

# Multiversion Locking Protocol with Freezing for Secure Real-Time Database Systems

Chanjung Park, *Member, IEEE Computer Society*,  
Seog Park, *Member, IEEE Computer Society*, and  
Sang H. Son, *Senior Member, IEEE*

**Abstract**—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 as a priority to be used by schedulers. Recently, security has become another important issue in many real-time applications. In many systems, sensitive information is shared by multiple users with different levels of security clearance. As more advanced database systems are being used in applications that need to support timeliness while managing sensitive information, there is an urgent need to develop protocols for concurrency control in transaction management that satisfy both timing and security requirements. In this paper, we propose a new multiversion concurrency control protocol that ensures that both security and real-time requirements are met. The proposed protocol is primarily based on locking. However, in order to satisfy timing constraints and security requirements, a new method, called the freezing method, is proposed. In order to show that our protocol provides a higher degree of concurrency than existing multiversion protocols, we define a new serializability for multiversion concurrency control, called FR-serializability, which is more general than traditional serializability. We present several examples to illustrate the behavior of our protocol, along with performance comparisons with other protocols. The simulation results show significant performance improvement of the new protocol.

**Index Terms**—Locking protocol, multiversion concurrency control, multilevel security, real-time database systems, transaction management.

## 1 INTRODUCTION

A real-time database system (RT/DBS) extends the set of correctness requirements from a conventional database system. In order to be correct, an RT/DBS must satisfy timing constraints associated with transactions which are typically expressed in terms of deadlines as well as data consistency [9], [27]. The time criticalness (priority) of a transaction is usually derived from both its timeliness requirement and its importance. In most cases, the priority assigned to a transaction is directly related to the deadline of the transaction and a transaction scheduler is to eliminate priority inversions<sup>1</sup> [9], [33]. Recently, security becomes another important requirement in many real-time applications because many systems maintain sensitive information to be shared by multiple users with different levels of security clearances.

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

- C. Park is with the Department of Computer Education, Cheju National University, 1 Ara-dong Jeju-si Jeju-do, 690-756, Korea. E-mail: park@educ.com.cheju.ac.kr.
- S. Park is with the Department of Computer Science, Sogang University, 1 Sinsu-dong Mapo-gu Seoul, 121-742, Korea. E-mail: spark@dblab.sogang.ac.kr.
- S.H. Son is with the Department of Science, University of Virginia, PO Box 400740, Charlottesville, VA. E-mail: son@cs.virginia.edu.

Manuscript received 27 Apr. 1998; revised 9 Apr. 2001; accepted 11 Apr. 2001.

For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 106721.

On the other hand, a multilevel secure database system (MLS/DBS) is a system which is shared by users with more than one clearance level and which contains data of more than one classification level [4], [17], [18]. In order for an MLS/DBS to be correct, it must satisfy two more requirements besides serializability. One is the elimination of covert channels and the other is the prevention of the starvations of high-level transactions. The unauthorized information flows in an MLS database are called covert channels. In order to prevent covert channels, low-level transactions should be neither delayed nor aborted by high-level transactions [11], [15]. In principle, MLS technique can be used for any system that contains sensitive data [29]. If an RT/DBS needs to maintain secure information, it can be a good example area for applying the MLS technique. As more advanced database systems are being used in applications that need to support security requirements as well as timeliness, we must integrate the MLS technique into an RT/DBS, namely Secure Real-Time Database System (SRT/DBS) [28], [30]. For SRT/DBSs, the concurrency control protocols in transaction management that satisfy both timing and security requirements need to be developed.

In this paper, we propose a new concurrency control protocol for SRT/DBSs which ensures one-copy serializability [7] and eliminates covert channels caused by data contention. In addition, the proposed protocol ensures that high-priority transactions are not blocked by low-priority transactions. This allows satisfaction of timing constraints and eliminates starvation of high-level transactions. Our protocol is based on multiversion locking scheme, but in

TABLE 1  
The Characteristics of Three DBSs

Type Features	RT/DBS	MLS/DBS	SRT/DBS
Data characteristics	– Temporal attributes – Sensor input	– Classification level	– Temporal and non-temporal – Classification level
Transaction characteristics	– Periodic – Event-driven – Priority-driven	– Clearance level	– Priority-driven – Clearance level
Requirements	– Serializability – No priority inversion	– Serializability – No covert channel	– Serializability – No priority inversion – No covert channel
Applications	– Command control system – Plant monitoring system	– Military information system – Database of insurance company	– Surveillance satellite system – Electronic commerce – Computerized auction system

order to meet the various requirements, it freezes the states of data objects so that each transaction can choose appropriate versions of the data items. Protocols based on multiversions require more storage than those based on a single version. However, the proposed protocol is based on multiversions for the following reasons. Concurrency control protocols that maintain a single version of each data item, such as 2PL-HP [1] and OPT-sacrifice [14], cannot avoid the starvation of low-priority transactions because low-priority transactions can be delayed or aborted by high-priority transactions. The protocols that maintain two versions for each data item can partly resolve the above starvation problems. However, due to the limitation on the number of versions, the protocols cannot help sacrificing one of the requirements. Thus, a multiversion scheme is considered appropriate to satisfy all the requirements for SRT/DBSs.

Next, we propose a new serializability, called *FR-serializability*, for multiversion concurrency control protocols. Based on *FR-serializability*, we show that our protocol has better performance than existing protocols based on multiversions. We then prove that if a multiversion (MV) history which indicates an interleaved execution order of transactions ensures *FR-serializability*, then it also guarantees one-copy serializability.<sup>2</sup> In order to prove that a multiversion concurrency control protocol works correctly, we must show that the MV histories produced by the protocol are one-copy serializable. Typically, existing multiversion protocols such as *Multiversion Two Phase Locking (MV2PL)* [6] or *Multiversion Timestamp Ordering (MVTO)* [7] have been developed on the basis of the definition of *1-serial* (or *one-copy serial*). *1-serial* is a sufficient condition for one-copy serializability. According to the definition of *1-serial*, each transaction is permitted to read the most recent versions of data items, whereas in the definition of *FR-serializability*, somewhat older versions can be read by any transaction. If a concurrency control protocol provides older data versions to transactions, we can obtain more serializable histories than the protocols that are based on

the definition of *1-serial*. As a result, a higher degree of concurrency can be achieved by *FR-serializability*.

The remainder of this paper is organized as follows: In Section 2, we describe the characteristics of RT/DBSs, MLS/DBSs, and SRT/DBSs, respectively. We then summarize the related works on SRT/DBSs. Some background on correctness criteria for the proposed protocol is presented in Section 3. In Section 4, after we define some terminologies that are used in the proposed protocol, we present a new concurrency control protocol based on locking for SRT/DBSs. We then provide some examples to illustrate how transactions are scheduled by our protocol. In Section 5, we propose a new serializability, called *FR-serializability*, for multiversion concurrency control. After giving the correctness proof of the proposed serializability, we show that the proposed serializability provides a higher degree of concurrency than previous protocols based on multiversions. We prove that our protocol ensures serializability, no covert channels and no priority inversion. After the performance results of our protocol are presented in Section 6, we conclude the paper in Section 7.

## 2 RELATED WORK AND APPLICATION AREAS

In principle, MLS/DBSs can be used for any system that contains sensitive data [29]. There is an increasing need for supporting applications that have timing constraints when managing sensitive data in advanced database systems. To support such applications, we must integrate MLS techniques into RT/DBSs, resulting in SRT/DBSs [30]. Since SRT/DBSs need to support both multilevel security and real-time requirements, it is easy to see that protocols for SRT/DBSs will be more complicated than those for MLS/DBSs or those for RT/DBSs. The three database systems have similarities and differences and their characteristics are described in Table 1.

There has been much research on concurrency control protocols for MLS/DBSs [3], [16], [18], [21], [23], [24], [25] and for RT/DBSs [1], [19], [27], [28]. In case of SRT/DBSs, several protocols have been proposed. The *Secure Real-Time*

2. One-copy serializability is defined in detail in Section 3.

*Two Phase Locking* (SRT-2PL) protocol [22] was proposed but the priority inversion problem can still occur. On the other hand, in [30], security requirements can be compromised by real-time requirements, and vice versa. In [11], security requirements are treated as additional correctness criteria for database systems, while real-time requirements are regarded just as performance issues. In their approach, to satisfy security requirements, a high-priority is always assigned to a low-level transaction. If that kind of priority assignment rule is adopted, security and real-time requirements can be guaranteed easily. This is reasonable in some situations, but a more general approach needs to consider the cases in which high-level transactions have an arbitrary priority. For SRT/DBSs, both security and real-time requirements should be considered as correctness criteria. We need a new solution to satisfy both requirements. An adaptive protocol with performance results which illustrate the benefit of using adaptive approach was proposed in [31]. In [31], the security factor and the deadline-miss factor are used to resolve the conflicts between security and timing requirements. In [32], the trade offs between timeliness and security are addressed and then a scheme which allows partial violations of security for improved timeliness has been proposed. A multiversion concurrency control algorithm for SRT/DBSs has been proposed in [13]. In the paper, transactions are classified into two types in order to reduce the number of deadline missing transactions. The proposed protocol in [13] satisfies the timing and security requirements.

As time-critical applications continue to evolve, real-time database systems become more complex and they need to support other properties such as fault-tolerance, security, and availability. Unfortunately, it is not easy to support nonorthogonal requirements. Some of the research issues are discussed in [33].

Not all applications have to run on multilevel systems. However, the applications we consider in this paper have some data classified at levels beyond the clearances of some users. Thus, we concentrated on the MLS requirements and the real-time requirements. For non-MLS environments, many research works have been proposed [8], [26]. One of them is role-based access control (RBAC) models which are receiving increasing attention as a generalized approach to access control. In [8], the RBAC model is extended in order to support temporal constraints on role activations, while [26] proposes a unified model for RBAC.

