusted.

The *version* is the field for a version, and contains a *timestamp*, a *value*, a *hhllptr*, and a *vlink*. The *vlink* is the pointer to the next version. The *hhllptr* is a pointer that resolves HH/LL-conflicts and maintains the number of higher security level transactions that read down the version.

Let T_j and T_k be two transactions with the HH/LL-conflict relationship on some data item x. Let $P(T_j)>P(T_k)$ and $L(T_j)>L(T_k)=L(x)$. In our security policy, this situation can occur when T_j executes a read down operation while T_k executes a write operation. The *hhllptr* is the field of x_i that T_j reads, *i.e.*, x_i is the old committed version of x available to T_j. Since the high priority transaction T_j can read old versions of x, T_j needs not to be blocked until the lower level transaction T_k writes. The *hhllptr* must be trusted. The *hhllptr* consists of three fields: *level*, *count*, and *clink*. The *level* and the *count* represent the security level of transactions that read down the version in HH/LL-conflicting mode and the number of the transactions, respectively. The *clink* is the pointer that points to the next node for lower level transactions in HH/LL-conflicting mode.

3. Algorithms for Version Management

When a HH/LL conflict occurs, the following procedure, called *HH/LL-procedure*, is needed for maintaining the version information. HH/LL-procedure is presented in Algorithm 1.

For each data item x, we maintain a list of transactions, denoted by *HH-list(x)*, in order to preserve the orderings of priorities. The *HH-list(x)* is a list of higher priority and higher security level transactions that are active when another transaction executes a write operation. HH-list(x) can be obtained by a lock table[3]. If a transaction T_j writes x at t_{now}, while another transaction T_i with higher priority and higher level than T_j is reading x, then T_i is inserted in HH-list(x). This insertion means that HH/LL-conflict can occur in the future because $P(T_i) > P(T_j)$. When a transaction reads a data item, HH-list is used to select the appropriate version according to its priority in the version selection algorithm.

```
/* When a transaction Tj selects the version xi of x, the following steps
   are required. Discussion of the version selection algorithm will be
   provided in a later section. In the algorithm, 'u->v' denotes that v is
   a member of u. */

<Type declaration>
xi : Data_itemT
new, node : Read_downT
FIND : boolean type whose value is either TRUE or FALSE.

    if (xi->hhllptr is null) then
        create a node new;
        new->count = 1;
        new->level = L(Tj);
        xi->hhllptr = new;
    else
        node = xi->hhllptr;
        FIND = FALSE;
        while (node is not null) do
            if (node->level is L(Tj)) then
                node->count = node->count + 1;
                FIND = TRUE;
                break the loop;
            end if
        end while
        if (FIND is FALSE) then
            create a node new ;
            new->count = 1;
            new->level = L(Tj);
            append new to xi->hhllptr;
        end if
end if
```

Algorithm 1. HH/LL-procedure.

Three different operations can be performed on a version: creation, deletion, and selection. Since there is no write/write conflict in the PSMVL protocol, a new version can be created without delay. Because of HH/LL-conflicts, two or more old versions must be stored. The version that is older than the latest committed version can be deleted when there exist no high level transactions that read that version. Let $T_i(P_i, L_i)$ be a transaction with priority P_i and security level L_i. When $T_i(P_i, L_i)$ is about to read a data item x, the version selection algorithm (Algorithm 2) selects an appropriate version of x for T_i.

```
/* When a transaction Ti reads a data item x, this procedure specifies
   the steps for selecting the right version of x. In the algorithm, 'u->v'
   denotes that v is a member of u. */
```

3) A lock table contains the information that which transactions have locks on some data items.

```
<Type declaration>
        x : Data_itemT
        hhllnode : Read_downT
        FIND : boolean type whose value is either TRUE or
            FALSE.

if Ti has a write lock on x, then
        Ti must read the version Ti writes;
else
        let hhllnode(a variable) be the node linked to the
        hhllptr of the first version of x;
        FIND = FALSE;
        for (all versions of x) do
            if (hhllnode is null) then
                let hhllnode be the node linked to the hhllptr
                of the next  version of x;
            else
            /* hhllnode is not null, i.e., some higher level
               transactions already read down the version */
                for (all nodes linked to the hhllnode) do
                        find a node such that the level of the
                        node is less than or equal to L(Ti);
                        if (there exists such a node) then
                                FIND = TRUE;
                                mark the version as xj;
                                break the loop;
                        end if
                end do
            end if
        end do
        if (FIND is TRUE) then
            return xj;
        else
            find the version such that it is the latest committed
            version of x before Ti reads the read operation;
            return the version;
        end if
end if
```

Algorithm 2. Version selection algorithm.

