다단계보안을 제공하는 데이타베이스 관리체계에서의 동시성 제어

손 용 락*⁰ , 문 송 천*
*한국과학기술원 정보 및 통신공학과

CONCURRENCY CONTROL IN MULTI-LEVEL SECURE DATABASE MANAGEMENT SYSTEMS: MLS/CC

Yonglak Sohn *[0] , Songchun Moon *

* Korea Advanced Institute of Science and Technology
Department of Information and Communication Engineering

1. Introduction

Multi-level secure database management system(MLS/DBMS) assigns data to more than one security levels and restricts database operations based on those levels. This paper is concerned with the transaction management in MLS/DBMS. We are now at the final stage of designing a secure concurrency control method that will be included in the relational DBMS 'IM', the first-ever developed one in Asia zone. This is the current status report of the DBMS 'MLS/IM' being developed at KAIST.

The goal of secure transaction management is to enforce security and at the same time to maintain a high degree of availability of the database to a large number of concurrent users. One of the most difficult challenges in developing MLS/DBMS is the ***covert channel*** problem. An inter-site communication occurs whenever a resource, such as memory, CPU time or a data item in a database, is shared. A user may send information to others by modulating the state of the shared resources. Other users who can detect changes in the state are capable of receiving the information. When this unintended communication channel leads to violation of a security policy, it is called a covert channel.

When covert channels have been eliminated, there is no illegal information flow. Nevertheless, another kind of threat still exists that violates the security of database systems. From a common-sense point of view, users classified at higher-security level perform more important tasks than users classified at lower-security level. If lower-security level users happen to obstruct the execution of higher-security level users' tasks such as retrievals and creations of information, they violate the security of database systems.

For secure scheduling, previous work [3,4,5,6] concentrated on eliminating the covert channels, but they failed to deal with the ***starvation*** of higher-security level users' requests. Moreover, ***deadlocks*** that could occur among same security level users' requests have never been dealt with. This paper proposes a secure scheduling scheme called ***multi-level secure concurrency***

controller (***MLS/CC***) that is able to eliminate not only covert channels but also starvation of higher security level users' requests and deadlocks. ***MLS/CC*** is based on the philosophy of concealment of uncommitted data. For this concealment, ***MLS/CC*** prevents the disclosure of uncommitted data and provides a yardstick against which a user is allowed to determine an appropriate data among several ones.

2. Models

2.1 Transaction Model

Transaction is an abstract unit of concurrent computation that executes atomically. The effects of a transaction do not interfere with other transactions that access the same data. Also, a transaction either happens with all of its effects made permanent or it doesn't happen and none of its effects are permanent. This paper models transactions as Definition 1.

Definition 1 (Transaction)

> Transaction is a finite sequence of database operations, such as read, write, commit, or abort operations. The sequence models the order in which database operations are send to the transaction scheduler. After the DBMS executes a transaction's commit(or abort) operations, the transaction is said to be committed(or aborted). A transaction that has started but is not yet committed or aborted is called active, A transaction is uncommitted if it is aborted or active. ■

In this paper, the read, write operations on data x and commit, abort operations of transaction are denoted by $R_i[x]$, $W_i[x]$, C_i, and A_i (respectively). The subscription of each operation indicates the transaction T_i that has tried the operation. The transaction number of T_i is denoted by $tn(T_i)$. When two or more transactions execute ***concurrently***, their database operations execute in an interleaved fashion. To describe the mechanisms for scheduling the concurrent transactions, this paper adopts the following definition of conflict.

Definition 2 (Conflict)

> Any two operations issued by concurrent transactions may ***conflicts*** if they both operate on the same data and at least one of them is write operation. ■

These conflicting operations can cause transactions to behave incorrectly, or interfere, thereby leading to an inconsistent database. The goal of ***transaction scheduling*** is to avoid interference and thereby avoid incorrectness.