Another area which requires both real-time and security properties is information warfare (IW). IW, also known as I-War, C4I,<sup>3</sup> and Cyberwar, has recently become of increasing importance to the military, the intelligence community, and the business world. IW is the offensive and defensive use of information and information system to exploit, corrupt, or destroy an adversary's information and information systems, while protecting one's own systems [12]. Thus, in order to protect ourselves from attacks, real-time and security features should be considered as well as cryptography, biometrics, digital signatures, and firewalls. The technical issues for IW are beyond the scope of our paper. The surveys on tools and techniques can be seen in [10].

3. C4I means Command, Control, Communications, Computers, and Intelligence.

### 3 CORRECTNESS CRITERIA

In this section, we briefly review the definition of one-copy serializability in [7], which is the correctness criterion for multiversion concurrency control protocols. We then summarize the MLS properties which should be provided by the protocols for SRT/DBSs.

#### 3.1 One-Copy Serializability

The objectives of concurrency control in DBSs are to avoid inconsistent retrievals and to preserve the correct state of the databases. *Serializability* is the definition of correctness for concurrency control in database systems. To describe serializability, we introduce some definitions presented in [7].

**Definition 1 [7].** A multiversion (MV) history  $H$  is serial if for any two transactions,  $T_i$  and  $T_j$ , that appear in  $H$ , either all of  $T_i$ 's operations precede all of  $T_j$ 's or vice versa.

**Definition 2 [7].** A serial MV history  $H$  is 1-serial (or one-copy serial) if for all  $i, j$ , and some data item  $x$ , if  $T_i$  reads the value of  $x$  created by  $T_j$ , then  $i = j$ , or  $T_j$  is the last transaction preceding  $T_i$  that writes into any version of  $x$ .

**Definition 3 [7].** An MV history  $H$  is one-copy serializable (or 1SR) if its committed projection,  $C(H)$ , is equivalent to a 1-serial MV history, where  $C(H)$  is the history obtained from  $H$  by deleting all operations that do not belong to committed transactions in  $H$ .

#### 3.2 Multilevel Security Property

There are three MLS properties: *value*, *delay*, and *recovery* secure properties [17]. Three properties represent a way of preventing a covert channel in a secure system. For their definitions, *purge* function is needed. Given a schedule  $H$  and a security level  $L$ ,  $\text{purge}(H, L)$  is the function that removes all operations from  $H$ , whose level is greater than  $L$  [17].

**Definition 4 [17].** For an input schedule  $H$ , the output schedule  $H'$  is value secure if  $\text{purge}(H', L)$  is view equivalent to the output schedule produced for  $\text{purge}(H, L)$ . For an input schedule  $H$  and an output schedule  $H'$ , a schedule is delay secure if for each level  $L$  in  $H$ , any operation  $o_i$  in  $\text{purge}(H, L)$  is delayed in the output schedule produced for  $\text{purge}(H, L)$  if and only if it is delayed in  $\text{purge}(H', L)$ . A schedule is recovery secure if a set of transactions  $T$  at level  $L$  is in a deadlock state when every transaction in  $T$  is waiting for an event that can only be caused by other transactions in  $T$ .

To preserve value security, a secure scheduler must ensure that a low-level transaction always reads the same version of a data item irrespective of the existence of high-level transactions which access the same data item. A secure scheduler enforces delay security by ensuring that low-level transactions are always delayed by transactions at the same level only, not by high-level transactions. Recovery security means that a secure scheduler must ensure that a deadlock does not occur across different levels of transactions.

### 4 A NEW PROTOCOL WITH FREEZING

In this section, we present a new protocol based on multiversion locking scheme with freezing method for SRT/DBSs. Since an SRT/DBS needs a security model in

TABLE 2  
Possible Priorities and Security Levels for  $T_i$  and  $T_j$

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	$P_{High}$	$L_{High}$	$P_{Low}$	$L_{Low}$

order to enforce its security policy of the system, we adopt the Bell-LaPadula (BLP) model [5] as our security model. The BLP model imposes the following properties on all data accesses. Let  $L(T)$  be the security level of a transaction  $T$  and  $L(x)$  be the security level of a data item  $x$ .

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

Before we present our protocol, let us examine the conflicts between timing requirements and security requirements. Let  $T_i$  and  $T_j$  be arbitrary 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. There are three possible cases for the priorities of these transactions:

1.  $P(T_i) = P(T_j)$ ,
2.  $P(T_i) > P(T_j)$ , and
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 than or equal to  $P(T_j)$ . In addition, there are three possibilities for the security levels of these transactions:

1.  $L(T_i) = L(T_j)$ ,
2.  $L(T_i) > L(T_j)$ , and
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  $L_{High} > L_{Low}$ .

Table 2 shows all possible combinations of priorities and security levels between  $T_i$  and  $T_j$ . In Table 2,  $L_{Eq}$  is used when two transactions have the same security level, while  $P_{Eq}$  is used when two transactions have the same priority. In the first case, the priorities and security levels of the two transactions are the same. Therefore, the only concern is in ensuring serializability. Security requirements and timing constraints can be ignored in this case. In the second and third cases, since the priorities of the two transactions are the same, timing requirements can be ignored. Thus, the low-level transaction should not be delayed by the high-level transaction. In these cases, any protocol for MLS/DBSs

can be used. In the fourth case, the security levels of the two transactions are the same. Hence, the transactions must be scheduled so that they meet the timing constraints as well as ensuring logical consistency. In this case, any protocol for RT/DBSs 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 be neither delayed nor aborted by  $T_j$ , so that covert channels are avoided. Both requirements can be satisfied by having  $T_i$  precede  $T_j$ . The approach proposed by [11] can be applied to this case. 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,  $T_j$  must not be blocked by  $T_i$  so that covert channels are avoided. We call this kind of conflict *Security and Priority conflict (SP-conflict)*. An SP-conflict is a conflict between a high-level transaction with high-priority and a low-level transaction with low-priority. Ideally,  $T_i$  should precede  $T_j$  because of their priorities. We resolve the SP-conflicts by using the proposed protocol.

#### 4.1 Terminologies and Rules

The proposed protocol is based on MV2PL protocol [6]. MV2PL relaxes the rule of 2PL so that the conflicts between read and write or write and write are eliminated. Hence, each data item may have many versions that are created by active transactions, called *uncertified* versions. However, transactions can read only the most recently certified version in order to meet one-copy serializability. A *certify* lock is used to delay the commitment of a transaction until there is no active reader of data items that are about to be overwritten. Certify locks only conflict with read locks and other certify locks. However, in order to resolve SP-conflicts, the conflicts between read and certify locks are permitted in the proposed protocol.

To present our protocol, we define some terminologies. We assume that each transaction  $T$  has a read set,  $RS(T)$ , and a write set,  $WS(T)$ , when  $T$  starts its execution.

**Definition 5.** For any active transactions  $T_i$  and  $T_j$  ( $i \neq j$ ),  $T_i$  is directly frozen by  $T_j$  if 1)  $L(T_i) > L(T_j)$  and 2) when  $T_i$  holds a read lock on some data item  $x$ ,  $T_j$  does a certify operation on  $x$  or when  $T_j$  holds a certify lock on  $x$ ,  $T_i$  holds a read lock on  $x$ .

**Definition 6.** For any active transactions  $T_i$  and  $T_j$  ( $i \neq j$ ),  $T_i$  is indirectly frozen by  $T_j$  if

1.  $L(T_i) \geq L(T_j)$ ,
2.  $RS(T_i) \cap RS(T_j) \neq \phi$ , and
3.  $RS(T_i) \cap WS(T_j) \neq \phi$ , and
4.  $T_j$  is directly or indirectly frozen.

**Definition 7.** The set of frozen points of  $T_i$ , denoted by  $fp(T_i)$ , is a set of pairs (time, TID)s, where time represents the point that  $T_i$  is directly or indirectly frozen and TID is a transaction identifier. If  $T_i$  is directly frozen by some transaction  $T_j$ , then a (certifying time of  $T_j$ , 0) is inserted into  $fp(T_i)$ . If  $T_i$  is indirectly frozen by some transaction  $T_j$ , then  $fp(T_j)$  is not empty by Definition 6. For each (time, TID) in  $fp(T_j)$ , (time,  $u(TID)$ ) is inserted into  $fp(T_i)$ , where  $u(TID)$  is the function such that if  $TID = 0$ , then  $u(TID) = j$ ,<sup>4</sup> otherwise,  $u(TID) = TID$ . In this case, TID cannot be  $j$ . If  $T_i$  is

4.  $j$  is the transaction identifier of  $T_j$ .

indirectly frozen by  $T_j$ , then  $T_j$  is also either a directly or indirectly frozen transaction, and  $T_j$  cannot be frozen by itself.

When a transaction  $T_2$  is directly frozen by  $T_1$  and  $T_3$  is indirectly frozen by  $T_2$ , a newly created pair (the certifying time of  $T_1$ , zero) is added to  $fp(T_2)$ , and  $T_3$  inherits the frozen points of  $T_2$ . For example, let  $fp(T_2)$  be  $\{(3, 0)\}$ . Then,  $fp(T_3)$  is set to  $\{(3, 2)\}$ . If a transaction has no read-down operation, then the set of frozen points of the transaction is empty. The rules for our protocol based on the notion of frozen points are presented below. Basically, our protocol is based on MV2PL and for our protocol, we assume that there is a shared clock which has the lowest security level. Transactions can read the clock but cannot update it.