4. The Protocol

Before we present our protocol, we define the following timestamp assignment rules for our protocol that are used in our protocol to select appropriate versions.

```
/* Let x be a data item and T, T' be transactions.  In addition, let R,
W, and C be read, write, and certify operations, respectively.  */

case 1: T requests a read operation on x
    if (x is locked with W or C) then
        if (any lock-holder has a higher or equal priority)
        then
            if (any lock-holder has a lower or equal security
               level) then
                block lock-requester ;
            end if
        else
            /* lock holder has a lower priority */
            Let T' be the transaction that has lower priority
                than that of T;
            if (L(T') is lower than L(T)) HH/LL-procedure();
            else if (L(T') is the same as L(T)) then
                for (all T' which has C on x) do
                    convert the lock from C to W;
                end do
            end if
        end if
    end if
    grant a read lock to T;

case 2: T requests a write operation on x
    Let T' be any transaction that already has a lock on
    x(lock holder).
    if (P(T') is higher than or equal to P(T)) then
        if (L(T') > L(T)) then HH/LL-procedure();
        else /* P(T') < P(T) */
            if (L(T) = L(T')) then
                T' is blocked by T;
                wait;
            end if
        end if
    end if
    grant a write lock to T;

case 3: T requests a certify operation on x
    Let T' be any transaction that already has a lock on
    x(lock holder);
    if (P(T') >  P(T)) then
        if (L(T') = L(T)) then
            the request is rejected and blocked;
        else if (L(T') > L(T)) then HH/LL-procedure();
        end if
    else if (P(T') = P(T)) then
        if (L(T') = L(T)) then
            if (T' holds a read or certify lock) then
                the request is rejected;
            else if (L(T') > L(T)) HH/LL-procedure();
            end if
    else /* P(T') < P(T) */
        if (L(T') = L(T) then
            if (T' holds a certify lock)  then
                for (all T' which has C on x) do
                    convert the lock from C to W;
                end do
            end if
        else if (L(T') < L(T)) HH/LL-procedure() ;
        end if
    end if
     grant C on x;
```

Algorithm 3. The protocol.

First, for each data item x, timestamp TS(x) is given to x when x is created. Second, for a read-only transaction T_i, the starting timestamp, S_TS(T_i) is assigned. For an update transaction T_j, both the starting timestamp, S_TS(T_j) and the committing timestamp, C_TS(T_j) are assigned. We assume that the system guarantees the uniqueness of each timestamp.

The compatibility matrix shown in Figure 1 is the basis for the PSMVL protocol. When transactions have the same priority

and the same security level, it behaves similarly to the MV2PL protocol.

Let H be a history over a set of transactions $\{T_0, T_1, \cdots, T_n\}$ produced by PSMVL. Then, H must satisfy the following properties. In order to list the properties of histories produced by executions of PSMVL, we need to include the operation f_i denoting the certification of T_i.

- ***P1.*** For every T_i, there is a unique starting timestamp S_TS (T_i); that is, $S\_TS(T_i) = S\_TS(T_j)$ iff $i = j$.
- ***P2*** For every T_i, f_i follows all of T_i's reads and writes and precedes T_i's commitment.
- ***P3*** For every $r_i[x_j]$ in H, if $i \neq j$, then $c_j < r_i[x_j]$. That is, every read operation reads a committed version.
- ***P4*** Let tnow be the time to execute $r_k[x_j]$. Then x_j is either *(a)* the most recently committed version before t_{now} or *(b)* the version that an active update transaction T_i whose security level is less than or equal to $L(T_k)$ reads down. In case *(a)*, $C\_TS(T_i) < C\_TS(T_j)$ or $S\_TS(T_k) < C\_TS(T_i)$. In case *(b)*, $C\_TS(T_i) < C\_TS(T_j) < S\_TS(T_k) < C\_TS(T_k)$ or $C\_TS(T_j) < S\_TS(T_k) < C\_TS(T_i)$. That is, for every $r_k[x_j]$ and $w_i[x_i]$ in H, *(a)* $C\_TS(T_i) < C\_TS(T_j)$ or *(b)* $S\_TS(T_k) < C\_TS(T_i)$.
- ***P5*** For every $r_k[x_j]$ and $w_i[x_i]$ (i, j, and k are distinct), either $f_i < r_k[xj]$ or $r_k[xj] < f_i$.
- ***P6*** For every $r_k[x_j]$ and $w_i[x_i]$ in H, $i \neq j$ and $i \neq k$, if $r_k[xj] < f_i$ and the priority of T_k is greater than that of T_i, then $C\_TS(T_k) < TS(w_i[x_i])$.
- ***P7*** For every update transaction T_i, there is a unique commit timestamp $C\_TS(T_i)$. That is, $C\_TS(T_i) = C\_TS(T_j)$ iff $i = j$.

5. Examples

In this section, we illustrate the operations of the protocol by showing two example histories produced by PSMVL protocol. We show how each transaction reads the right version to meet various requirements.

Example 1 Assume that $P_1 < P_2 < P_3$, $L_1 > L_2 > L_3$, $L(x) = L_3$, and $L(y) = L_2$. The operations of each transaction are specified as shown in Table 2.

