# 다단계 보안 데이터베이스에서 동시성 제어를 위한 양방향 기부 잠금 규약

김 희 완† · 이 혜 경†† · 김 응 모†††


## 요 약

본 논문에서는 다단계 보안 데이터베이스에서 동시성 정도를 향상시키고, 보안 요구사항을 만족하는 향상된 트랜잭션 스케쥴링 프로토콜을 제안한다. 전통적인 직렬성 표기를 가진 두단계 잠금 기법을 다단계 보안 데이터베이스에 적용했다. 이타적 잠금기법은 기부라는 사상을 사용하여 트랜잭션이 더 이상 그 객체를 요구하지 않을 때 다른 트랜잭션들이 그 객체를 로크할 수 있도록 미리 객체에 대한 로크를 해제함으로써 트랜잭션들의 대기시간을 줄이기 위해서 제안된 것이다. 확장형 이타적 잠금기법은 처음에 기부되지 않는 객체까지도 처리하는 좀 더 완화된 기법이다. 본 논문에서는 다단계 보안 데이터베이스에서 단기 트랜잭션의 기아현상을 최소화하도록 하였다. 본 프로토콜은 확장형 잠금 기법(XAL/MLS)을 기초로 하였으나, 새로운 방법인 다단계 보안 데이터베이스를 위한 양방향 기부 잠금 규약(2DL/MLS)으로 보안 요구와 동시성 제어를 동시에 만족한다. 제안된 프로토콜의 효율성은 실험의 결과로 확인되었다.


**키워드 : 이타적 잠금기법, 동시성 제어, 양방향 기부 잠금 기법, 다단계 보안 데이터베이스**

# A Two-way Donation Locking Protocol for Concurrency Control in Multilevel Secure Database

Hee-Wan Kim† · Hae-Kyung Rhee†† · Ung-Mo Kim†††


## ABSTRACT

In this paper, we present an advanced transaction scheduling protocol to improve the degree of concurrency and satisfy the security requirements for multilevel secure database. We adapted two-phase locking protocol, namely traditional syntax-oriented serializability notions, to multilevel secure database. Altruistic locking, as an advanced protocol, has attempted to reduce delay effect associated with lock release moment by use of the idea of donation. An improved form of altruism has also been deployed for extended altruistic locking (XAL). This is in a way that scope of data to be early released is enlarged to include even data initially not intended to be donated. We also adapted XAL to multilevel secure database and we first of all investigated limitations inherent in both altruistic schemes from the perspective of alleviating starvation occasions for transactions in particular of short-lived nature for multilevel secure database. Our protocol is based on extended altruistic locking for multilevel secure database (XAL/MLS), but a new method, namely two-way donation locking for multilevel secure database (2DL/MLS), is additionally used in order to satisfy security requirements and concurrency. The efficiency of the proposed protocol was verified by experimental results.


**Key word : Altruistic Locking, Concurrency Control, Two-way Donation Locking, Multilevel Secure Database**

## 1. Introduction

A Multilevel secure database is a secure system which is shared by users from more than one clearance levels and contains data of more than one sensitivity levels [1]. When the database scheduler use the scheduling protocol to multilevel secure database, it must satisfy both the concurrency and the security requirements at the same time.

A data item's correctness is guaranteed by standard transaction scheduling schemes like *two-phase locking* (2PL)[6] for the context of concurrent execution environment. We adapted two-phase locking protocol to multilevel secure database. But when short-lived transactions are normally mixed with long-lived ones, degree of concurrency might be hampered by selfishness associated with lock retention. In 2PL, lazy release of lock could aggravate fate of misfortune for long-lived ones in that they are more vulnerable to get involved in deadlock situations. And short-lived transactions suffer from starvation or livelock affected by long-lived ones. To reduce the degree of livelock, the idea of altruism has been suggested in the

† 정 회 원 : 삼육대학교 컴퓨터과학과 교수
†† 정 회 원 : 경인여자대학 멀티미디어정보전산학부 교수
††† 종신회원 : 성균관대학교 전기전자및컴퓨터공학부 교수
논문접수 : 2000년 12월 7일, 심사완료 : 2001년 1월 15일

literature. *Altruistic locking* [4], *AL* for short, is basically an extension to 2PL in the sense that several transactions may hold locks on an object simultaneously under certain conditions. Such conditions are signaled by an operation *donate*. Like yet another primitive *unlock*, donate is used to inform the scheduler that further access to a certain data item is no longer required by a transaction entity of that donation. The basic philosophy behind *AL* is to allow long-lived transactions to release their locks early, once it has determined a set of data to which the locks protect will no longer be accessed. *Extended altruistic locking* [4], *XAL* for short, attempted to expand the scope of donation in a way that data to be early disengaged is augmented by extra data originally not conceived to be rendered. Our protocol is based on extended altruistic locking (XAL) but a new method, namely two-way donation locking for multilevel secure database (2DL/MLS), is additionally used in order to satisfy security requirements and concurrency in multilevel secure database.