1.  $FR_1$ . For each data item  $x$ , a timestamp  $ts(x)$  is assigned when it is certified.
2.  $FR_2$ . For each transaction  $T$ , its read set,  $RS(T)$ , and its write set,  $WS(T)$ , are declared when it starts.
3.  $FR_3$ . The conflicts between the same level and the same priority transactions are resolved by the rules of MV2PL. When a transaction  $T$  reads a data item  $x$  at the same level,  $T$  reads the most recently certified version of  $x$ . According to our security model, a transaction can carry out write operations only on data items at the same level. Since our protocol is based on multiversions,  $T$  can write data items at any time. When  $T$  certifies, if there are either read or certify locks held by other transactions on the same data items, then  $T$  must wait until the other transactions release their locks; otherwise,  $T$  can certify.
4.  $FR_4$ . When transactions read, write, or certify data items at the same level as themselves, if a conflicting operation occurs between  $T$  and a high-priority transaction, then  $T$  should be blocked so as not to cause priority inversion.
5.  $FR_5$ . When  $T$  reads a low-level data item  $x$ , if a high-priority transaction  $T_{high}$  holds a certify lock on  $x$ , then, by our security model, the security level of  $T_{high}$  is equal to the security level of  $x$ . Therefore,  $T$  should be blocked to satisfy the timing and security requirements.
6.  $FR_6$ . When  $T$  reads a low-level data item  $x$ , if no transaction holds a certify lock on  $x$ , then  $T$  can acquire a read lock on  $x$  without delay. Otherwise, since  $T$  and  $T_{low}$  can be neither aborted nor delayed (SP-conflict), the following operations are performed:
  - a. If  $T$  has an empty  $fp(T)$ , then without waiting for the commitment of  $T_{low}$ ,  $T$  reads the most recent version of  $x$  that was certified before the read operation of  $T$  on  $x$ .  $T$  is then directly frozen by  $T_{low}$ . The tuple (certifying time of  $T_{low}$ , 0) is inserted into  $fp(T)$ .
  - b. Otherwise (not empty),  $T$  reads the most recently certified version before time  $t$ , where  $t$  is the element of  $(t, TID) \in fp(T)$  such that for all  $(t_i, TID_i) \in fp(T)$ , there is no  $(t_i, TID_i)$  with  $t_i < t$ .
7.  $FR_7$ . When  $T$  certifies on  $x$ , if there exist transactions  $T_h$  that hold read locks on  $x$ , then the security level  $L(T_h)$  of each  $T_h$  is higher than or equal to  $L(T)$ . By our security model,  $L(T) = L(x)$ .

- a. If  $L(T_h) = L(T)$ , then  $T$  is scheduled according to the priorities of the two transactions, i.e., if  $P(T) > P(T_h)$ , then  $T_h$  is blocked; otherwise,  $T$  is blocked by  $T_h$ .
- b. If  $L(T_h) > L(T)$ , then check if  $P(T)$  is higher than  $P(T_h)$ . If  $P(T) > P(T_h)$ , then  $T_h$  is blocked for security and priority reasons. However, if  $P(T) < P(T_h)$ , then an SP-conflict occurs. In that case, rule  $FR_6$  is applied.

Next three rules describe how the  $fp$  of a transaction is affected and how values are assigned by other directly or indirectly frozen transactions at a lower level or the same level.

8.  $FR_8$ . By Definition 7, when  $T_i$  holds a read lock on  $x$ , if a low-level transaction  $T_j$  tries to certify on  $x$  and a SP-conflict occurs, then  $T_i$  is directly frozen by  $T_j$ . The tuple (certifying time of  $T_j$ , 0) is inserted into  $fp(T_i)$ .
9.  $FR_9$ . During the execution of a transaction  $T_i$ , if  $T_j$  is recently and directly frozen by  $T_k$ ,  $L(T_i) \geq L(T_j)$ , and  $RS(T_i) \cap RS(T_j) \neq \phi$ , then (certifying time of  $T_k$ ,  $j$ ) is inserted into  $fp(T_i)$ . This means that  $T_i$  is indirectly frozen by  $T_j$ .
10.  $FR_{10}$ . When  $T_i$  is recently generated,  $T_i$  checks if any low-level frozen transaction  $T_j$  exists. If there is no such transaction, then  $fp(T_i)$  is set to empty; otherwise, all the elements in  $fp(T_j)$ , where  $T_j$  is the transaction that has the highest level lower than  $T_i$  among all frozen transactions, are copied into  $fp(T_i)$  in the manner described in Definition 7.

Let  $H$  be a history produced by our protocol. Then,  $H$  must satisfy the following properties. For transactions which have the same security level and the same priority, the properties of MV2PL are also applied to our protocol. For transactions having different levels and different priorities, our proposed protocol has additional properties. Let  $cr_i$  be the certification of  $T_i$ .

1.  $P_1$ . For every  $T_i$ ,  $cr_i$  follows all of  $T_i$ 's reads and writes and precedes  $T_i$ 's commitment.
2.  $P_2$ . For every  $w_k[x_k]$  and  $r_k[x_j]$  in  $H$ , if  $w_k[x_k] < r_k[x_j]$ , then  $j = k$  and  $L(T_k) = L(x)$ .
3.  $P_3$ . If  $r_k[x_j]$  and  $w_i[x_i]$  are in  $H$ , then either  $cr_i < r_k[x_j]$  or  $r_k[x_j] < cr_i$ . This means that  $r_k[x_j]$  is strictly ordered with respect to the certification operation of every transaction that writes  $x$ .
4.  $P_4$ . For every  $r_k[x_j]$  and  $w_i[x_i]$  ( $i, j$ , and  $k$  distinct), if  $cr_i < r_k[x_j]$  and  $L(T_i) = L(T_k)$ , then  $cr_i < cr_j$ . This means that if transactions are at the same level, each reads either a version written by itself or the most recently certified version of  $x$ .
5.  $P_5$ . For every  $r_k[x_j]$  in  $H$ , if  $j \neq k$ , then  $cr_j$  precedes  $r_k[x_j]$  and  $L(T_k) \geq L(x)$ . Otherwise,  $w_k[x_k]$  precedes  $r_k[x_k]$  and  $L(x) = L(T_k)$ . This means that each transaction reads a certified version of each data item. In case of read operations at the same level, the transaction can also read a version written by itself. However, when a high-level transaction reads low-level data, it always reads a certified version of the data.
6.  $P_6$ . For every  $w_i[x_i]$  and  $w_j[x_j]$ , either  $cr_i < cr_j$  or  $cr_j < cr_i$ . If  $L(T_k)$  equals to  $L(T_i)$ , then this property

can easily be proven by the properties of MV2PL. The case where  $L(T_k) < L(T_i)$  never occurs due to our security model. Let us consider the case in which  $L(T_k) > L(T_i)$ . In order to prevent covert channels, the low-level transaction  $T_i$  can certify its operations even though  $T_k$  holds a read lock. However, since  $T_k$  becomes a frozen transaction,  $T_k$  does not read any data item that is created after the certification of  $T_i$ . In addition, there is no write conflict between  $T_k$  and  $T_i$  because of our security model. Therefore,  $T_k$  precedes  $T_i$  in serialization order.

The pseudocode for the proposed protocol is summarized in Fig. 1.

## 4.2 Examples

In this section, we examine two example histories generated by our protocol and describe how transactions select versions of data items.

**Example 1.** The first example is shown in Fig. 2. Let  $T_i$  be a transaction,  $L_i$  be the security level of  $T_i$ , and  $P_i$  be the priority of  $T_i$ . Assume that  $L_4 < L_3 < L_2 < L_1$ ,  $P_4 > P_3 > P_2 > P_1$ ,  $L(x) = L_4$ ,  $L(y) = L_3$ , and  $L(z) = L_2$ .

The MV history in Fig. 2 cannot be generated by MV2PL protocol because when  $T_4$  requests a certify lock on  $x$  at time 3,  $T_4$  is blocked by  $T_2$  which holds a read lock on the same data item  $x$ . For security and priority reasons,  $T_2$  and  $T_4$  should not be blocked. In the proposed protocol,  $T_4$  is not blocked. At time 3,  $T_2$  is directly frozen by  $T_4$  and  $fp(T_2) = \{(3, 0)\}$ . When  $T_1$  starts at 8, the scheduler checks if  $RS(T_1) \cap RS(T_2) = \phi$ . Since there is no common data item in  $RS(T_1)$  and  $RS(T_2)$ ,  $fp(T_1)$  is set to  $\phi$ . Therefore,  $T_1$  reads the most recent version of  $y$  and  $z$ , i.e.,  $y_3$  and  $z_2$ .

**Example 2.** Consider another example in Fig. 3. Assume that  $L_5 < L_4 < L_3 < L_2 < L_1$ ,  $P_5 > P_4 > P_3 > P_2 > P_1$ ,  $L(x) = L_4$ ,  $L(y) = L_5$ , and  $L(z) = L_3$ .

The example history in Fig. 3 cannot be generated by MV2PL because of the properties of MV2PL, i.e.,  $T_4$  is blocked by  $T_3$  at time 3. However, according to our protocol,  $fp(T_3) = \{(3, 0)\}$  at time 3. At time 4,  $T_2$  starts with  $fp(T_2) = \{(3, 3)\}$  because  $T_2$  is indirectly frozen by  $T_3$  and the elements in  $fp(T_3)$  are copied to  $fp(T_2)$  according to the rule  $FR_{10}$ . At time 7,  $T_2$  and  $T_3$  are directly frozen by  $T_5$ . Therefore,  $fp(T_3)$  includes  $\{(3, 0), (7, 0)\}$  and  $fp(T_2)$  includes  $\{(3, 3), (7, 0)\}$ . When  $T_1$  starts at time 8, it is affected by  $T_3$  because

$$RS(T_1) \cap RS(T_3) \neq \phi.$$

Therefore,  $fp(T_1)$  includes  $\{(3, 3), (7, 2)\}$  by  $FR_{10}$  and  $T_1$  reads  $x_0$  which is created before time 3. At 11, since  $T_3$  commits,  $fp(T_1) = \{(7, 2)\}$ . Thus,  $T_1$  reads  $z_3$ . This history is equivalent to  $T_2 T_3 T_1 T_4 T_5$ .