$T_1(P_1, L_1)$: $r_1[x]$ c_1
$T_2(P_2, L_2)$: $r_2[x]$ $w_2[y]$ c_2
$T_3(P_3, L_3)$: $r_3[x]$ $w_3[x]$ c_3

	1	2	3	4	5	6
$T_1(P_1, L_1)$					$r_1[x_3]$	c_1
$T_2(P_2, L_2)$			$r_2[x_0]$	rejected		
$T_3(P_3, L_3)$	$r_3[x_0]$	$w_3[x_3]$		c_3		

Table 2. The first example.

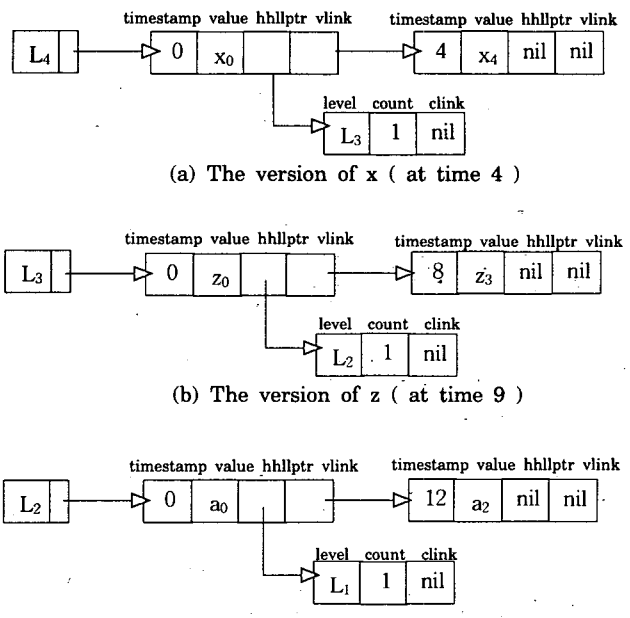


Fig. 2. A sequence of operations for the second example.

In this example, $S\_TS(T_1) = 5$, $C\_TS(T_1) = 6$, $S\_TS(T_2) = 3$, $S\_TS(T_3) = 1$, and $C\_TS(T_3) = 4$. Since $P(T_3) > P(T_2)$ and $L(T_3) < L(T_2)$, T_2 is rejected by T_3 at time 4(by the rule in Figure 1 (a)).

Example 2 Assume that $P_1 > P_2 > P_3 > P_4$, $L_1 > L_2 > L_3 > L_4$, $L(z) = L_3$, $L(x) = L_4$, and $L(a) = L_2$. The operations of each transaction are specified as shown in Table 3.

At time 3, HH-list(x) = $\{T_2, T_3\}$. At time 4, since a HH/LL-conflict between T_2 and T_4 occurs, as shown in Figure 2 (a), $x_0 \rightarrow$ *hhllptr* points to a new node which contains L_3 and a count of 1. At time 5, T_3 reads x_0 because T_3 is in HH-list(x). At time 6, HH-list(z) = $\{T_2\}$. At time 7, T_1 reads x_0 because $x_0 \rightarrow$*hhllptr* is not null and it contains lower level transaction L_3. At time 9, since HH-list(z) is not null, T_2 reads z_0 and the versions of z are as shown in Figure 2 (b). At time 10, HH-list(a) = $\{T_1\}$ and at time 11, T_1 reads a_0 because T_1 is in HH-list(a).

Ⅳ. Correctness proofs

In this section, we prove that the PSMVL protocol guarantees one-copy serializability and no priority inversion. In addition, we show that it satisfies multilevel security requirements.

1. Serializability

Theorem 1 *A multiversion schedule, H, is one-copy serializable (1SR) if and only if MVSG(H, ≪) is acyclic[6].* □

Theorem 2 *Every history produced by PSMVL is 1SR.*

$T_1(P_1, L_1)$: $r_1[x]$ $r_1[a]$ c_1 $T_2(P_2, L_2)$: $r_2[x]$ $r_2[z]$ $w_2[a]$ c_2
$T_3(P_3, L_3)$: $r_3[z]$ $r_3[x]$ $w_3[z]$ c_3 $T_4(P_4, L_4)$: $w_4[x]$ c_4

	1	2	3	4	5	6	7	8	9	10	11	12	13
$T_1(P_1, L_1)$							$r_1[x_0]$				$r_1[a_0]$		c_1
$T_2(P_2, L_2)$		$r_2[x_0]$							$r_2[z_0]$	$w_2[a_2]$		c_2	
$T_3(P_3, L_3)$	$r_3[z_0]$				$r_3[x_0]$	$w_3[z_3]$		c_3					
$T_4(P_4, L_4)$			$w_4[x_4]$	c_4									

Table 3. The second example.

