Transaction Scheduling in Multi-Level Secure Database Systems

Yonglak Sohn Songchun Moon

Dept. of Information and Communication Engineering
Korea Advanced Institute of Science and Technology (KAIST)
207-43 Cheongryangni Dongdaemun Seoul 130-012, Republic of Korea
E-mail: syl@sicom.kaist.ac.kr

Abstract

Transactions are vital for multi-level secure database management systems(MLS/DBMSs) because concurrent execution of transactions potentially has conflicts among their accessing to shared data. When conflict occurs, only one transaction is granted to access the shared data, other transactions should be delayed until they can safely use the data. Those conflicts may lead to the security problems in MLS/DBMS. If conspirators produce those conflicts intentionally, they can establish the unexpected communication path called covert channel between high security level users and low security level users. This paper proposes a transaction scheduling scheme called Conflict-Insensible Scheduling (CIS) that hides conflicts from low security level transactions to prevent the covert channels.

1: Introduction

In tight security systems, all database users and data items are assigned security levels and it is the responsibility of the system to assure that all users are allowed to have accesses to only those data for which they have been granted. To assure this, so called multi-level secure database management system(MLS/DBMS) is the key. Most commercial DBMSs provide some form of database security based on so called discretionary access control(DAC)[7] in which they simply control the modes of privileges of users to data. They could be called single-level security scheme. In the single-level security, if a process is affected by a malicious piece of code such as computer virus, the access to data made by one user could pass the data to other users, because a process usually inherits all rights of user who invoked it. In this respect, more sophisticated and strong security policies are necessary. Enforcing multi-level security, one approach to realize this idea, can be found in the mandatory access control(MAC)[7]. It is important to note that security could be harmed either through data accesses or through unexpected communication paths. MAC is sufficient against data accesses, but not against unexpected communication channels.

When the unexpected communication channel leads to violate a security policy, it is called covert channel[3]. Communication through covert channel is possible whenever resources, such as CPU time, memory blocks, disk sectors, or data items in database, are shared. Any way of changing the states of the shared resources is a candidate for the covert channel. A user who can change the state of the shared resource can send signal to others. Accesses to the shared resources, of course, are under the control of system software, e.g., operating system or DBMS. The system software, however, is capable of controlling only the usage of the shared resources, such as memory allocation, competition for disk sectors, or consistency of the data.

It is really hard to grasp the intents of users in accessing shared resources. It should be noted that covert channels are solely created by malicious intent of users. For instance, in some DBMSs a user is allowed to retrieve the system catalogs to see how many users are given permission to read or update the files that have been created by him. This sort of system information can be made open to users. In this respect, it is almost impossible to block all ways of touching system information from users. For another instance, one can control the access to a shared data item in the database as `1' for hold, `0' for release. Other user who shares the data item can distinguish the states of data into held and released. In this way, only one bit of meaningful information can play a decisive role in extracting a very important fact. For one more example, the fact that the president of a republic had an accident can be represented by only one bit. If this bit of information can be manipulated by unauthorized users, it could really be a threat to the design of secure systems.

Since the database is composed of very large number of shared data items, the covert channel using the shared

0-8186-6430-4/94 $4.00 © 1994 IEEE

data items in the database can convey a large number of bits, and as a result increases the bandwidth of the covert channel. Moreover, compared to the other shared resources such as a CPU or a tape driver, competition for each data item is loose, thus only a small number of noisy bits can exist in the covert channel. As the bandwidth of covert channel is increased and the noisy bits in the covert channel are decreased, the covert channel can deliver many correct bit streams for a fixed interval of time.

In MLS/DBMSs, there are two types of covert channels; namely storage channel and timing channel[3]. The storage channel is established when a low security level transaction learns of the existence of data classified at high security level. For instance, consider that a unique data named `A' exists and is classified at high security level. Undoubtedly, low security level transactions cannot see the value of `A.' However, if a low security level transaction attempts to create a new data named `A,' the request will be denied. Through this denial, the low security level transaction learns of the existence data A that has been classified at high security level. In this sense, a high security level transaction could signal information to a low security level transaction by removing and creating A. In contrast to the storage channel, timing channel signals information by controlling a delay that is observable in a low security level transaction. The timing channel depends on the delay of a low security level transaction that shares a data that the low security level transaction and the high security level transaction are in conflict over.