## 5 CORRECTNESS PROOFS

In this section, we propose a new serializability, called *FR-serializability*, for multiversion concurrency control, and discuss its correctness. Since 1-serial condition is sufficient for one-copy serializability, there are many 1SR MV

histories that are not 1-serial. Let us consider the following serial MV histories  $H_1$ ,  $H_2$ , and  $H_3$  over transactions  $T_1$ ,  $T_2$ , and  $T_3$ . In the histories,  $c_i$  represents the commitment of  $T_i$ ,  $r_i[x_j]$  represents the read operation of  $T_i$  reading the version  $x_j$  of  $x$ , and  $w_i[x_i]$  denotes  $T_i$  writes a version  $x_i$  of  $x$ .

$$\begin{aligned} H_1 &= w_1[x_1]w_1[y_1]c_1r_2[x_1]r_2[y_1]w_2[x_2]w_2[y_2]c_2r_3[x_2]r_3[y_2]c_3 \\ H_2 &= w_1[x_1]w_1[y_1]c_1r_2[x_1]r_2[y_1]w_2[x_2]w_2[y_2]c_2r_3[x_1]r_3[y_1]c_3 \\ H_3 &= w_1[x_1]w_1[y_1]c_1r_2[x_1]r_2[y_1]w_2[x_2]w_2[y_2]c_2r_3[x_1]r_3[y_2]c_3. \end{aligned}$$

$H_1$  is a 1-serial MV history, while  $H_2$  and  $H_3$  are not 1-serial because  $T_3$  in  $H_2$  and  $T_3$  in  $H_3$  do not read the most recent versions. However, since  $H_2$  is equivalent to a serial SV history  $T_1T_3T_2$ ,  $H_1$  and  $H_2$  are 1SR and  $H_3$  is not 1SR. In the above three MV histories, we can notify that the versions of the data items which are read by a transaction have some relationships. For example, in order to ensure 1SR, when  $T_3$  chooses  $x_1$ ,  $T_3$  reads  $y_1$  not  $y_2$ , while  $T_3$  reads  $y_2$  when  $T_3$  chooses  $x_2$ .

Now, we define one possible relationship among the versions of the data items which are read by a transaction. For the following definitions, we assume that each transaction  $T$  has a read set,  $RS(T)$ , which contains the data items read by  $T$  and a write set,  $WS(T)$ , which contains the data items written by  $T$ . We also assume that a data item  $x$  has a timestamp when it is created. For data items  $x$  and  $y$ , if  $x$  is created more recently than  $y$ , then  $x$  has a greater timestamp than  $y$ .

**Definition 8.** For any two transactions  $T_i$  and  $T_j$  in a serial MV history  $H$ , we say  $T_i < T_j$  when  $T_i$  precedes  $T_j$  in  $H$ .

**Definition 9.** For a serial MV history  $H$  over a set of transactions,  $T$ , a set of affecting previous transactions of  $T_i$  in  $T$ ,

$$\begin{aligned} APT(T_i) &= \{T_j \mid T_j \in T, T_j < T_i, RS(T_j) \cap RS(T_i) \neq \phi, \\ &\text{and } RS(T_j) \cap WS(T_i) \neq \phi\}. \end{aligned}$$

In history  $H_1$ ,  $T_1 < T_2 < T_3$ .  $APT(T_i)$  contains the transaction,  $T_j$ , which precedes  $T_i$  and its read set has a common data item with both the read set and the write set of  $T_i$ .  $T_i$  is affected by one of the transactions in  $APT(T_i)$  when it selects the appropriate versions of data items to read. To illustrate the use of  $APT$ , let us consider the following history  $H_4$ .

$$\begin{aligned} H_4 &= w_0[x_0]w_0[y_0]c_0w_1[x_1]c_1 \\ &\quad r_2[x_1]r_2[y_0]c_2 \quad r_3[x_2]w_3[y_3]c_3 \quad r_4[x_2]r_4[y_2]c_4. \end{aligned}$$

In history  $H_4$ ,  $APT(T_3) = \{T_2\}$ , since  $T_2 < T_3$  and  $RS(T_2) \cap RS(T_3) = \{x\}$  and  $RS(T_2) \cap WS(T_3) = \{y\}$ . Thus, the versions of the data items read by  $T_3$  are affected by those read by  $T_2$ .

**Definition 10.** A serial MV history  $H$  is FR-serial, if for all  $i, j$ , and some data item  $x$ , a transaction  $T_i$  reads  $x$  from  $T_j$  (i.e.,  $r_i[x_j]$  is in  $H$ ),  $r_i[x_j]$  is the first read operation of  $T_i$ , and  $T_i$  and  $T_j$  satisfy the following conditions (see also Fig. 4):

1. If  $APT(T_i) = \phi$ , then  $T_j$  is either  $T_i$  or any transaction, which creates a version of  $x$  before  $T_i$  starts. Otherwise (i.e.,  $APT(T_i) \neq \phi$ ), we define  $T_p$  as the transaction that creates a data item  $y$ , where  $y$  has

```

01 /* Let  $x$  be a data item and  $T, T'$  be transactions. In addition, let R, W, and C be read, write, and
02 certification operations, respectively. */
03 case 1:  $T$  requests a read operation on  $x$ .
04     if ( $x$  is locked with  $W$  or  $C$ ) then
05         if (any lock-holder has a higher or equal priority) then
06             if (any lock-holder has a lower or equal security level) then
07                 Block lock-requester; Return ;
08             end if
09         else /* lock holder has a low-priority */
10             Let  $T'$  be the transaction that has low-priority than that of  $T$ ;
11             if ( $L(T')$  is lower than  $L(T)$ ) then
12                 Insert a frozen point (the current time, 0) into  $fp(T)$ ; /* directly frozen */
13             end if
14             else if ( $L(T')$  is the same as  $L(T)$ ) then
15                 for (all  $T'$  which have  $C$  on  $x$ ) do
16                     Convert the lock from  $C$  to  $W$ ;
17                 end do
18             end if
19         end if
20     end if
21     Grant a read lock to  $T$ ;
22 case 2:  $T$  requests a write operation on  $x$ .
23     /* there exists no conflict between write and other operations. */
24     Grant a write lock to  $T$ ;
25 case 3:  $T$  requests a certify operation on  $x$ .
26     Let  $T'$  be any transaction that already has a read or a certify on  $x$ ;
27     if ( $P(T') \geq P(T)$ ) then
28         if ( $L(T') = L(T)$ ) then
29             the request is rejected and blocked;
30         else if ( $L(T') > L(T)$ ) then
31             Freeze the active readers ;
32             Grant the certify lock to  $T_i$  ;
33         end if
34     else /*  $P(T') < P(T)$  */
35         if ( $L(T') = L(T)$ ) then
36             if ( $T'$  holds a certify lock) then
37                 for (all  $T'$  which have  $C$  on  $x$ ) do
38                     Convert the lock from  $C$  to  $W$ ;
39                 end do
40             end if
41         else if ( $L(T') > L(T)$ ) then Abort  $T'$ ;
42     end if
43     Grant  $C$  on  $x$ ;

```

Fig. 1. Pseudocode for the proposed protocol.

the greatest timestamp among all data items in  $RS(T_k)$ ,  $T_k \in APT(T_i)$ . If  $y$  and  $x$  are not the same data item, then let  $T_q$  be the transaction that creates the last version of  $x$  before  $T_p$  starts. Otherwise (i.e.,  $y$  and  $x$  are the same data item), let  $T_q$  be  $T_p$ . Then,  $T_j$  is either 1)  $T_i$ , 2)  $T_q$ , or 3) any transaction which creates  $x$  between  $T_q$  and  $T_i$ .

2. The rest of the read operation of  $T_i$  select, appropriate versions as follows: If there exists a transaction which creates a version of  $x$  after  $T_j$  finishes, then  $T_i$  reads the latest version created before  $T_k$  starts, where  $T_k$  creates  $x$  after  $T_j$  and there is no  $T_s$  which creates  $x$  and  $T_j < T_s < T_k$ . Otherwise, each read operation of  $T_i$  selects the version written by itself or the most recent version created before  $T_i$  starts.

Time Transaction	1	2	3	4	5	6	7	8	9	10	11
$T_1$								$r_1[y_3]$		$r_1[z_2]$	$cr_1$
$T_2$	$r_2[x_0]$					$r_2[z_0]$	$w_2[z_2]$		$cr_2$		
$T_3$				$w_3[y_3]$	$cr_3$						
$T_4$		$w_4[x_4]$	$cr_4$								

Fig. 2. First example history.

Time Transaction	1	2	3	4	5	6	7	8	9	10	11	12	13
$T_1$								$r_1[x_0]$				$r_1[z_3]$	$cr_1$
$T_2$				$r_2[y_0]$					$cr_2$				
$T_3$		$r_3[x_0]$				$r_3[y_0]$				$w_3[z_3]$	$cr_3$		
$T_4$	$w_4[x_4]$		$cr_4$										
$T_5$					$w_5[y_5]$		$cr_5$						

Fig. 3. Second example history.

In other words, when a transaction  $T$  executes its first read operation on a data item  $x$ ,  $T$  checks whether  $APT(T)$  is empty or not. If  $APT(T)$  is empty, then  $T$  can read any version of  $x$  created either by the preceding transactions of  $T$  or by itself. Otherwise,  $T$  cannot read an arbitrary version of  $x$ . In that case, the valid versions of  $x$  that  $T$  reads are decided by timestamps. When a version of  $x$  chosen by the first read operation of a transaction is determined, the versions chosen by the rest of the read operations of the transaction should be created prior to the next version of  $x$ .

**Definition 11.** An MV history  $H$  is FR-serializable, if its committed projection,  $C(H)$ , is equivalent to an FR-serial MV history.

FR is the abbreviation for "First-Read operation dependent." FR-serial eliminates the limitation that transactions must read the most recent versions of data items. It provides a set of versions a transaction can read. Let us consider the history  $H_4$  once more. Since  $APT(T_3)$  is not empty,  $y$  becomes  $x_1$ ,  $T_p$  and  $T_q$  are the same, and  $T_q$  becomes  $T_1$ . Therefore,  $T_3$  should read a version of  $x$  which is either created by  $T_1$  or created after  $T_1$ . In the history,  $T_3$  should choose  $x_1$  for reading in order to ensure FR-serial. Let us examine which versions of data items  $T_4$  should read. Since  $APT(T_4)$  is empty,  $T_4$  can read  $x_0$  as well as  $x_1$ . However, by Definition 10, if  $T_4$  reads  $x_0$ ,  $y_0$  should be chosen and if  $T_4$  reads  $x_1$ , then  $y_3$  should be read. We can refine the definition of '1-serial' in terms of 'FR-serial' as follows: An MV history over a transaction set  $T$  is 1-serial, if for each transaction in  $T$  and some data item  $x$ , the first read operation of a transaction selects the most recent version of  $x$  or the version written by itself, and each of the rest read operations of the transaction selects the most recent versions created before the next version of  $x$ . In the following, we prove that each FR-serial MV history has an equivalent 1-serial MV history and then, if an MV history is FR-serializable, the history is also 1SR.