Proof: Let $\{T_1, T_2, \cdots, T_n\}$ be a set of transactions, and H be a history produced by PSMVL protocol over $\{T_1, T_2, \cdots, T_n\}$. We will prove that MVSG(H, $\ll$) is acyclic by showing that every edge $T_i \rightarrow T_j$ in MVSG(H, $\ll$) is in timestamp order. We define a version order $\ll$ by $x_i \ll x_j$ only if $C\_TS(T_i)<C\_TS(T_j)$. Suppose $T_i \rightarrow T_j$ is an edge of SG(H)[4]. This edge corresponds to a reads-from relationship(i.e., for some x, T_j reads x from T_i). Then, by *PSMVL*$_3$, $C\_TS(T_i)<S\_TS(T_j)<C\_TS(T_j)$. Let $r_k[x_j]$ and $w_i[x_i]$ be in H where i, j, k are distinct, and consider the version order edge that they generate. There are two cases: (1) $x_i \ll x_j$, which implies $T_i \rightarrow T_j$ is in MVSG(H, $\ll$); and (2) $x_j \ll x_i$, which implies $T_k \rightarrow T_i$ is in MVSG(H, $\ll$). *Case (1)* by definition of $\ll$, $C\_TS(Ti)<C\_TS(Tj)$. *Case (2)* by $\mathbf{P_4}$, either $C\_TS(T_i)<C\_TS(T_j)$ or $S\_TS(T_k)<C\_TS(T_i)$. The first case is impossible, because $x_j \ll x_i$ implies $C\_TS(T_j)<C\_TS(T_i)$. Hence, it must be true that $S\_TS(T_k)<C\_TS(T_i)$. We show that $S\_TS(T_k)<C\_TS(T_i)$ ensures $T_k \rightarrow T_i$ in MVSG(H, $\ll$). There are two possible cases: (1) $P(T_k)>P(T_i)$ and (2) $P(T_k)<P(T_i)$.

In case (1), T_k starts before T_i commits, and $P(T_k)>P(T_i)$, and T_k reads the older version x_j rather than x_i. Therefore, it ensures that $T_k \rightarrow T_i$. In case (2), T_k has a lower priority and a higher security level than T_i. Thus, if T_i executes $w_i[x_i]$ before T_k commits, then T_k is aborted following the compatibility matrix in Figure 1 (a). However, there exists $r_k[x_j]$ in H. Thus, $C\_TS(T_k)<S\_TS(T_i)<C\_TS(T_i)$ and it ensures that $T_k \rightarrow T_i$. Since all edges in MVSG(H, $\leq$) are in timestamp order, MVSG(H, $\ll$) is acyclic. By Theorem 1, H is 1SR. □

2. Timing Constraints

Theorem 3 *A higher priority transaction is neither delayed nor aborted by low-priority transactions due to data contention on low-level data.*

Proof: Let T_i and T_j be two transactions such that $P(T_i)>P(T_j)$ where $P(T_i)$ and $P(T_j)$ are the priorities of T_i and T_j respectively. Let $L(T_i)$ be the security level of T_i. There are three possible cases.

The first case is where $L(T_i)>L(T_j)$. When both T_i and T_j are about to access the same data item x, T_i reads down x while T_j writes into x because of their levels. Since $P(T_i)>P(T_j)$, by the version selection algorithm, T_i reads x written by T_k (not T_j) such that $C\_TS(T_k)<S\_TS(T_i)$. Thus, T_i is neither delayed nor aborted due to T_j. The second case is where $L(T_i)=L(T_j)$. Because T_i and T_j have the same level, they should be scheduled only by a protocol for RTDBMS that avoids priority inversion. Therefore, T_i is not aborted or delayed by T_j. The last case is where $L(T_i)<L(T_j)$. T_i has a higher priority than T_j. Hence, if T_i conflicts with T_j on the same data item x, T_j is aborted by T_i using the compatibility matrix in Figure 1 (a) and (d). For all possible cases, high-priority transaction T_i precedes T_j. □

3. Security Properties

Theorem 4 *No low-level transaction is ever delayed or aborted by a high-level transaction. In addition, a low-level transaction is not interfered with due to data contention by a high-level transaction.*

Proof: By the MLS property, a transaction can read and write data items at its own level and only read down data items at lower levels. Let T_i and T_j be two transactions such that $L(T_i)>L(T_j)$ where $L(T_i)$ is the security level of T_i. If T_i and T_j are conflicting with each other, then we can see that T_i reads down the data item x while T_j writes into x. There are two possible cases.

The first case is when $P(T_i)<P(T_j)$. Because $L(T_i)$ is greater than $L(T_j)$ and $P(T_i)$ is less than $P(T_j)$, T_i is aborted or blocked according to the compatibility matrix in Figure 1 (a) and (d). Therefore, T_j is neither delayed nor aborted by T_i. The second case is when $P(T_i)>P(T_j)$. By the compatibility matrix in Figure 1 (b) and (c), T_j writes x without delaying and HH/LL-procedure is performed. Thus, T is neither delayed nor aborted by T_i. Since low-level transactions are neither delayed nor aborted, there is no security violations. □

V. Performance Evaluation

In this section, we present the simulation results to show the performance of PSMVL, compared with three other concurrency

4) A serialization graph for a history H, SG(H), is a direct graph whose nodes are transactions and whose edges represent all conflicting relationships between two transactions.