For the storage channel, perceiving the existence of data can be obtained through a write operation to the data classified at high security level. If a security policy forces a restriction on the write which accesses a high security level data, the storage channel can be prevented since it is impossible for a low security level transaction to learn the existence of high security level data. Most of past work on the covert channel analysis adopted this sort of restriction as their security policies. However, since read attempt by a high security level transaction toward a low security level data cannot be restricted by any security policy, and write operation is essential to build a database, conflict between read and write is unavoidable. In this respect, a special idea for preventing timing channels under the circumstances that high security level transaction's read operations conflict with low security level transaction's write operations is necessary. This paper deals with this issue.

Execution of concurrent transactions may lead to create unexpected covert channels. In database systems, any two operations issued by concurrent transactions may conflict if they operate on the same data item and one or both of them are write operations. A schedule is concurrent if it is an interleaved sequence of operations from more than one transaction. An interleaved sequence of reads and writes potentially interferes with each other. Transaction scheduler reorders those reads and writes in order to maintain the correctness of the database and to allow as much concurrency as possible. However, considered from the MLS/DBMS point of view, there is a problem of scheduling several concurrent transactions because the interference between reads and writes could lead to the delay of some operations; this delay could be represented as one bit of information which could be leaked to unauthorized persons through covert channel. Example 1 shows this sort of covert channel due to read/write or write/write conflicts.

Example 1 (Covert channel due to conflicting operations):

Suppose that two transactions, T_1 and T_2, execute concurrently and they access a shared data `A' (Figure 1). Suppose also that before their execution the last committed value of A is 0. If the scheduler outputs the sequence of reads and writes as they are arrived, T_2 reads 0 from A at time t_0, T_1 writes 10 to A at t_1 and commits at t_2, and T_2 then commit at t_3. At t_2, the value of data A stored in the database is changed from 0 to 10, and thus the value of A read by T_2 at t_0 is inconsistent with the value of A in the database. At t_1, T_2 may use the value of A read at t_0. In case that A represents the human life, e.g., `0' means living and `10' means death, using the inconsistent value of A may lead T_2 to obtain undesired result. To maintain the consistency of the value of A read by T_2 and the value of A stored in database, the scheduler should reorder the two conflicting operations, T_1's Write(A) and T_2's Read(A), and as a result produces either one of schedule: output-1 or output-2. In case of output-1, T_1 experiences a delay because T_2 interferes with T_1. In case of output-2, T_2 instead experiences a delay. In output-1, T_1 can interpret its experience of delay as `1' or `0' that could serve as one bit of unauthorized information arrived from T_2. If T_1 and T_2 do not conflict, i.e. neither do they run concurrently nor do access a shared data, then the scheduler would simply output the reads and the writes in the exactly the same order as they are arrived; in this case no covert channel arises, since T_1 and T_2 do not experience delay at all.

If T_1 and T_2 are classified at different security levels and scheduled according to the conventional scheduling schemes, such as locking or timestamp ordering, the higher security level transaction can leak bit stream to the lower security level transaction by controlling the former's operations that interfere with the latter's operations.

Moreover, if T_1 and T_2 were conspired each other, they would apply promised protocols to a communication and an interpretation of the bit stream. According to the protocol, the low security level transaction can obtain the bit stream from high security level transaction, and then can transform the bit stream to an information. This information flow is illegal because it allows a lower security level user to obtain the high security level information through a covert channel established between T_1 and T_2.