## 2. Related Work

### 2.1 Multilevel Secure Database

Each data item in multilevel secure database is labeled with its security classification and each user is assigned a clearance level. In example, we will use the following hierarchical levels ordered as follows :

**Top Secret ≥ Secret ≥ Confidential ≥ Unclassified**

We applied the security models using Bell and LaPadula model [1] to multilevel secure database. Information is allowed to flow from an object(subject) with security classification level $l_1$ to a subject (object) with classification level level $l_2$ only if $l_2 \geq l_1$.

The BLP model requires that the system satisfy the following properties[2].

Simple Security Condition

A subject may have read access to an object only if the subject's classification level dominates the object's sensitivity level.

*-Property (Star Property)

A subject may have write access to an object only if the object's sensitivity level dominates the subject's classification level.

We used ts, s, c and u to denote the hierarchical level for subject(transaction) and object(data item) orderly in this paper.

### 2.2 Altruistic Locking

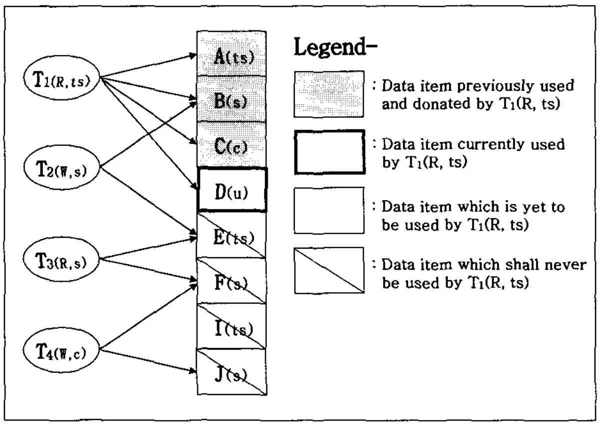
A transaction consists of database accesses and concurrency control operation, such as Lock and Unlock. It is well known that schedules of well-formed two-phase transactions that observe this rule are correct[3]. 2PL protocols ensure that these conditions are met, they produce serializable schedules. *AL* is a modification to *2PL* under certain conditions. *AL* provides a third concurrency control operation, called Donate, along with Lock and Unlock. Donate operation is used to inform the scheduler that access to an object is no longer required by the locking transaction. Donate and Lock operations need not be two-phase, i.e., when Donate is used, the donating transaction is free to continue to acquire new locks. Several rules control the use of the Donate operation by well-formed transactions. Transactions can only donate objects which they currently have locked. However, they may not access any object that they have donated. A donate operation is not a substitution for unlock operation. A well-organized transactions must unlock every object that it locks, regardless of whether it donated any of those objects. Transactions are never required to donate any objects ; donations are always optional. Donate operations are beneficial since they can permit other transactions to lock the donated object before it is unlocked.

### 2.3 Applying Extended Altruistic Locking to MLS

While the donation of wake is rigid in *AL* in terms of fixedness of its size, a dynamic way of forming a wake could be devised given that serializability is never violated. This was realized in *XAL* by simply letting data originally not intended to bestow to be dynamically included in a wake predefined. The rule is that wake expansion comes true only after a short transaction has already accessed data in its predefined wake list. So, the presumption made for *XAL* is that a short transaction still restlessly wishes to access data of its wake-dependent long transaction even after it has done with data in its wake list. The assumption could be called data-in-wake-list-first/other-data-later access fashion. *XAL* therefore performs inevitably badly if others-first wake-later access paradigm is in fact to be observed. Example 1 shows this.

Example 1(Delay Effect Caused by Donation Extension) : Suppose that *T1*(*R*, *ts*) attempts to access data items, *A(ts)*, *B(s)*, *C(c)* and *D(u)*, orderly in multilevel secure database.

Note that data items, *E(ts)*, *F(s)*, *I(ts)*, and *J(s)* shall not be accessed by *T1(R, ts)* at all. Presume that *T1(R, ts)* has already locked and successfully donated *A(ts)*, *B(s)* and *C(c)*. *T1(R, ts)* now is supposed in the stage of accessing D(u). Suppose also that there are three more transactions concurrently in execution along with *T1(R, ts)* : *T2(W, s)* wishing for *B(s)* and *E(ts)*, *T3(R, s)* wishing for *E(ts)* and *F(s)*, and *T4(W, c)* wishing for *F(s)* and J(s)(Figure 1).




(Figure 1) Four Transactions Competing for Same Data Donated

In case XAL/MLS, If *T2(W, s)* initially requests *E(ts)* first rather than *B(s)*, *T2(W, s)* can certainly acquire *E(ts)* but it fails for *B(s)* because wake relationship cannot honor *E(ts)* as a member of the wake list. Once this sort of wake dependency is detected, *T2(W, s)* can be allowed to access *B(s)* only after it is finally released by *T1(R, ts)*. *T2(W, s)* in this case is therefore blocked. *T3(R, s)* must then be blocked for *E(ts)* to be released by *T2(W, s)*. *T4(W, c)* as well must be blocked for *F(s)* to be released by *T3(R, s)*, forging a chain of blockage. End of Example 1.