T_R \ T_H	R	W	C
R	Y	Y	Y*
W	A	Y	Y
C	—	Y	Y*

(a) Lock holder(T_H) has a lower priority

T_R \ T_H	R	W	C
R	Y	N	N
W	Y	Y	Y
C	N	N	N

(b) Lock requester(T_R) has a lower priority.

Y: yes(allowed) N: no(not allowed)
— : impossible
A: aborted(a lock holder is aborted)
Y*: yes but a certify lock converts to a write lock.

Fig. 3. The compatibility matrices for UMV2PL.

control protocols. Since the proposed protocol is based on multi-version locking, we choose the protocols that are based on locking.

The first protocol is the 2PL-HP protocol[1] for real-time databases. The 2PL-HP protocol is based on the 2PL with a priority-based conflict resolution scheme to eliminate priority inversion. The 2PL-HP maintains a single version.

The second protocol we compared with PSMVL is the Unconditional Multiversion Two Phase Locking (UMV2PL) protocol[11] for real-time databases. The UMV2PL protocol is based on MV2PL[6] and its compatibility matrix is shown in Figure 3. In UMV2PL, high priority transactions can abort low priority transactions in order to avoid priority inversion. However, when a high priority transaction requests a read or a certify lock, if a low priority transaction holds a certify lock, then the low priority transaction can convert its certify lock to a write lock for eliminating conflicts between two transactions. And thus, the UMV2PL reduces the number of the abortions of low priority transactions.

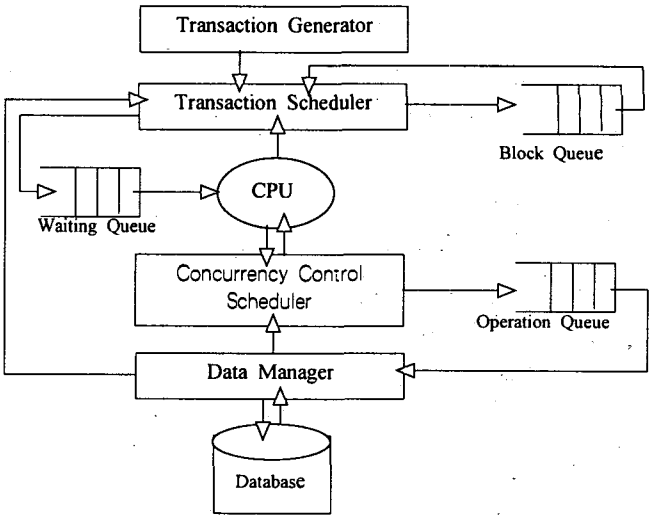
The third protocol is SRT-2PL[12] which is also based on locking and maintains two versions for each data item. As already mentioned, the SRT-2PL adopts the strict static locking scheme for the same level read and write operations of transactions while it uses the secondary copy of each of data items for read-down operations. The SRT-2PL cannot eliminate the priority inversion problem completely.

By comparing the performance of PSMVL with those protocols, the cost for satisfying security and timing requirements can be quantified.

1. Simulation Model

In order to evaluate the performance of our protocol, we use SLAM II[3] and adopt the simulation model as shown in Figure 4 (a). The parameters used in the simulation study are presented in Figure 4 (b).

We compare PSMVL with UMV2PL, 2PL-HP, and SRT-2PL in terms of the number of restart transactions, the average service time per transaction, and the fairness which shows how



(a) Simulation model

Parameter	Value
Database size	100
Slack time	10
Security levels	5
Page hit ratio	0.5
Number of data accesses per transaction	5 - 30
Disk access time	25 msec
CPU computation time	10 msec
Restart overhead	10

(b) Parameters

Fig. 4. The simulation model and parameters.

evenly the missed deadlines are spread across the input transactions of the various security levels. When a transaction is generated, it is delivered to a transaction scheduler which assigns a deadline and priority to the transaction as follows. Since we assume a soft deadline for each transaction, when a transaction misses its deadline, it is not aborted.

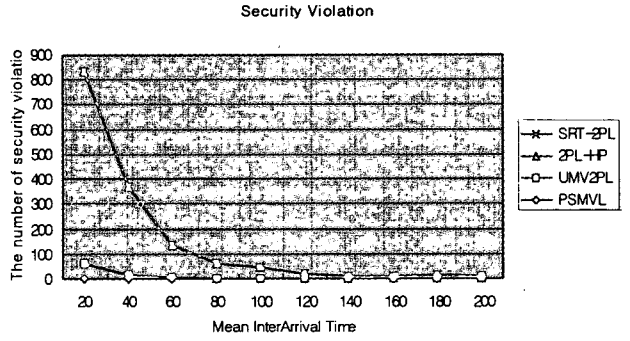
F_1 : $DeadLine(T) = ArrivalTime(T) + SlackTime * TransactionSize(T) * CPUComputationTIME$