**Lemma 1.** Every FR-serial MV history has no cycle.

**Proof.** We must show that for an FR-serial MV history  $H$  over  $T = \{T_1, T_2, \dots, T_n\}$  and a version order  $\ll$ ,  $MVSG(H, \ll)$ <sup>5</sup> has no cycle. Assume that  $T_i < T_j$  in  $H$  and there is a cycle between  $T_i$  and  $T_j$ . Then, 1) for an edge  $T_i \rightarrow T_j$ ,  $w_i[x_i] < r_j[x_i]$  or  $w_i[x_i] < r_j[x_k]$  for some  $x$ , where  $x_i \ll x_k$  and 2) for an edge  $T_j \rightarrow T_i$ ,  $w_j[y_j] < r_i[y_j]$  or  $w_j[y_j] < r_i[y_k]$  for some  $y$ , where  $y_j \ll y_k$ . Hence, four cases are possible:

5. A version order  $\ll$  of each data item  $x$  for an MV history is a total order on the committed versions of  $x$ . Given an MV history  $H$  and the version order  $\ll$  for  $H$ , the multiversion serialization graph for  $H$  and  $\ll$ ,  $MVSG(H, \ll)$ , is a graph whose nodes are committed transactions in  $H$ , and whose edges have two types, the conflicting edge and the version order edge. A conflicting edge from  $T_i$  to  $T_j$  exists, if  $C(H)$  contains the operations  $w_i[x_i]$  and  $r_j[x_i]$  for some  $x$ . For any  $r_k[x_i]$  and  $w_i[x_i]$ , where  $i, j, k$  are distinct, the version order edge from  $T_i$  to  $T_j$  exists, if  $x_i \ll x_j$ . Otherwise, a version order edge from  $T_k$  to  $T_i$  occurs [7].

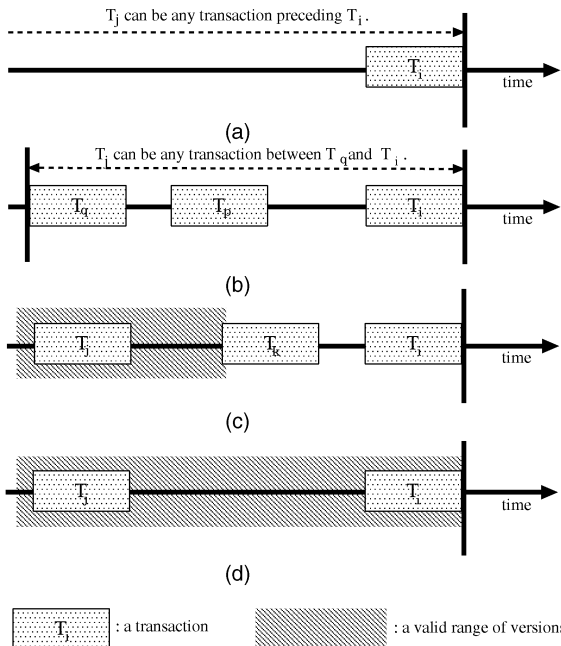


Fig. 4. FR-serializability. (a)  $APT(T_i)$  is  $\phi$ . (b)  $APT(T_i)$  is not  $\phi$ . (c) The valid range of versions  $T_i$  can read when  $T_k$  exists. (d) The valid range of versions  $T_i$  can read when  $T_k$  does not exist.

1.  $w_i[x_i]r_i[y_j]c_i r_j[x_i]w_j[y_i]c_j$ ,
2.  $w_i[x_i]r_i[y_k]c_i r_j[x_i]w_j[y_j]c_j$ ,  $y_j \ll y_k$ ,
3.  $w_i[x_i]r_i[y_j]c_i r_j[x_k]w_j[y_j]c_j$ ,  $x_i \ll x_k$ , and
4.  $w_i[x_i]r_i[y_k]c_i r_j[x_k]w_j[y_j]c_j$ ,  $x_i \ll x_k$ , and  $y_j \ll y_k$ .

In the first case, by Definition 10, when  $T_i$  reads  $y$ ,  $T_i$  must select either a version of  $y$  created before  $T_i$  starts or the version written by itself. However, since  $y_j$  is created after  $T_i$  commits, this case is impossible. In the second case,  $T_i$  reads  $y_k$ , which is created after  $T_j$  commits. This case also violates the definition of FR-serial and, hence, is also impossible. In the third case,  $T_k$  exists and creates  $x$  after  $T_i$ . Thus,  $T_j$  reads for each data item the most recent version before  $T_k$  starts. However,  $T_i$  reads  $y_j$  which is created after  $T_i$  commits. This case is also impossible. In the fourth case, the version order of  $x$  is  $T_i < T_k < T_j$ , while the version order of  $y$  is as  $T_i < T_j < T_k$ . There exists no  $T_k$ , which can satisfy the above version ordering. Therefore, the above four cases are impossible and the contradiction is proved. Next, we assume that  $T_{i_1}, T_{i_2}, \dots, T_{i_n}$  make a cycle. Then, since  $H$  is a serial history, some transaction reads a version that is not created yet for the same reason as in the above case. This violates the definition of FR-serial. Therefore, an FR-serial history has no cycle.  $\square$

**Lemma 2.** *Every FR-serial MV history has an equivalent serial SV history.*

**Proof.** By Lemma 1, for an FR-serial MV history  $H$ ,  $MVSG(H, \ll)$  has no cycle. Thus, there exists a topological ordering of  $MVSG(H, \ll)$ . Let  $H_s$  be a serial MV history  $T_{i_1} T_{i_2} \dots T_{i_n}$ , where  $T_{i_1} T_{i_2}, \dots, T_{i_n}$  is a topological sort of  $MVSG(H, \ll)$ . Then, we must show that  $H_s$  is 1-serial. This means that the topologically sorted history of  $H$  has the same read-from relationships as a 1-serial history. Our proof is similar to the one presented in [7]. Consider any reads-from relationship in  $H_s$ , say  $T_k$  reads  $x$  from  $T_j$ ,  $k \neq j$ . Let  $w_i[x_i](i \neq j \text{ and } i \neq k)$  be any other write on  $x$ . If  $x_i \ll x_j$ , then  $MVSG(H, \ll)$  includes the version order edge  $T_i \rightarrow T_j$ , which makes  $T_i$  precede  $T_j$  in  $H_s$ . If  $x_j \ll x_i$ , then  $MVSG(H, \ll)$  includes the version order edge  $T_k \rightarrow T_i$ , which makes  $T_k$  precede  $T_i$  in  $H_s$ . Thus, no transaction that writes  $x$  falls in between  $T_j$  and  $T_i$  in  $H_s$ . Therefore,  $H_s$  is 1-serial and any FR-serial MV history has an equivalent 1-serial MV history.  $\square$

**Theorem 1 [7].** *The committed projection  $C(H)$  of an MV history  $H$  is equivalent to a serial SV history if  $H$  is 1SR.*

**Theorem 2.** *If an MV history  $H$  is FR-serializable, then  $H$  is also 1SR.*

**Proof.** By Lemma 2, any FR-serial MV history has a 1-serial MV history. Let the 1-serial MV history be  $H_s$ . By the definition of 1-serial,  $H_s$  has an equivalent serial SV history  $H'_s$ . Since  $H$  is equivalent to  $H_s$  and  $H_s$  is equivalent to  $H'_s$ ,  $H$  is equivalent to  $H'_s$  by the transitivity of equivalence.  $H$  has an equivalent serial SV history and  $H = C(H)$ , because  $H$  is a serial history. Therefore,  $H$  is 1SR by Theorem 1.  $\square$

Finally, we prove that the proposed protocol guarantees one-copy serializability, ensures no priority inversion, and eliminates covert channels.

**Theorem 3.** *Every schedule produced by the proposed protocol is 1SR.*

**Proof.** The proof of the FR-serializability of our protocol is similar to Theorem 3. We prove that  $MVSG(H, \ll)$  is acyclic by showing that every edge  $T_i \rightarrow T_j$  in  $MVSG(H, \ll)$  is in certification order: if  $T_i \rightarrow T_j$ , then  $cr_i < cr_j$ . Suppose  $T_i \rightarrow T_j$  is an edge of the serialization graph,  $SG(H)$ , of an MV history  $H$  which is a direct graph whose nodes are transactions and whose edges represent all conflicting relationships between two transactions. This edge corresponds to a reads-from relationship (i.e., for some  $x$ ,  $T_j$  reads  $x$  from  $T_i$ ). Then, since all transactions read certified versions of data items,  $cr_i < cr_j$ .