End of Example 1. ■

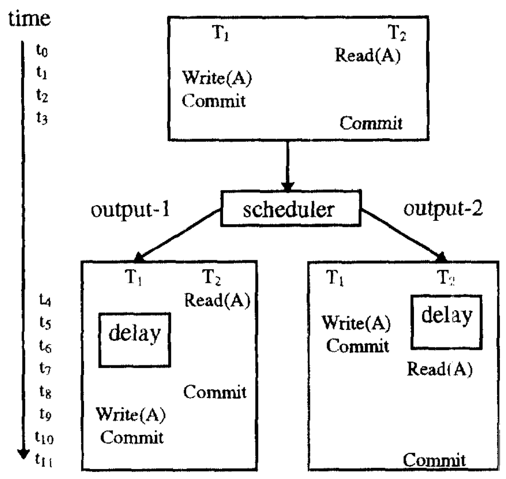

Figure 1. Reordering the conflicted operations

The elimination of interference between two transactions, so called noninterference, is necessary to make a system prevent covert channels. If a scheduler does not control the two conflicting operations, the write operation may lead to lost update and the read operation may lead to dirty read. This then forces rejection of one of the conflicting operations. Therefore, conflicts between two transactions always allow the outcome of transactions to distinguish into two different states, i.e., rejected form or permitted to access to a shared data. The rejection incurs a delay in the execution of the transaction whose operation had been rejected. The low security level transaction, then, is able to recognize the states from its experience of delay incurred by an interference from a high security level transaction. It then can interpret each state as `1' or `0.' Therefore, a low security level transaction can infer the high security level information by the contribution of the interference from a high security level transaction. To prevent the interference from a high security level transaction, the output schedule of a low security level transaction must be unaffected by the input schedule of high security level transaction. If the interference from a high security level transaction has been prevented, a low security level transaction cannot distinguish between the different output schedules; one has been scheduled in response to a sequence of inputs including the high security level transactions' operations and the other in response to an input sequence in which all high security level operations have been removed.

Most of past works [8,9,10] on concurrency control in MLS/DBMSs have been concerned with the noninterference to prevent the covert channels. They realized the noninterference by giving a precedence to the operations of low security level transactions. The idea of their realization is to let low security level transactions proceed without sensing the interference from high security level transactions. Although they can prevent covert channels, in this way, theirs delaying the execution of high security level transactions brings about another problem.

This paper proposes a concurrency control scheme in MLS/DBMS called conflict-insensible scheduling (CIS). The CIS hides conflicts from low security level transactions to prevent the covert channels. The CIS allows a low security level transaction to run to completion without an experience of delays. Therefore, the transaction can run as if it has not been interleaved with the operations of high security level transaction. To avoid the conflicts, giving a precedence to low security level transactions might be considered. However this is not true in our CIS because it definitely creates unfairness against high security level transactions.

The basic idea of the CIS is that when conflict occurs between low security level transaction's write operation and high security level transaction's read operation, it allows the low security level transaction to process the conflicted write operation by recording the result of write operation to a buffer. However, at the internal level, for the atomicity of transactions, the results of the low security level transaction's conflicted write operations are observable to other transactions only after the low security level transaction has committed. Moreover, CIS does not withdraw the access right that has been granted to the read operation of high security level transaction by allowing the high security level transaction to read the data resided in database. The fact that CIS does not violate the serializability is proved at section 5. CIS is applicable to the locking scheme and assumes that a transaction tries at most one write operation to the same data. This is reasonable because if a transaction writes to the same data twice, it itself imposes the lost update.

The rest of this paper is organized as follows. Section 2 describes the related works and their problems. Section 3 introduces the security model and transaction model. In section 4, the conflict-insensible scheduling is proposed and shows how it conceals the abort and the delays of a transaction to prevent the covert channels. Section 5 shows the correctness of conflict-insensible scheme. Conclusion is given in section 6.

2: Related Work

Most of the past work on concurrency control in MLS/DBMS realize the noninterference between the high and the low security level transactions by avoiding the conflicts [8,9,10]. To avoid the conflicts, when conflicts occur, they give a precedence to the operations of low security level transaction, so that the low security level transaction can proceed its execution without interference of high security level transaction.

In [8], they simply let the high security level transaction set read locks on low security level data items as in a conventional, untrusted database system. If a low security level transaction tries a write lock on one of the same data items, they immediately grant low security level transaction's write lock and change the high security level transaction's read lock to an orange lock, indicating the possibility of an incorrect read. If high security level transactions read set includes orange locks, it should be aborted.

In [9], they forces high security level transactions to wait to start until theirs read set has a null intersection with the write sets of all low security level transactions that are active. Therefore the low security level transactions can proceed their executions without interference of high security level transactions.