F_2 : $Priority(T) = DeadLine(T) * 100$

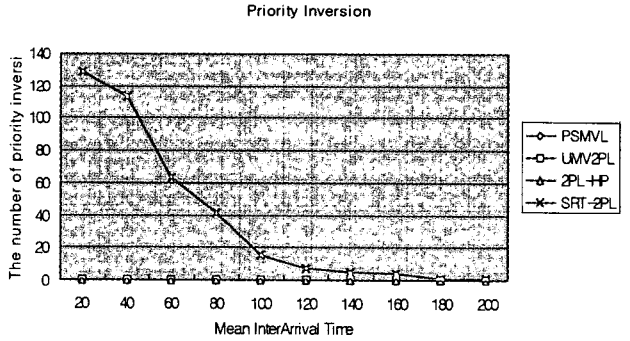
To compute the fairness, for each security level i, we use the formula,

$$F_3: \quad Fairness(i) = \frac{MissTrans_i / NoTrans_i}{MissTrans / NoTrans}$$

In the formula F_3, $MissTrans_i$ and $NoTrans_i$ are the number of transactions at level i which miss the deadlines and the number of transactions at level i, respectively, whereas *MissTrans* and *NoTrans* are the total number of transactions which miss deadlines and the total number of all input transactions, respectively. If *MissTrans* = 0, then we let *Fairness(i)* be 0.

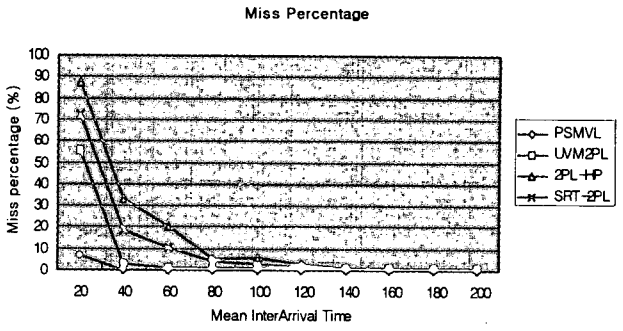


(a) Transaction size = 10. Write operation ratio = 0.7.

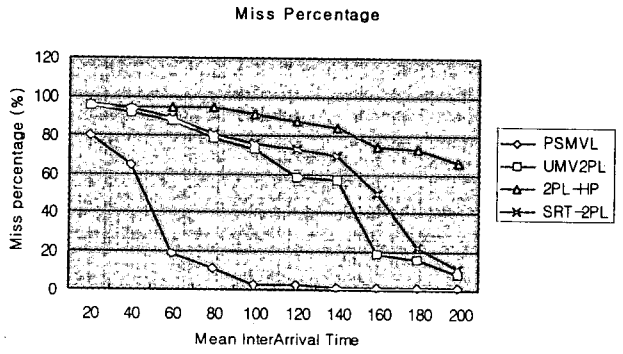


(b) Transaction size = 10. Write operation ratio = 0.7.

Fig. 5. Security violations and priority inversions.



(a) Transaction size = 10.



(b) Transaction size = 20.

Fig. 6. The miss percentage. write operation ratio = 0.7.

2. *Experimental Results*

The results of our performance analysis are shown in Figures 5, 6, 7, and 8.

In Figure 5 (a), we compare the four protocols PSMVL, UMV2PL, 2PL-HP, and SRT-2PL in terms of the number of times that low level transactions are delayed or aborted by high level transactions. The *x*-axis represents the mean interarrival time(MIAT) which is the average time interval between the generations of transactions. If MIAT is small, then transactions are created more frequently. As shown in Figure 5 (a), low level transactions are never delayed by high level transactions in both PSMVL and SRT-2PL. On the other hand, UMV2PL, which is based on multiversion, has fewer blockings than 2PL-HP which is based on single version. The 2PL-HP has the worst performance because of its 'wasted restarts'. For short transactions, the number decreases rapidly when MIAT is increased gradually, while the number decreases slowly for long transactions.

In Figure 5 (b), we compare the four protocols in terms of the number of times that high priority transactions are delayed or aborted by low priority transactions. The *y*-axis represents the number of priority inversions.

As shown in Figure 5 (b), PSMVL, 2PL-HP, and UMV2PL eliminate the priority inversion problem, while SRT-2PL does not resolve that problem completely.