Next, let  $w_i[x_i]$ ,  $w_j[x_j]$ , and  $r_k[x_k]$  be operations in  $H$ , where  $i, j, k$  are distinct, and consider the version order edge that they generate. There are two cases: 1)  $x_i \ll x_j$  and 2)  $x_j \ll x_i$ . The first case implies  $T_i \rightarrow T_j$  is in  $MVSG(H, \ll)$ . Thus, by definition of  $\ll$ ,  $cr_i < cr_j$ . The second case implies  $T_k \rightarrow T_i$  is in  $MVSG(H, \ll)$ . Then, by  $P_4$ , either  $cr_i < cr_j$  or  $cr_k < cr_i$ . The first option is impossible because the definition of  $\ll$  requires  $cr_j < cr_i$ . Therefore,  $cr_k < cr_i$ . Since all edges in  $MVSG(H, \ll)$  are in certification order,  $MVSG(H, \ll)$  is acyclic. Thus,  $H$  is FR-serializable and by Theorem 2,  $H$  is 1SR.  $\square$

**Theorem 4.** *Low-level transactions are neither delayed nor aborted by high-level transactions. In addition, low-level transactions are not affected by data contention with high-level transactions.*

**Proof.** From our security model, a transaction can read and write data items at its own level and can only read down data items at lower levels. Let  $T_i$  and  $T_j$  be two transactions such that  $L(T_i) > L(T_j)$ . If  $T_i$  and  $T_j$  conflict with each other, it is the situation where  $T_i$  reads down a data item  $x$  while  $T_j$  certifies on  $x$ . There are two possible cases: 1)  $P(T_i) < P(T_j)$  and 2)  $P(T_i) > P(T_j)$ . In the first case, since  $L(T_i) > L(T_j)$  and  $P(T_i) < P(T_j)$ ,  $T_j$  should not be delayed to ensure security and real-time requirements. In the second case, since  $L(T_i) > L(T_j)$  and  $P(T_i) > P(T_j)$ , an *SP-conflict* occurs. In order to prevent covert channels,  $T_i$  and  $T_j$  should not be blocked. In our protocol,  $T_i$  is frozen by  $T_j$ . Therefore,  $T_j$  is neither delayed nor aborted by  $T_i$ . These cases show that the proposed protocol satisfies security requirements.  $\square$

**Theorem 5.** *A high-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 conflicting transactions on the same data item  $x$  such that  $P(T_i) > P(T_j)$ . Then, there are two 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  $x$ , the levels of the transactions show that  $T_j$  reads down  $x$  while  $T_i$  writes into  $x$ . According to the level of  $T_j$  and the priority of  $T_j$ , the high-priority transaction  $T_i$  blocks  $T_j$  to avoid covert channels and priority inversion. The second case is where  $L(T_i) > L(T_j)$ , i.e., an *SP-conflict* occurs. Because of their levels, we can see that  $T_i$  reads  $x$  while  $T_j$  certifies  $x$ . Since  $T_i$  is frozen by  $T_j$ , high-priority

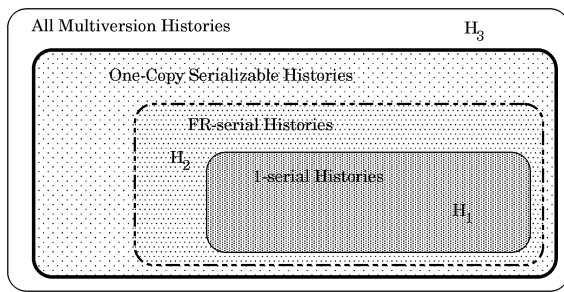


Fig. 5. Relationships.

transaction  $T_i$  is not blocked. Instead,  $T_i$  reads some prior version which was certified before  $T_j$  certifies. For all possible cases, high-priority transaction  $T_i$  precedes  $T_j$ .  $\square$

Fig. 5 shows the relationships between FR-serial and 1-serial MV histories. The histories  $H_1$ ,  $H_2$ , and  $H_3$  in the Fig. 5 are presented at the beginning of this Section. Fig. 5 represents that if a concurrency control protocol ensures FR-serializability, then it generates one-copy serializable MV histories which cannot be produced by the existing protocols based on multiversions. Thus, such a protocol will produce FR-serializable MV histories and will also provide a higher degree of concurrency and better performance than existing protocols based on multiversions.

## 6 PERFORMANCE EVALUATION

In this section, we present simulation results to show the performance of the proposed protocol, called FREEZE, compared with four other concurrency control protocols. Since FREEZE is based on multiversion locking, we compare it with protocols which are based on locking. By comparing the performance of FREEZE with those protocols, the cost of satisfying security and timing requirements can be quantified. The four protocols used for comparison are as the following.

The first protocol is MV2PL [6]. The second protocol is the 2PL-HP protocol [1] for RT/DBSs. The 2PL-HP protocol is based on the 2PL protocol with a priority-based conflict resolution scheme to eliminate priority inversion. The 2PL-HP maintains a single version. The third protocol is the Unconditional Multiversion Two Phase Locking (UMV2PL) protocol [19] for real-time databases. The UMV2PL protocol is based on MV2PL [7] and 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, a low-priority transaction which holds a certify lock can convert that lock to a write lock, so as to eliminate conflicts between the two transactions. The UMV2PL thus reduces the number of abortions of low-priority transactions. The fourth protocol is SRT-2PL [22] which is also based on locking and maintains two versions for each data item. As already mentioned, the SRT-2PL adopts a strict static locking scheme for the same level read and write operations, while it uses the secondary copy of each data item for read-down operations. The SRT-2PL cannot eliminate the priority inversions completely.

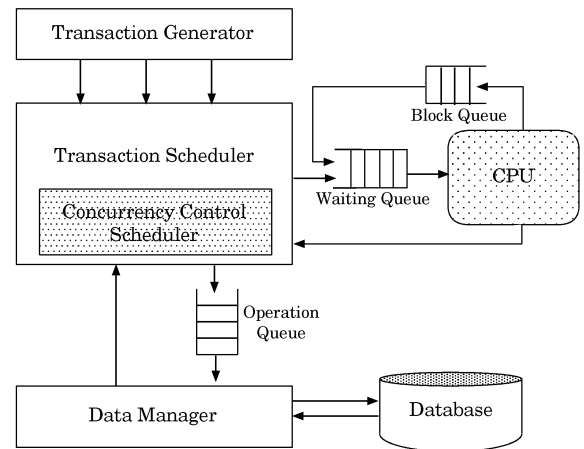


Fig. 6. Simulation model.

### 6.1 Simulation Model

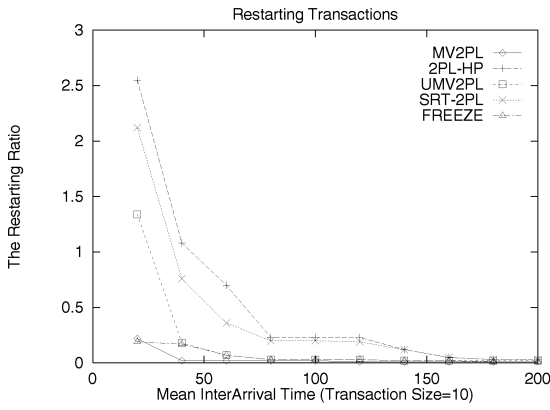
In order to evaluate the performance of our protocol, we use SLAM II [2] to implement the database model shown in Fig. 6. The parameters used in the simulation study are presented in Table 3. We set the parameters based on the values commonly used in database simulation studies [20].

We compare the proposed protocol with MV2PL, 2PL-HP, UMV2PL, and SRT-2PL in terms of the number of restarted transactions, the average service time per transaction, and fairness which shows how evenly missed deadlines are spread across the input transactions of the various security levels. When a transaction is generated, it is delivered to the transaction scheduler which assigns a deadline and a priority to the transaction according to the formulas  $F_1$  and  $F_2$ . Since we assume a soft deadline for each transaction, when a transaction misses its deadline, it is not aborted.

In our paper, a higher number presents a higher priority. Also, the priority of a transaction is inversely proportional to its deadline. Formula  $F_2$  represents that. To show how many transactions are restarted, we use formula  $F_3$  [20]. To compute the average service time per transaction  $T_i$ , formula  $F_4$  is used. To show the percentage of transactions that miss their deadlines, denoted by *Miss Percentage*, we use formula  $F_5$  [20]. Finally, in order to compute the fairness

TABLE 3  
Parameters

Parameter	Value
Database Size	100
Slack Time	10
Number of Security Levels	4
Page Hit Ratio	0.5
Number of Operations per Transaction	5 ~ 30
Disk Access Time	25 msec
CPU Computation Time	10 msec
Restarting Overhead	10


 Fig. 7. Restart transactions ( $Transaction\ Size = 10$ ).

for each security level  $i$ , we use formula  $F_6$  [11]. In formula  $F_6$ ,  $MissTrans_i$  and  $NoTrans_i$  are the number of transactions at level  $i$  which miss the deadlines and the total number of transactions at level  $i$ , respectively, whereas  $MissTrans$  and  $NoTrans$  are the total number of transactions that miss deadlines and the total number of all transactions, respectively. If  $MissTrans = 0$ , then we set Fairness( $i$ ) to 0.

1.  $F_1$ .

$$Deadline(T) = ArrivalTime(T) + SlackTime * TransactionSize(T) * CPUComputationTime.$$

2.  $F_2$ .

$$Priority(T) = VeryLargeNumber - Deadline(T) * 100.$$

3.  $F_3$ .

$$Restarting\ Ratio = \frac{Number\ of\ Restarting\ Transactions}{Total\ Number\ of\ Transactions}.$$

4.  $F_4$ .

$$Average\ Service\ Time = \frac{(\sum_{i=1}^N FinishTime(T_i) - StartTime(T_i))}{N},$$

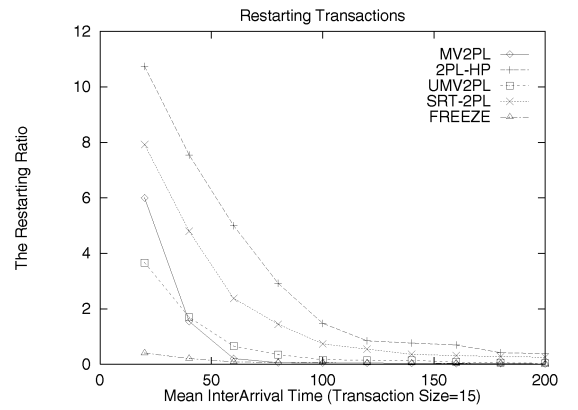
where  $N$  is the total number of transactions.