2.2 Transaction on Multiversion Data

MLS/CC is based on the ***multiversion transaction scheduling*** scheme. In multiversion scheduling scheme, each write operation on data item x produces a new version of x. Thus, for each data item in database, there is a list of associated versions. $R_i[x]$ is performed by returning the value of x from an appropriate version in the x's list to T_i. However, the existence of multiple versions is visible only to the scheduler, and not to the user transactions that refer to the data as x. The versions of x are denoted by x_i, x_j, ..., x_n. For each version, the transaction interval is maintained. The transaction interval of x_i is composed of ***read-stamp*** and ***write-stamp***, denoted by ***R-stamp[x_i]*** and ***W-stamp[x_i]*** respectively. W-stamp[x_i] stands for the transaction that has created x_i, and R-stamp[x_i] indicates the largest transaction number of any transaction that has read x_i.

A ***multiversion history H*** represents the sequence of operations on the versions of data. Thus, the scheduler maps $W_i[x]$ and $R_i[x]$ in H into $W_i[x_i]$ and $R_i[x_j]$ ($j \le i$). To map $R_i[x]$ to $R_i[x_j]$, the scheduler examines W-stamps of x to find the version x_j that has the maximal W-stamp less than or equal to $tn(T_i)$. To map $W_i[x]$ to $W_i[x_i]$, the scheduler again examines to find the version of x_k that has the maximal W-stamp less than $tn(T_i)$. If R-stamp[x_k] > $tn(T_i)$, then scheduler rejects $W_i[x]$. Otherwise, the scheduler maps $W_i[x]$ to $W_i[x_i]$, and thereby new version x_i is created.

An H is ***one-copy serializable (1SR)*** if it is equivalent to a serial history over the same set of transactions executed over a single version database. To determine if an H is 1SR, a modified serialization graph, called ***multiversion serialization graph (MVSG(H))*** is used [1]. That H is 1SR if MVSG(H) is acyclic is proved in [1].

2.3 Security model

Previous work on secure transaction scheduling basically adopted ***Bell-LaPadula model*** [6] or ***restricted Bell-LaPadula model*** [3,4,5] as their security models. The Bell-LaPadula model is adopted as the security model in this paper since it is more flexible than the restricted Bell-LaPadula model.

The database system consists of a finite set D of objects (data items) and a set T of subjects (transactions). There is a lattice S of security levels with ordering relation '<'. A security level S_i dominates a security level S_j if $S_i \ge S_j$. There is a labeling function L that maps objects and subjects to a security level:

$$L: D \cup T \rightarrow S$$

Bell-LaPadula model has two requirements: (1) ***Simple Security Property***- If T_i reads data x then $L(T_i) \ge L(x)$, and (2) ****-Property***- If T_j writes data x, then $L(T_j) \le L(x)$.

Bell-LaPadula model is sufficient to prove that security is not violated through data accesses [2]. However, if $L(T_i) > L(T_j)$ and $W_j[x]$ is delayed due to $R_i[x]$ or $W_i[x]$, then a covert channel between T_i and T_j shall be established. Furthermore, since $W_j[x]$ could obstruct the execution of $W_i[x]$ and $R_i[x]$, T_j can force T_i fall into starvation. Consequently, complementary measures to Bell-LaPadula model are necessary in MLS/DBMS.

3. Related work and problems

3.1 Related work

If $L(T_i) > L(T_j)$ and T_i and T_j are in a conflict over the shared data, the unconditional precedence approach used in [3,4,5] immediately grants T_j's operation. Since T_j can be executed without T_i's interference, the unconditional precedence approach can schedule the transactions without inducing covert channels. However, such scheduling scheme is unfair because T_i will suffer from starvation due to the interference caused by T_j.

[6] adopts multiversion scheduling to reduce the rate of conflicts. In [6], for version control, the R-stamp and the W-stamp of each version are set to the security level of a transaction. The lower the transaction's security level, the larger the R(W)-stamp becomes, and vice versa. Since $L(T_i) > L(T_j)$, R-stamp$_i[x_k]$ is less than $L(T_j)$. Therefore, $W_j[x_j]$ can also be executed without the interference of $R_i[x_k]$. Consequently, the security-level based approach used in [6] can schedule the transactions without inducing covert channels.

3.2 Problems in related work: unfairness and deadlocks

In case T_i and T_j are scheduled by the unfair schedulers like ones in [3,4,5,6], T_i or T_j may fall into starvation due to the schedule giving precedence to one over the other unconditionally. If $L(T_i) > L(T_j)$ and T_j intentionally induces T_i's starvation, T_j threatens the security of database system, even if there is no illegal information flow. In **Example 1**, T_j exploits the weakness of unconditional precedence, thereby putting T_i into starvation.