In [10], timestamp ordering scheme is applied to the concurrency control and presents two solutions for the problem of what to do about high security level transaction's read operation for low security level data item. The first solution forces high security level transactions to wait until it is safe to execute read operations. In the second solution, high security level transactions do not wait to read low security level data, but do delay their commitments until timestamp ordering with respect to conflicting operations are guaranteed. If a conflicting operation occurs, the high security level transaction is aborted.

Although these works allow low security level transaction to achieve noninterference from high security level transactions, the starvation of high security level transaction is inevitable. Consequently, the result of their scheduling is unfair. If the write operations of a low security level transaction or the read operations of a high security level transaction occur very frequently, the high security level transaction will run into a starvation. Example shows the starvation of high security level transaction.

Example 2 (Starvation of high security level transaction due to the precedence):

T(H) is classified at high security level. $T_1(L)$, $T_2(L)$, $T_3(L)$, and $T_4(L)$ are classified at low security level, and data A is a high security level data (Figure 2). Although T(H) gets the read lock to A earlier than other transactions, $T_1(L)$'s write lock is granted at t_2. Therefore, T(H) relinquishes the value read at t_1. $T_1(L)$ commits at t_3, nevertheless, T(H) cannot read because $T_2(L)$ requests write lock at t_3. To achieve the noninterference from high security level transactions, T(H)'s R(A) cannot be executed until all the write locks of low security level transactions are released at t_6.

Time....T(H).....T_1(L).....T_2(L).....T_3(L).....T_4(L)
t_0...........B-Tx
t_1...........R(A)....B-Tx
t_2.......................W(A).....B-Tx
t_3.......................Cmt......W(A).....B-Tx
t_4......................................Cmt.......W(A).....B-Tx
t_5..Cmt......W(A)
t_6..Cmt
t_7............R(A)
t_8............Cmt

Figure 2. Starvation of high security level transactions

Starvation of T(H) lasts for 5 clocks, i.e., from t_2 to t_6. Let us assume that the scheme that gives precedence to low security level transactions has been applied to a DBMS running at missile site. Imagine that an enemy launched a missile and A is the path of the missile that decides headquarters to launch a missile killer. However, T(H), the transaction of headquarters, cannot monitor the path of the missile at proper since A is held by low security level transactions for updating. At t_7, T(H) can read the last path of the missile that may be the target of the enemy's missile.

End of Example 2. ■

3: The Models

In this section, we present the models for security and transaction

3.1: Security Model

The security model used in this paper is that of Bell and LaPadula [6,7]. The database system consists of a finite set D of data items and a set T of transactions. There is a lattice of S of security levels with ordering relation <. There is a labeling function L that maps data items and transactions into a security level:

$L: D \cup T \rightarrow S$

Definition 1:

Security level u **dominates** v in a lattice if u v and there is no security level w for which u > w > v. ❐

This paper considers two mandatory access control requirements:

(Simple Security Property)

If transaction T_i reads data item x then $L(T_i) \geq L(x)$.

(Restricted *-Property)

If transaction T_j writes data item x then $L(T_j) = L(x)$.

According to the simple security property, a data item is allowed a read access to a transaction only if the former's security level is identical or higher than the latter's security level. The restricted *-property allows a transaction to write data item only if the former's security level is identical to the latter's security level

Definition 2:

Read-down, **read-equal**, and **read-up** are the operations that transaction T_i reads data item x if its labeling functions are $L(T_i) > L(x)$, $L(T_i) = L(x)$, and $L(T_i) < L(x)$ respectively. ❐

Definition 3:

Write-down, **write-equal**, and **write-up** are the operations that transaction T_j writes data item x if its labeling functions are $L(T_j) > L(x)$, $L(T_j) = L(x)$, and $L(T_j) < L(x)$ respectively. ❐

The security model in this paper allows a transaction to issue read-down, read-equal, and write-equal operations. This is sufficient to prove that security is not violated through data access [6,7].

3.2: Transaction Model

A transaction is an abstract unit of concurrent computation that executes atomically. The effects of a transaction do not interfere with each other transaction that accesses 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 5.