5.  $F_5$ .

$$Miss\ Percentage = 100 * \frac{Number\ of\ Tardy\ Jobs}{Total\ Number\ of\ Jobs}.$$

6.  $F_6$ .

$$Fairness(i) = \frac{MissTrans_i / NoTrans_i}{MissTrans / NoTrans}.$$

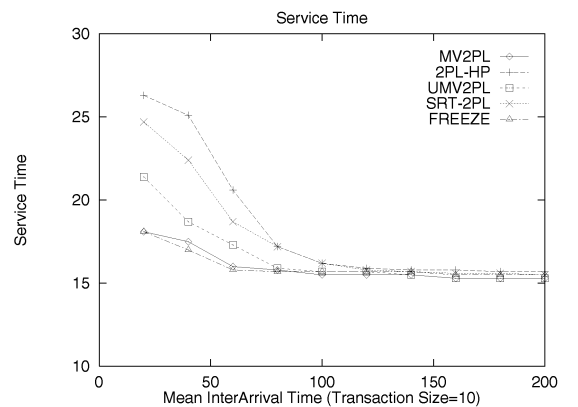

 Fig. 8. Restart transactions ( $Transaction\ Size = 15$ ).

## 6.2 Performance Analysis

The results of our performance evaluation are shown in the figures below. As already mentioned, we compare the FREEZE with MV2PL, 2PL-HP, UMV2PL, and SRT-2PL in terms of the number of restarted transactions, the average service time per transaction, and fairness, which shows how evenly missed deadlines are spread across various security levels.

Figs. 7 and 8 show the restarting ratio of transactions. The restarting ratio highly depends on the number of versions. 2PL-HP maintains a single version per data item and if there exists a data contention with a high-priority transaction, the low-priority transaction is always aborted. Thus, it shows the worst performance when mean interarrival time (MIAT) is small. Since SRT-2PL provides two copies per data item, it shows a better performance than 2PL-HP but it is worse than the protocols based on multiversions. Among the protocols based on multiversions, since our protocol adopts a freezing method instead of abortion-based method, it has the fewest number of restarting transactions. However, if transaction size becomes bigger, MV2PL suffers from deadlocks. In MV2PL, some transactions can be in a deadlock state which does not occur in 2PL. Therefore, when MIAT is very small, deadlocks degrades the performance of MV2PL.

Figs. 9 and 10 show the average service time per transaction. The  $x$  and  $y$  axes represent interarrival times


 Fig. 9. Service time ( $Transaction\ Size = 10$ ).

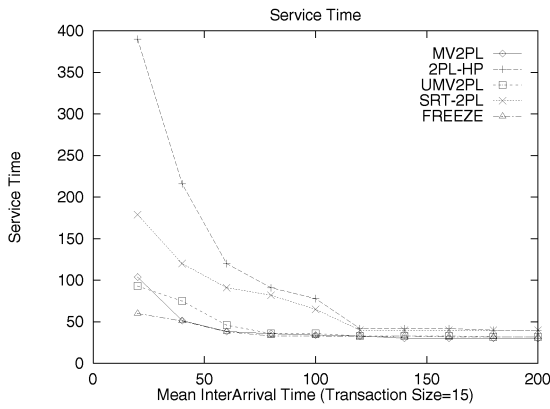


Fig. 10. Service time (*Transaction Size = 15*).

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 may be restarted several times. As shown in Figs. 7 and 8, when the number of restart transactions increases, it takes more time to finish the transactions. Compared with other protocols, the proposed protocol FREEZE is affected less than others and thus, it shows better performance when the MIAT is small. However, when MIAT becomes large, the service time of FREEZE is larger than those of MV2PL or UMV2PL because it has additional features to support both security requirements and real-time requirements.

Figs. 11 and 12 show the percentage of transactions that miss their deadlines, denoted by *Miss Percentage*. We use formula  $F_4$  to calculate the miss percentages. Even though 2PL-HP was developed for real-time database systems, the number of Ns (which represents not compatible) in the compatibility matrix of 2PL-HP is higher than other protocols. Thus, 2PL-HP shows the worst performance among all protocols. Meanwhile, among the protocols based on multiversions, UMV2PL has a better performance than MV2PL or SRT-2PL, since it has been proposed for real-time database systems. The number of Ns in the compatibility matrix of UMV2PL is higher than that in the compatibility matrix of FREEZE. This causes UMV2PL to have more

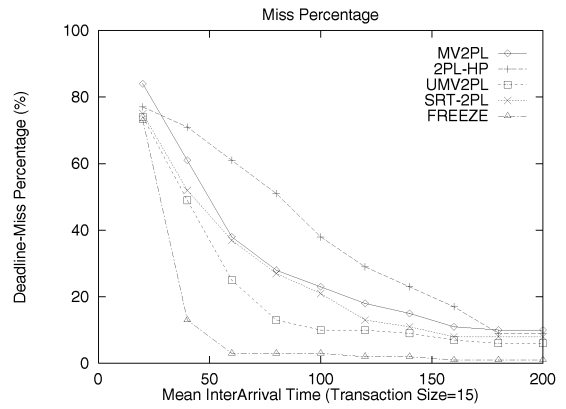


Fig. 12. Miss percentages (*Transaction Size = 15*).

restart transactions than FREEZE. This is especially true for a high arrival rate. When the mean interarrival time is small, FREEZE shows a better performance than UMV2PL. Therefore, a high arrival rate increases the number of restart transactions and results in a high miss percentage.

When the transactions are short and the arrival rate of the transactions is low, the miss percentage is greatly reduced. However, for long transactions, the miss percentage is higher. This indicates that long transactions contribute to the increase in the number of restart transactions. Since 2PL-HP uses a single version of data, the number of restart transactions is higher with 2PL-HP than with UMV2PL. On the other hand, since SRT-2PL adopts the basic 2PL rules and maintains two versions for each data item, it has the fewer number of restart transactions than 2PL-HP. However, because of the number of versions maintained by protocols, UMV2PL has the fewer restart transactions than SRT-2PL.

Fig. 13 shows the fairness of the proposed protocol. In the figure, security level 4 is the highest, while level 1 is the lowest. When the transaction size is 15, only the transactions of levels 2 and 4 miss their deadlines. This is because in the proposed protocol, the highest level transactions are not always sacrificed. When the transaction size becomes smaller, the number of missed deadline transactions decreases. If the number of transactions missing their deadlines is small, then the divisor in formula  $F_3$  is very

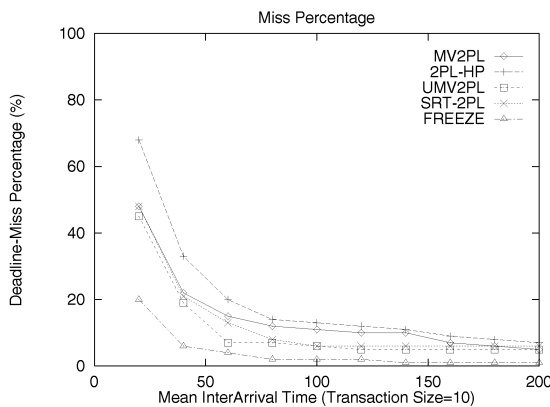


Fig. 11. Miss percentages (*Transaction Size = 10*).

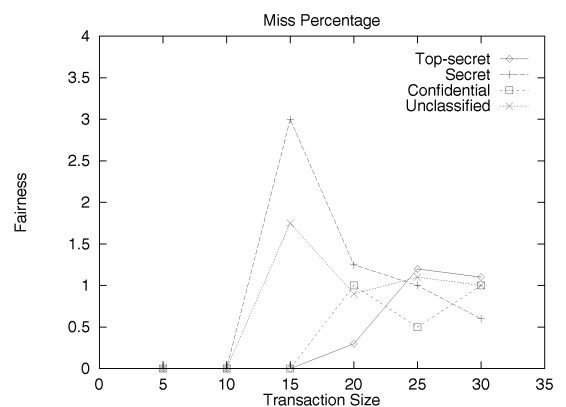


Fig. 13. Fairness of the proposed protocol (*MIAT = 40*).

small. As a result, for security level  $i$ ,  $Fairness(i)$  becomes large if the numerator of the formula  $F_3$  is not zero. Fig. 12 also shows that as the transaction size increases, the fairness values for the different security levels approach each other. This means that the number of transactions missing their deadlines is evenly distributed across the security levels and is not influenced by the security level.

## 7 CONCLUSIONS

An SRT/DBS must ensure the logical consistency of the database, security requirements, and timing constraints. In this paper, we have proposed a new concurrency control protocol for SRT/DBSs. It freezes the time so that each transaction chooses the appropriate versions of data items. It also ensures one-copy serializability and eliminates both covert channels and the starvation of high-level transactions. In addition, there is no priority inversion. In 1-serial MV histories, transactions should read the most recent version of each data item. However, there are many one-copy serializable schedules which are not 1-serial. Therefore, in this paper, we have defined a sufficient condition for one-copy serializability, called FR-serial, which is more general than 1-serial. and proposed FR-serializability. Our protocol ensures FR-serializability. It sometimes provides a data version which is not the most recent one. This feature helps avoid abortion of high-level transactions. Therefore, it can also resolve the starvation of high-level transactions. We have also presented a simulation model and an evaluation of the relative performance of the protocol compared with other protocols. The simulation study shows that even though the proposed protocol has additional features such as satisfying security and real-time requirements, it provides reasonable performance compared with four other protocols.

The work described in this paper can be extended in several ways. First, in the experiments, we have focused on the number of restarting transaction. The protocols used by our experiments have compatibility matrices and they schedule transactions according to the matrices. When comparing their matrices, our protocol provides the fewest number of denials to transactions when they request locks. As a result, our protocol shows the best performance among all protocols. However, in our protocol, high-level transactions may read stale values and the staleness is traded for real-time performance and security. There is no attempt made to quantify or limit the staleness in our simulation study. In the future, we plan to examine the staleness by performing more simulation studies. In addition, the mechanism to limit the number of data copies that could be created as the result of freezing should be developed. Second, in this paper, we have not considered any trade offs between real-time and security requirements, although these may be possible, depending on the specification of the application. For example, it would be interesting to see how a policy of screening out transactions that are about to miss their deadline would affect performance. Third, we have restricted ourselves by not distinguishing temporal and nontemporal data management. By exploiting the semantic information in 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 SRT/DBS, including architectural issues, recovery, and data models. We have started to look into those issues.