Example 1 (*Starvation of higher-security level transaction due t orange-locks*):

```
Ti : Ri[a]Ri[b]Ri[c]Ri[d] Ai              Ri[a]Ri[b]Ri[c]Ri[d]Ai          ........
Tj :            Wj[b]          Cj                   Wj[b]          Cj    ........
Time ------------------------------------------------------------------------->
```

Fig.1 T_i's starvation due to the conflict between $R_i[b]$ and $W_j[b]$

Suppose that two concurrent transactions, T_i and T_j, interleave their execution as follows, as time goes from left to right. Suppose also that $L(a,b,c,d) \geq L(T_i) > L(T_j)$.

When T_j tries to set a write-lock on b, a scheduler immediately grants T_j's write-lock and changes T_i's read-lock on b to an *orange-lock* [3], which indicates a possibility of an incorrect read. If T_i has orange-locks when it has reached its *home-free-point* [3], which means that all data items to be read by T_i are read-locked and read into T_i's local work space, T_i is aborted. If T_j tries the operation $W_j[b]$ periodically and intentionally, T_i is thrown into starvation. Note that any higher-security level transaction cannot acquire information at a proper time due to the interference caused by lower-security level transaction. Therefore, starvation of higher-security level transaction is insecure.

End of Example 1 ■

The interference in two or more write operations are shown in **Example 2.**

Example 2 (Starvation of higher-security level transaction in security-level based approach):

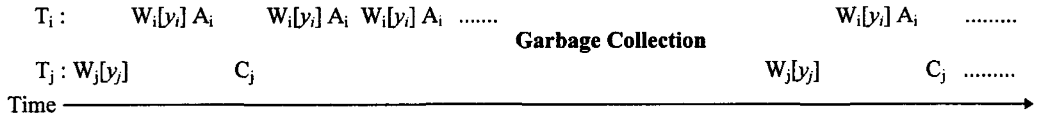

Fig. 2 T_i's starvation due to the interference of T_j's $W_j[y_j]$

Suppose that $L(y) \geq L(T_i) > L(T_j)$ and that a scheduler proposed in [6] is used to synchronize T_i and T_j (Fig. 2). According to the Bell-LaPadula model, both T_i and T_j can produce new versions of y. Since W-stamp$_i[y_i]$ < R-stamp$_j[y_j]$, $W_i[y_i]$ is rejected, and thus T_i is aborted. To make matters worse, $W_i[y_i]$ is repeatedly rejected until garbage collector erases y_j. If T_j tries $W_j[y_j]$ intentionally whenever garbage collector erases y_j, T_i is thrown into starvation. Since higher-security level transaction cannot produce information at a proper time because of the interference caused by lower-security level transaction, the security-level based approach is insecure.

End of Example 2 ■

When $L(T_i) = L(T_j)$, the exposure of uncommitted data approach taken in [3,4,5,6] is subject to deadlocks. The delayed access method in [3,4] leads to deadlock in the process of granting rights to shared data. The delayed commit method in [5,6] makes transactions read uncommitted values, so that each transaction must wait for the commitment of a transaction from which the uncommitted values had been read. **Example 3** shows a possible deadlock that could occur when transactions are scheduled according to the scheduling scheme proposed in [6].

Example 3 (Deadlock in multiversion scheduling):

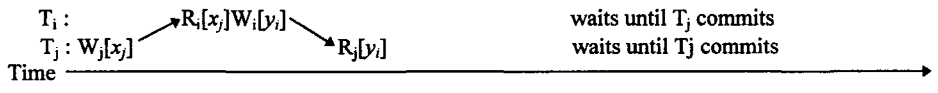

Fig. 3 Deadlock due to reading the uncommitted value

If L(x) = L(y) = L(T_i) = L(T_j), then $R_i[x_j]$ is able to read x_j created by $W_j[x_j]$, and $R_j[y_i]$ is able to read y_i created by $W_i[y_i]$ (Fig. 3). Since both T_i and T_j have read the uncommitted values, T_i is not allowed to commit until T_j commits, and similarly T_j is not allowed to commit until T_i commits, respectively. Therefore, both T_i and T_j are thrown into deadlock.