Figure 6 shows the percentage of transactions that miss their deadlines, denoted by *Miss Percentage*. Miss percentage is calculated with the following equation:

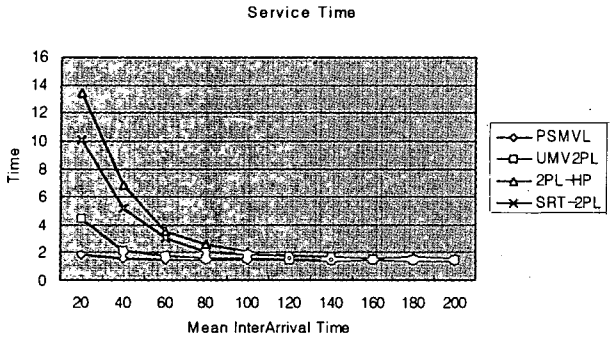
*Miss Percentage = 100 * (the number of tardy jobs / the total number of jobs)*

The number of Ns(No) in the compatibility matrix of UMV2PL is more than that in the compatibility matrix of PSMVL. This causes UMV2PL to have more restart transactions than PSMVL. This is especially true for a high arrival rate, i.e., when MIAT is small, PSMVL shows better performance than UMV2PL. Therefore, a high arrival rate increases the number of restart transactions and results in high miss percentage.

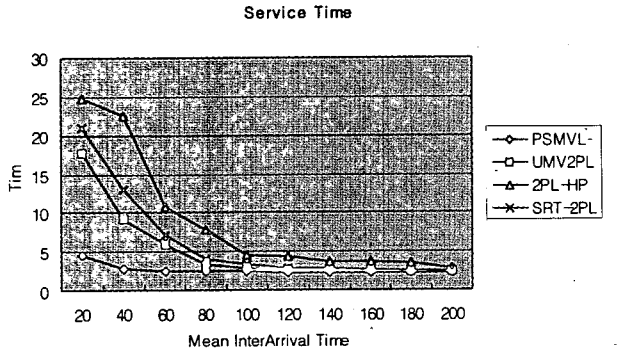
When the transactions are short and the arrival rate of transactions is low, the miss percentage is rapidly reduced. However, for long transactions, the miss percentage is reduced more slowly. This indicates that long transactions are one of the causes that increase the number of restart transactions. Since 2PL-HP uses a single-version, the number of restart transactions is higher when the transactions are scheduled using 2PL-HP, compared to UMV2PL. On the other hand, SRT-2PL adopts the basic 2PL rules and maintains two versions for each of data items. Thus, it has the fewer restart transactions than 2PL-HP but UMV2PL has the better performances than SRT-2PL.

Figure 7 shows the average service time per transaction. Let T_i be a transaction. Then, the average service time is

$$\frac{\sum_{i=1}^{N}(\text{FinishTime}(T_i)-\text{StartTime}(T_i))}{N}$$



(a) Transaction size = 10.



(b) Transaction size = 15.

Fig. 7. The average response time. write operation ratio = 0.25.

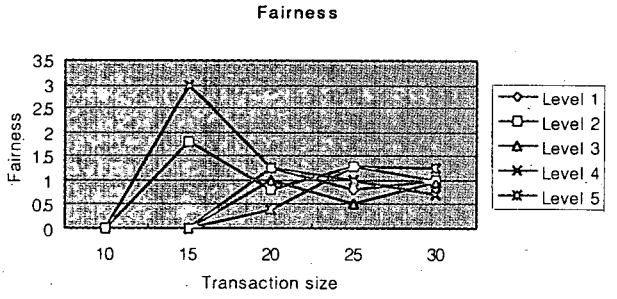


Fig. 8. The fairness of the PSMVL. Mean InterArrival Time=40.

where N is the total number of transactions. The x and y axes represent interarrival times and average service times, respectively. Since we assume a soft deadline for each transaction, when a transaction misses its deadline, it is not aborted. Instead, it continues execution until its commitment. Therefore, transactions that miss their deadlines can be restarted several times. As shown in Figure 6, when the number of restart transactions increases, it takes more time to finish the transactions. PSMVL shows better performance than UMV2PL and SRT-2PL, even though PSMVL has additional features such as security requirements. If the time interval between two transactions is short, the possibility of conflicts between transactions is increased.

Figure 8 shows the fairness of the PSMVL. In the figure, the level 5 is the highest, while the level 1 is the lowest. When the transaction size is 15, only the transactions of level 2 and the transactions of level 4 miss their deadlines. This represents that in the PSMVL, the highest level transactions are not always sacrificed. And when the transaction size becomes smaller, the number of missed deadline transactions decrease. Therefore, if the number of deadline-missing transactions is small, then the divisor of the formula F_3 is very small. As a result, for a security level i, *Fairness(i)* becomes big if the numerator of the formula F_3 is not zero. Figure 8 also shows that when the transaction size increases, for each security level i, the value of *Fairness(i)*s is getting closer. This means that the number of deadline-missing transactions is evenly distributed across the security levels and is not influenced by the security levels.

VI. Conclusion

Database systems for real-time applications must satisfy timing constraints associated with transactions. Typically a timing constraint is expressed in the form of a deadline and is represented by a priority. In this paper, we have classified transaction processing systems according to their requirements and identified the conflicting nature of security requirements and real-time requirements. To address the problem, we have presented a new priority-driven multiversion locking protocol for scheduling transactions to meet their timing constraints in real-time secure database systems. The schedules produced by the protocol were proven to be one-copy serializable. We also presented our simulation model and evaluation results of the relative performance of the protocol, compared with other protocols.