Definition 4:

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

Definition 5:

All transactions are **single-level**. That is, every action in a transaction has the same security level, and thus transaction's security level is assigned only once at the transaction starts. ❐

As two or more transactions can access the same data concurrently, these concurrent accesses must be controlled for the correctness of both data and transactions. The schedulers appear in this paper is based on the locking scheme. Locking is a mechanism commonly used to solve the problem of synchronizing access to shared data. Each data item has a lock associated with it. Before a transaction T_1 may access a data item, the scheduler first examines the associated lock. If no transaction holds the lock, then the scheduler obtains the lock on behalf of T_1. If T_2 does hold the incompatible lock, then T_1 has to wait until T_2 releases it. The locking rule adopted in this paper is strict two phase locking (strict 2PL).

Definition 6:

The **strict 2PL** requires the scheduler to release all of a transaction's locks together, when the transaction terminates, i.e., after the data manager acknowledges the processing of commit or abort. ❐

Definition 7:

T(H) denotes the high security level transaction and **T(L)** denotes the low security level transaction, i.e., L(T(H)) > L(T(L)).

4: Conflict-Insensible Scheduling

4.1: Features of Conflict-Insensible Scheduling

Conflict-Insensible Scheduling (**CIS**), this paper proposes, is a new concurrency control scheme for preventing covert channels. CIS allows T(L) to run to completion without an exposure of delays. Most of the past work attempted to prevent covert channels by giving precedence to the operations of T(L) which conflict with the operations of T(H). Thus, they definitely create unfairness against T(H).

In contrast to the past works, CIS does not give precedence to the operations T(L), so that T(H) does not need to suffer from unfairness. When conflict occurs between T(H)'s read and T(L)'s write, CIS allows T(L) to process it's write operation immediately on the buffer instead of delaying or giving precedence to the write operation. Thus, a conspirator who has been monitored the actions of colleague's transaction (classified at low security level) cannot perceive the conflicts, so that the conspirator infers that there is no information leaking from high security level conspirator even though he had tried to send an information.

CIS is based on the strict 2PL scheme. CIS does not suspend the execution of T(L)'s conflicting operation until the conflicted T(L) can obtain its requested lock, but goes on the conflicting operation and proceeds to execute next operations. When T(L) meets a commit operation, CIS announces T(L)'s commitment and allows the effects of T(L)'s write operations to be observable to other transactions. For the durability of transaction, CIS records the effect of T(L)'s write operations to the database after CIS announced T(L)'s commitment. When every transaction requests to abort, CIS immediately executes the rollback procedure.

From now on, every case of conflict that occurs in the input to CIS is considered and the processing of CIS on each case is shown.

Case 1: T(H) reads before T(L) writes.

```
T(H) : R(X)
T(L) :        W(X)
```

CIS allows T(H) to read X from database and T(L) to write X at write-buffer.

Case 2: T(H) reads the same data twice and T(L) writes and reads the same data.

```
T(H) : R(X)            R(X)
T(L) :        W(X)            R(X)
```

CIS allows T(H) to read X from database and T(L)'s W(X) to write X at write-buffer. T(L)'s R(X) reads X from write-buffer instead of database, i.e., it reads the effect of T(L)'s W(X). Since T(L)'s W(X) writes X at a different place from the place T(H) accessed, T(L)'s W(X) can be executed immediately. Moreover, since T(L)'s R(X) reads X from write-buffer, the conflict between the first R(X) of T(H) and T(L)'s W(X) is not observable to T(L)'s R(X). Therefore, T(L) cannot perceive the conflict between the T(H)'s R(L) and T(L)'s W(X).

As a result of case 1 and case 2, T(L) does not experience any delay and CIS does not withdraw the access right that had been granted to T(H)'s R(X) while T(L) and T(H) are executing concurrently and both of them access the same data. Consequently, the covert channel between T(H) and T(L) has not been established and T(H) does not need to be suffered from unfairness. The effects of T(L) in write-buffer are written into database only after there is no transaction that has conflicted with T(L) and has not been committed yet. When T(H) tries to read operation and other transactions had conflicted with T(H)'s previous read operations, T(H) is prohibited from reading the value in write-buffer.