## ACKNOWLEDGMENTS

This research was supported in part by the Brain Korea 21 Project in 2001.

## REFERENCES

- [1] R.K. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," *ACM Trans. Database Systems*, vol. 17, no. 3, pp. 513-560, Sept. 1992.
- [2] A. Alan and B. Pritsker, *Introduction to Simulation and SLAM II* third ed., Indiana: Systems Publishing, 1986.
- [3] V. Atluri, E. Bertino, and S. Jajodia, "Providing Different Degrees of Recency Options to Transactions in Multilevel Secure Databases," *Proc. IFIP WG11.3 Ninth Ann. Working Conf. Database Security*, pp. 255-273, Aug. 1995.
- [4] V. Atluri, S. Jajodia, T.F. Keefe, C. McCollum, and R. Mukkamala, "Multilevel Secure Transaction Processing: Status and Prospects," *Proc. IFIP WG11.3 10th Ann. Working Conf. Database Security*, pp. 79-98, July 1996.
- [5] D.E. Bell and L.J. LaPadula, "Secure Computer Systems: Mathematical Foundations," Technical Report MTR-2997, The Mitre Corporation, Bedford, Mass., 1973.
- [6] P.A. Bernstein and N. Goodman, "Multiversion Concurrency Control-Theory and Algorithms," *ACM Trans. Database Systems*, vol. 8, no. 4, pp. 465-483, Dec. 1983.
- [7] P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Reading, Mass.: Addison-Wesley, 1987.
- [8] E. Bertino, P.A. Bonatti, and E. Ferrari, "TRBAC: A Temporal Role-based Access Control Model," *Proc. Fifth ACM Workshop Role-Based Access Control*, pp. 21-30, July 2000.
- [9] C. Chaney and S. Son, "Supporting the Requirements for Multilevel Secure and Real-Time Databases in Distributed Environments," *Proc. IFIP 11th Working Conf. Database Security*, pp. 57-71, Aug. 1997.
- [10] D.E. Denning, *Information Warfare and Security*. Reading, Mass.: Addison-Wesley, 1998.
- [11] B. George and J. Haritsa, "Secure Transaction Processing in Firm Real-Time Database Systems," *Proc. ACM SIGMOD*, pp. 462-473, June 1997.
- [12] I. Goldberg, *Information Warfare, I-War, IW, C4I, Cyberwar*. <http://www.psycom.net/iwar.1.html>, Jan. 1996.
- [13] H. Han, S. Park, and C. Park, "A Concurrency Control Protocol for Read-Only Transactions in Real-Time Secure Database Systems," *Proc. Seventh Int'l Conf. Real-Time Computing Systems and Applications*, pp. 458-462, Dec. 2000.
- [14] J. Haritsa, M. Carey, and M. Livny, "Dynamic Real-Time Optimistic Concurrency Control," *Proc. 11th IEEE Real-Time Systems Symp.*, pp. 94-103, Dec. 1991.
- [15] S. Jajodia and V. Atluri, "Alternative Correctness Criteria for Concurrent Execution of Transactions in Multilevel Secure Database Systems," *Proc. IEEE Symp. Security and Privacy*, pp. 216-224, May 1992.
- [16] S. Jajodia, L.V. Mancini, and I. Ray, "Secure Locking Protocols for Multilevel Database Management Systems," *Proc. IFIP 10th Working Conf. Database Security*, pp. 177-194, July 1996.
- [17] T.F. Keefe and W.T. Tsai, "Multiversion Concurrency Control for Multilevel Secure Database Systems," *Proc. 10th IEEE Symp. Research in Security and Privacy*, pp. 369-383, May 1990.
- [18] T.F. Keefe, W.T. Tsai, and J. Srivastava, "Database Concurrency Control in Multilevel Secure Database Management Systems," *IEEE Trans. Knowledge and Data Eng.*, vol. 5, no. 6, pp. 1039-1055, Feb. 1990.
- [19] W. Kim and J. Srivastava, "Enhancing Real-Time DBMS Performance with Multiversion Data and Priority Based Disk Scheduling," *Proc. 12th IEEE Real-Time Systems Symp.*, pp. 222-231, Dec. 1991.

- [20] V. Kumar, *Performance of Concurrency Control Mechanisms in Centralized Database Systems*. New Jersey, Prentice-Hall, 1986.
- [21] J. McDermott and S. Jajodia, "Orange Locking: Channel-Free Database Concurrency Control Via Locking," *Proc. IFIP Sixth Working Conf. Database Security*, pp. 267-284, Aug. 1995.
- [22] R. Mukkamala and S. Son, "A Secure Concurrency Control Protocol for Real-Time Databases," *Proc. IFIP Ninth Working Conf. Database Security*, pp. 235-253, Aug. 1995.
- [23] S. Pal, "A Locking Protocol for Multilevel Secure Databases Providing Support For Long Transactions," *Proc. IFIP Ninth Working Conf. Database Security*, pp. 199-221, Aug. 1995.
- [24] C. Park and S. Park, "Alternative Correctness Criteria for Multiversion Concurrency Control and a Locking Protocol via Freezing," *Proc. Int'l Database Eng. and Applications Symp.*, pp. 73-81, Aug. 1997.
- [25] C. Park, S. Park, and S. Son, "Priority-Driven Secure Multiversion Locking Protocol for Real-Time Secure Database Systems," *Proc. IFIP 11th Working Conf. Database Security*, pp. 200-211, Aug. 1997.
- [26] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST Model for Role-Based Access Control: Towards a Unified Standard," *Proc. Fifth ACM Workshop Role-Based Access Control*, pp. 47-63, July 2000.
- [27] L.C. Shu and M. Young, "Correctness Criteria and Concurrency Control for Real-Time Systems: A Survey," Technical Report SERC-TR-131-P, Purdue Univ., Ind., Nov. 1992.
- [28] S. Son, S. Park, and Y. Lin, "An Integrated Real-Time Locking Protocol," *Proc. Eighth Int'l Conf. Data Eng.*, pp. 527-534, Feb. 1992.
- [29] S. Son and B. Thuraisingham, "Towards a Multilevel Secure Database Management System for Real-Time Applications," *Proc. IEEE Workshop Real-Time Applications*, pp. 131-135, May 1993.
- [30] S. Son, R. David, and B. Thuraisingham, "An Adaptive Policy for Improved Timeliness in Secure Database Systems," *Proc. IFIP Ninth Working Conf. Database Security*, pp. 223-233, Aug. 1995.
- [31] S. Son, R. David, and C. Chaney, "Design and Analysis of an Adaptive Policy for Secure Real-Time Locking Protocol," *J. Information Sciences*, vol. 99, nos. 1/2, pp. 101-135, June 1997.
- [32] S. Son, R. Mukkamala, and R. David, "Integrating Security and Real-Time Requirements Using Covert Channel Capacity," *IEEE Trans. Knowledge and Data Eng.*, vol. 12, no. 6, pp. 865-879, Dec. 2000.
- [33] J. Stankovic, S. Son, and J. Hansson, "Misconceptions About Real-Time Databases," *Real-Time Database Systems: Architecture and Techniques*, K. Lam and T. Kuo, eds., Kluwer Academic, pp. 9-16, 2001.



**Chanjung Park** received the BS degree from Sogang University, Seoul, Korea, in 1988, the MS degree from Korea Advanced Institute of Science and Technology (KAIST) in 1990, and the PhD degree in computer science from Sogang University in 1998. She is a senior lecturer in the Department of Computer Education at Cheju National University, Korea. From 1990 to 1999, she worked for Korea Telecom as a technical staff member. Her current research interests are database security, real-time systems, web-based education, and internet security. Dr. Park is a member of the IEEE Computer Society and the Korea Information Science Society.



**Seog Park** received the BS degree in computer science from Seoul National University, Korea, in 1978, the MS and the PhD degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1980 and 1983, respectively. He is a professor of computer science at Sogang University, Seoul, Korea. Since 1983, he has been working in the Department of Computer Science of the College of Engineering, Sogang University. His major research areas are database security, real-time systems, data warehouse, digital library, multimedia database systems, role-based access control, and Web database. Dr. Park is a member of the IEEE Computer Society, ACM, and the Korea Information Science Society. Also, he has been a member of Database Systems for Advanced Application (DASFAA) steering committee since 1999.



**Sang H. Son** received the BS degree from Seoul National University, Korea, the MS degree from Korea Advanced Institute of Science and Technology (KAIST), and the PhD degree in computer science from University of Maryland, College Park, in 1986. He is a professor in the Department of Computer Science at the University of Virginia. His current research interests include real-time computing, database systems, distributed systems, and information security.

The goal of his research is to design and evaluate algorithms for responsive computer systems and databases for complex real-time applications. Recently, he is working on supporting multi-dimensional requirements, including real-time, security, and fault-tolerance, and applying feedback control framework in soft real-time systems. Dr. Son is an associate editor of *IEEE Transactions on Parallel and Distributed Systems*. He served as the guest editor for the *IEEE Transactions on Software Engineering*, program chair and general chair of several real-time and database conferences, including IEEE Real-Time Systems Symposium, Workshop on Parallel and Distributed Real-Time Systems, Workshop on Real-Time Database Systems, and International Conference on Real-Time Computing Systems and Applications. He has served as an ACM National Lecturer for 1991-1993, and is the editor of the book *Advances in Real-Time Systems*, published by Prentice-Hall in 1995, coeditor of *Real-Time Database Systems: Issues and Applications*, published in 1997 by Kluwer Academic Publishers, and coauthor of *Database Recovery*, published in 1998 by Kluwer Academic Publishers. He is a senior member of IEEE Computer Society.

► For more information on this or any computing topic, please visit our Digital Library at <http://computer.org/publication/dilib>.