The work described in this paper can be extended in several ways. First of all, in this paper we have not considered any trade-offs between real-time requirements and security requirements. A trade-off could have been made between those two conflicting requirements, depending on the specification of the application. For example, it would be interesting to see how a policy to screen out transactions that are about to miss their deadline would affect performance. Secondly, we have restricted ourselves by not distinguishing temporal and non-temporal data management. By exploiting the semantic information of transactions and the type of data they access, the protocol could be extended to provide a higher degree of concurrency. Finally, in this paper, we have restricted ourselves to the problem of real-time secure concurrency control in a database system. There are other issues that need to be considered in designing a comprehensive MLS/RT DBMSs, including architectural issues, recovery, and data models. We have started to look into those issues.

Acknowledgement

This research has been supported by Institute of Information Telecommunication and Assessment of Korea.

References

[1] Abbott, R. K. and H. Garcia-Molina, "Scheduling Real-Time Transactions : Performance Evaluation", *Proceedings of the 14th VLDB Conference*, September 1988.

[2] Abbott, R. K. and H. Garcia-Molina, "Scheduling Real-Time Transactions : A Performance Evaluation", *ACM Transactions on Database Systems*, September 1992.

[3] Alan, A. and B. Pritsker, *Introduction to Simulation and SLAM II*, Systems Publishing Corporation, 3rd Edition, 1986.

[4] Bell, D. E. and L. J. LaPadula, "Secure Computer Systems : Mathematical Foundations", *ESD-TR-73-278*, Mitre Corporation, 1973.

[5] Bernstein, P. A. and N. Goodman, "Multiversion Concurrency Control - Theory and Algorithms", *ACM Transactions on Database Systems*, Vol. 8(No. 4), December 1983.

[6] Bernstein, P. A., V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[7] Haritsa, J. M. Carey, and M. Livny, "Data Access Scheduling in Firm Real-Time Database Systems", *Real-Time Systems Journals*, Vol. 4(No. 3), 1996.

[8] Hong, Seongsoo, "Scheduling Real-Time Programs with Static Compiler Transformations", *Journal of Electrical Engineering and Information Science*, Vol. 1(No. 3), September 1996.

[9] Keefe, T. F. and W. T. Tsai, "Multiversion Concurrency Control for Multilevel Secure Database Systems", *Proceedings of the 10th IEEE Symposium on Research in Security and Privacy*, May 1990.

[10] Keefe, T. F., W. T. Tsai, and J. Srivastava, "Multiversion Secure Database Concurrency Control", *Proceedings of the 6th International Conference on Data Engineering*, February 1990.

[11] Kim, W. and J. Srivastava, "Enhancing Real-Time DBMS Performance with Multiversion Data and Priority Base Disk Scheduling", *Proceedings of the 12th IEEE Real-Time Systems Symposium*, December 1991.

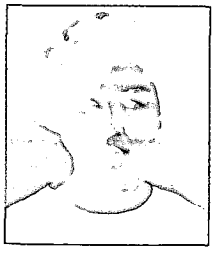
[12] Mukkamala, R. and Sang H. Son, "A Secure Concurrency Control Protocol for Real-Time Databases", *Proceedings of the 9th IFIP Working Conference on Database Security*, August 1995.

[13] Shu, L. C. and M. Young, "Correctness Criteria and Concurrency Control for Real-Time Systems : A Survey", *Technical Report SERC-TR-131-P*, November 1992.

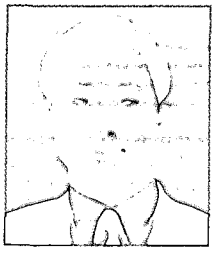
[14] Son, S. H. and B. Thuraisingham, "Towards a Multilevel Secure Database Management System for Real-Time Applications", *Proceedings of IEEE Workshop on Real-Time Applications*, May 1993.

[15] Son, S. H. and R. David, and B. Thuraisingham, "An Adaptive Policy for Improved Timeliness in Secure Database Systems", *Proceedings of the 9th IFIP Working Conference on Database Security*, August 1995.

[16] Son, S. H., S. Park, and Y. Lin, "An Integrated Real-Time Locking Protocol", *Proceedings of the 8th International Conference on Data Engineering*, February 1992.



Chan-jung Park received the B.S. degree in computer science from Sogang University in 1988 and the M.S. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1990. From 1990 to 1994, she worked at Korea Telecom as a technical staff. Since 1994, she has been a Ph.D. student in computer science department, Sogang University. Her major research areas are database security, real-time systems, and transaction processing systems.



Seog Park received the B.S. degree in computer science from Seoul National University in 1978, the M.S. and the Ph.D. degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1980 and 1983, respectively. Since 1983, he has been working in the Department of Computer Science of the College of Engineering, Sogang University. His major research areas are real-time systems, database security, data warehouse, digital library, and multimedia database systems. Dr. Park is a member of ACM, the IEEE Computer Society and the Korea Information Science Society.