Case 3: T(H) reads X after T(L) tried W(X).

```
T(H) :        R(X).....delay......R(X)
T(L) : W(X)          commit
```

According to the strict 2PL scheme, T(H)'s R(X) should be delayed until T(L) releases write-lock on X. The delay of T(H)'s R(X) is natural in the view of locking scheme and it does not bring about the information flow with the violation of security policy.

Case 4: Conflict between R(X) and W(X) that are issued by the transaction classified at the same security level.

```
T_1(L) :        R(X)......delay.....R(X)
T_2(L) : W(X)          commit
```

Analogously to the case 3, T1(L)'s R(X) should be delayed until $T_2(L)$ releases the write-lock on X.

The R(X)s appeared at both case 3 and case 4 can be executed more efficiently in case that they can read X from write-buffer instead of database. For achieving this efficiency, CIS announces T(L)'s or $T_1(L)$'s commitment when CIS serves the commit operation.

Case 5: Conflict between R(X) and W(X); both of them are issued by the transactions that are classified at the same security level and R(X) appears before W(X).

```
T_1(L) : R(X)          commit
T_2(L) :        W(X)......delay...W(X)
```

Although R(X) conflicts with W(X) and R(X) issued before W(X), if they are issued by the transactions classified at the same security level, they are scheduled according to the strict 2PL scheme. Notwithstanding the delay of W(X), since $T_1(L)$ and $T_2(L)$ are classified at the same security level, there is no information flow with the violation of security policy.

Case 6: Conflict between two W(X)s.

```
T_1 : W(X)            commit
T_2 :        W(X)......delay......W(X)
```

According to the restricted *-property, a transaction can write data that are classified at the same security level of the transaction. Therefore, T_1 and T_2 should be classified at the same security level. Analogously to the case 5, the delay of W(X) does not bring about the illegal information flow.

4.2: Algorithm for CIS

In this section, a concurrency control algorithm for CIS is presented. In this algorithm, T_i and T_j are transactions that are competing for access shared data. The write-buffer copes with the write operations that have conflicted with other transaction's read operations. Moreover, we assume that all the database operations handled by this algorithm have already been validated according to the security policy, so that the database operations are never rejected with the violation of security of the system. The conflicts between the database operations are detected and informed by the lock manager.

Algorithm CIS:

[1] If a transaction aborts, spaces in write-buffer that have held the effect of the transactions write operation is given back.

[2] If a transaction tries to commit, announce the commitment immediately. The effects of the transaction's write operations that have been held within the write-buffer are written into the database.

[3] Whenever a transaction tries to read on the shared data and the transaction's attempt does not produce a conflict and moreover, the transaction has not conflicted with others, the transaction accesses the data residing in database.

[4] When T_j's write operation conflicts with T_i's read operation and $L(T_j) < L(T_i)$, T_j records the effect of the write operation into the write-buffer.

[5] Although a transaction can try write operation without a conflict with other transactions, it records the effect of the write operation into the write-buffer instead of the database.

[6] In case that T_i's operation and T_j's operation conflict with each other and $L(T_i) = L(T_j)$, the conflict is handled according to the strict 2PL scheme.

[7] When T_i's read operation has been delayed due to the conflict with T_j's write operation, T_i executes the delayed read operation when T_j commits without distinction of security levels. In this case, T_i reads from write-buffer instead of database.

[8] The repeatable read operation gets the data from the same place that has been accessed by its previous read or write operations. For instance, if a transaction wrote data into write-buffer, it's repeatable read gets the data from write-buffer.

End of Algorithm. ■

5: Proof of Correctness

In this section, correctness of algorithm CIS is proved with the fact that once transactions are scheduled by CIS, they are serializable. For the convenient description, following definitions are needed.

Definition 8:

A **history H** indicates the order in which the operations of the transactions were executed relative to each other. If transaction T_i specifies the order of two of its operations, these two operations must appear in that order in any history that include T_i. □

Definition 9:

A history H is **serializable** if its committed projection is equivalent to a serial history H. Committed projection is a history over the set of committed transactions in H. □

Definition 10:

Let $T = \{T_i, ..., T_n\}$. The **serialization graph (SG)** for H, denoted SG(H), is a directed graph whose nodes are the transactions in T that are committed in H and whose edge are all $T_i \rightarrow T_j$ ($i \neq j$)such that one of T_j's operations precedes and conflicts with one of T_j's operations in H. □

Theorem 1:

A history H is serializable if SG(H) is acyclic.