End of Example 3 ■

4. Multi-Level Secure Concurrency Controller: MLS/CC

We propose ***MLS/CC*** as a transaction scheduler that is capable of scheduling transactions without covert channels, starvation of higher-security level transaction, and deadlocks. Although ***MLS/CC*** is based on multiversion scheduling, but the philosophy of ***MLS/CC*** is solidified to allow communication of two or more transactions only via the committed data. Thus, any two communicating transactions can never encounter each other whilst they are active. For the concealment of uncommitted data and the selection of proper data, ***MLS/CC*** has the following properties.

Ordering Property: When a transaction begins, it is assigned a transaction number which is larger than those of the committed transactions and smaller or equal to those of the transactions that will arrive later. This property provides a yardstick for deciding the version that should be selected for the transaction's read operations.

Readability Property: Transaction reads the committed value that has the largest W-stamp among various versions. However, the W-stamp of the selected version will always be smaller than the transaction number of the transaction that read it. That is, it reads the value that has the youngest and committed version.

Visibility Property: As long as a transaction is active, the effects of its write operations are invisible to other transaction; that is, those effects are visible only after the transaction commits. While the transaction commits, the R-stamps and W-stamps of the versions that have been produced by the write operations are set to the transaction number that is larger than those of active transactions.

These properties are sufficient to prove that ***MLS/CC*** is one-copy serializable(1SR) [1] and secure. Before proving the soundness of ***MLS/CC***, we describe an algorithm for secure transaction scheduling as follows.

The VC appearing at both steps [1] and [6], controls the visibility of data items. For example, VC at step [1] forces T_i to read the committed version of data and VC at step [6] restricts y_i to be visible only to the transactions that have begun after T_i had committed. VC serves the purpose of assigning transaction number to T_i and is incremented when T_i completes. This is possible because the transaction number order need not necessarily correspond to the order in which transactions complete their execution. VC also serves as a yardstick, so that ***MLS/CC*** can guarantee the Ordering Property. According to steps [2] and [7], ***MLS/CC*** can achieve the Readability Property. Step [5] preserves ***MLS/CC***s Visibility Property. Setting R-stamps to $-\infty$ at steps [3] and [5] guarantees that the write operations of other transactions' can be executed without interference from T_i's read operations.

Algorithm 1 (MLS/CC): Input: T_i / Output: new data version y_i, R-stamp$_i[x_k]$

```
Actions of T_i    Actions of MLS/CC corresponding to the actions of T_i
Begin(T_i)        [1] $tn(T_i)$ <- VC
R_i(x_k)          [2] Return the k-th version of x which has the largest W-stamp among
                      various versions and W-stamp$_k[x_k]$ is smaller or equal to $tn(T_i)$;
                  [3] Set R-stamp$_i[x_k]$ to $-\infty$;
W_i[y_i]          [4] Create a new version of $y$;
                  [5] Set R-stamp$_i[y_i]$ and W-stamp$_i[y_i]$ to $-\infty$ and $\infty$ respectively;
End(T_i)          [6] VC++;
                  [7] Set R-stamp$_i[x_k]$, R-stamp$_i[y_i]$, and W-stamp$_i[y_i]$ to VC;
```

End of Algorithm 1■

Theorem 1 (1SR): *MLS/CC* is one-copy serializable.

Proof*:** Assume that $L(T_i) > L(T_j)$. If T_i reads the version x_j created by T_j, an edge $T_j \rightarrow T_i$ is added to MVSG(H). According to the Readability Property, T_i reads the committed value x_j and W-stamp$_j[x_j] < tn(T_i)$. With the Visibility Property, $tn(T_j) <$ W-stamp$_j[x_j]$, so that $tn(T_j) < tn(T_i)$. In case T_j reads the version y_i created by T_i, owing to the Visibility Property, T_j can read y_i after T_i commits. After T_i commits, according to the Visibility Property, $tn(T_i) <$ R-stamp$_i[y_i]$ and $tn(T_i) <$ W-stamp$_i[y_i]$. Since $tn(T_j) < tn(T_i)$, $tn(T_j) <$ W-stamp$_i[y_i]$. Therefore, T_j can never read y_i, so that $T_i \rightarrow T_j$ is never added to MVSG(H). Now that there is no cycle between T_i and T_j, ***MLS/CC is one-copy serializable. ❒

Theorem 2 (Secureness): *MLS/CC* is secure.

Proof*:** Assume that $L(T_i) > L(T_j)$. While T_i is active and T_i has accessed $x_{i(k)}$, R-stamp$_i[x_{k(i)}]$ is set to $-\infty$, resulting R-stamp$_i[x_{k(i)}] <$ W-stamp$_j[x_j]$. Furthermore, while T_j is active, W-stamp$_j[x_j]$ is set to ∞, so W-stamp$_j[x_j]$ is never smaller than R-stamp$_i[x_{k(i)}]$. Therefore, T_j's $W_j[x_j]$ can be executed without the interference from T_i's $R_i[x_k]$ or $W_i[x_i]$. Now that $L(T_i) > L(T_j)$, there is no covert channel between T_i and T_j. According to the Readability Property, when T_i tries to read the effect of T_j, T_j has already been committed. Moreover, with the Visibility Property, T_i never reads T_j's uncommitted value. Therefore, the abortion of T_j can never cause the abortion of T_i. Because $L(T_i) > L(T_j)$, the starvation of higher-security level transaction never occurs. Hence that there is neither covert channel nor starvation of higher-security level transaction, ***MLS/CC is secure. ❒

Theorem 3 (Deadlock-free): *MLS/CC* is free from deadlock.

Proof: Assume T_i and T_j interleave their execution and create x_i and y_j respectively. According to the Visibility Property, W-stamp$_j$[y_j] is set to ∞ while T_j is active, so tn(T_i) is always smaller than W-stamp$_j$[y_j]. Therefore, while T_j is active, T_i can never read y_j because the Readability Property forces T_i to read y_k whose W-stamp is smaller than tn(T_i). Analogous to T_i's behavior, T_j also can never read x_i while T_i is active. Now that T_i and T_j never read the uncommitted version of data, *MLS/CC* is free from deadlock. □

5. Conclusions

In this paper, we demonstrated that it is possible to eliminate covert channels without causing the starvation in higher-security level transaction. Furthermore, compared to the previous work, ***MLS/CC*** is expected to outperform the unconditional precedence approach and the security-level based approach due to following benefits. First, higher-security level transactions do not suffer from the starvation, so that they can execute concurrently with lower-security level transactions. Moreover, as long as transactions are controlled under the ***MLS/CC***, they are never confronted with deadlocks. An obvious burden of maintaining multiple versions is storage space; however, maintaining multiple versions would not incur much overhead of ***MLS/CC***, since multiple versions shall be of use anyway by the recovery algorithm employed in an MLS/DBMS.

When selecting an appropriate version among multiple versions, a more serious consideration is needed. An investigation of selecting proper version, whilst a transaction reads data or a data manager collects garbage, that incorporates the security policy will be included our future research. For saving storage space, a method of reducing the number of versions, as well, will be studied.

References

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

[2] Ravi Sandhu, "Mandatory Controls for Database Integrity," DATABASE SECURITY, III: Status and Prospects, ed. David L.Spooner, Carl Landwehr, Elsevier Science Publishers B.V., 1990, pp. 143 - 150.

[3] John McDermott and Sushil Jajodia, "Orange Locking: Channel-Free Database Concurrency Control via Locking," DATABASE SECURITY, VI: Status and Prospects, ed. Bhavani M. Thuraisingham, Carle. Landwehr, Elsevier Science Publishers B.V., 1993, pp. 267 - 284.

[4] Oliver Costich and Sushil Jajodia, "Maintaining Transaction Atomicity in MLS Database Systems with Kernalized Architecture," DATABASE SECURITY, VI: Status and

Prospects, ed. Bhavani M. Thuraisingham, Carle. Landwehr, Elsevier Science Publishers B.V., 1993, pp. 249 - 265.

[5] P. Ammann and S. Jajodia, "A Timestamp Ordering Algorithm for Secure, Single-Version, Multi-Level Databases," DATABASE SECURITY, V: Status and Prospects, ed. Carl E. Landwehr, Sushil Jajodia, Elsevier Science Publishers B.V., 1992, pp. 191 - 202.

[6] T. F. Keefe, W. T. Tsai and J. Srivastava, "Multilevel Secure Database Concurrency Control," Proceedings of IEEE Symposium on Security and Privacy, 1990, pp. 337 - 344.