***Proof*:**

Proved in [1]. □

Theorem 2:

History of transactions that have been scheduled by CIS is serializable.

***Proof*:**

We will prove the correctness of CIS by showing that scheduling the transactions with the features presented in section 4.1 does not lead to cycle in SG(H). For case 1, according to the definition 10, an edge $T(H) \rightarrow T(L)$ is added into SG(H). When T(H) tries to read, since it always gets the value from database and the effects of T(L) in write-buffer are never written into database until T(H) commits, T(H) never reads the value written by T(L). Therefore, $T(L) \rightarrow T(H)$ cannot be added into SG(H). Consequently, there is no cycle for case 1.

For case 2, since T(L)'s W(X) is executed after T(H)'s R(X), $T(H) \rightarrow T(L)$ is added into SG(H). However the second read operation of T(H) gets the value that has already been got by the first read operation of T(H), so that $T(L) \rightarrow T(H)$ cannot be added into SG(H). Consequently, there is no cycle for case 2.

For case 3,4,5,6, they obey the strict 2PL scheme. When SG(H) contains the edges that have been produced by the strict 2PL scheme, there is no cycle in SG(H). This is proved in [1].

Therefore, SG(H) that contains the edges produced by CIS is acyclic. According to the theorem 1, the history of transactions that are scheduled by CIS is serializable. □

6: Conclusions

In this paper, a concurrency control scheme in MLS/DBMS called conflict-insensible scheduling (CIS) has been proposed. Covert channel analysis is one of the most difficult challenges in MLS/DBMS. CIS prevents covert channels by hiding the conflicts that occurred between high security level transactions and low security level transactions.

The major contribution of CIS is that it does not give precedence to low security level transactions, and thus high security level transaction does not suffer from unfairness. Most of the past works prevent the covert channel by giving precedence to low security level transactions. Therefore, CIS is expected to get a better performance than past works.

The covert channels appeared in this paper are restricted within the covert channels that occur due to the data conflicts. Further research on preventing covert channels will deal with the covert channels that occur in data security violation. The conventional security models, such as Bell-LaPadula model, are sufficient to prove that security will not be violated through data accesses. However, when the security models are correlated with the conflicts of concurrent transactions, data security violation may easily occur. Therefore, reconsideration of the Bell-LaPadula model will be included in future researches.

Works on the performance evaluation for CIS will be continued. In addition, recovery management for CIS and the extended CIS that will be applicable to multiversion concurrency control scheme are remained for further works.

References

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

[2] Boris Kogan and Sushil Jajodia, "Concurrency Control in Multilevel Secure Databases based on Replicated Architecture, " Proceedings of the International Conference on Management of Data, ACM SIGMOD, 1990, pp. 153 - 162.

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

[4] Iwen E. Kang and Thomas F. Keefe, "Recovery Management for Multilevel Secure Database System," DATABASE SECURITY, VI : Status and Prospects, ed. Bhavani M.Thuraisingham, Carle. Landwehr, Elsevier Science Publishers B.V., 1993, pp. 225 - 247.

[5] Simon R. Wiseman, "On the Problem of Security in Databases," DATABASE SECURITY, III : Status and Prospects, ed. David L. Spooner, Carl Landwehr, Elsevier Science Publishers B.V., 1990, pp. 301 - 310.

[6] Marshal D. Abram and Gray W. Smith, "A Generalized Framework for Database Access Controls," DATABASE SECURITY, IV : Status and Prospects, ed. Sushil Jajodia, Carl E. Landwehr, Elsevier Science Publisher B.V. 1991, pp. 171 - 177.

[7] 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.

[8] 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.

[9] 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.

[10] 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

[11] Myong H. Kang, Oliver Costich, and Judith N, Froscher, "A Practical Transaction Model and Untrusted Transaction Manager for a Multilevel-Secure Database System," DATABASE SECURITY, VI : Status and Prospects, ed. Bhavani M. Thuraisingham, Carl E. Landwehr, Elsevier Science Publishers B.V., 1993, pp. 285 - 300.