To resolve this sort of chained delay, others-first wake-later approach could be made viable in a way of including others, not honored before, to a wake list. This enhancement is one of substances, made in our proposed protocol, which could be considered as *backward donation*, compared to *XAL*, which is based on *forward donation*. One other major substance of our proposed protocol is to let more than one long transaction donate while serializability is preserved in multilevel secure database. The notion of *two-way donation locking with multilevel secure database* is thus developed in our protocol. Our protocol allows more donation than one long transaction, but for the sake of presentation simplicity, degree of donation is limited to two in this paper.

## 3. Proposed Protocol

### 3.1 Assumptions

To describe wake expansion rule in detail, simplifications were made mainly with regard to transaction management principle.

① (*Transaction Operation*) : All transactions have either read or write operation to their data items.
② (*Security Policy*) : A transaction and its data items follow MAC policy by the Bell and LaPadula model.
③ (*Donation Privilege*) : Only long-lived transactions are privileged to use donate operation.
④ (*Commit Policy*) : A long-lived transaction eventually commits.
⑤ (*Deadlock Handling*) : If a transaction happens to fall into deadlock situation, that transaction will be eliminated by using a certain deadlock timeout scheme.

In this paper, the multiplicity is rendered to the case of two to measure the effect of donation variety. Two- way donation locking protocol with Multilevel Secure Database, *2DL/MLS* for short, can be pseudo-coded as follows (Algorithm Wake Expansion).

```
Algorithm(Wake Expansion Rule of 2DL/MLS)
Input:LT1; LT2; ST
 /* ST:short trans; LT1, LT2:long trans */
BEGIN
 FOREACH LockRequest
   IF(LockRequest.ST.data = Lock)
   THEN
 /* Locks being requested by ST already granted to long trans other
    than LT1 and LT2 */
    Reply:=ScheduleWait(LockRequest);
    ELSE IF(LockRequest.ST.data = Donated) THEN
 /* Locks being requested by ST donated by long trans other than
    LT1 and LT2 */
    FOREACH (ST.wake LT1 OR LT2)
     IF(ST.wake = LT1) THEN
 /* Donation conducted by LT1? */
      IF(ST.data LT1.marking-set) THEN
 /* Data being requested by ST to be later accessed by LT1 ? */
       Reply:=ScheduleWait(LockRequest)
      ELSE
       Reply:=SecurityCheck(LockRequest)
      ENDIF
    ELSE
    IF(ST.data LT2.mark5ing-set) THEN
 /* Data being requested by ST to be later accessed by LT2 ? */
       Reply := ScheduleWait(LockRequest)
    ELSE
       Reply := SecurityCheck(LockRequest)
    ENDIF
  ENDIF
  ENDFOR
```

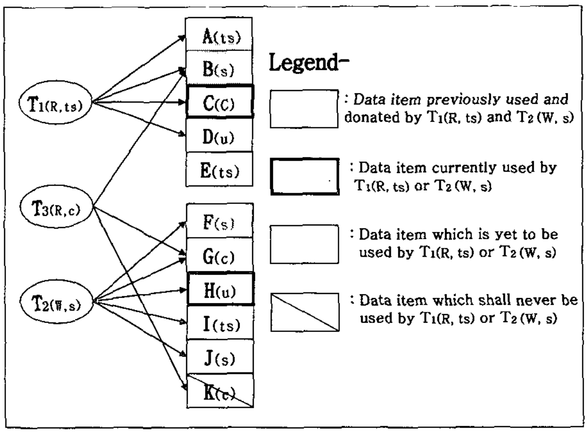
```
ELSE
      Reply := SecurityCheck(LockRequest)
 ENDIF
  IF(Reply = Abort) THEN
/* Lock request of ST aborted */
   Abort Transaction(Transactionid);
   Send(Abort);
   Return();
   ENDIF
 ENDFOR
END
SecurityCheck(TRAN, DATA, GUBUN)
/* TRAN:transaction to be transferred : DATA:data item to be transferred */
BEGIN
 IF(TRAN.R = True) THEN /* Simple-property (Read Option) */
  IF( TRAN.level Data.level ) THEN /* Transaction's level check */
   IF( GUBUN = Lock ) THEN
         Reply := ScheduleLock(LockRequest)
    ELSE
         Reply := ScheduleDonated(LockRequest)
    ENDIF
   ELSE /* No read up */
       Reply := DiscardData(LockRequest)
   ENDIF
ELSE /* *-property(Write Option) */
   IF( TRAN.level Data.level ) THEN /* transaction level check */
    IF( GUBUN = Lock ) THEN
         Reply := ScheduleLock(LockRequest)
    ELSE
         Reply := ScheduleDonated(LockRequest)
    ENDIF
   ELSE /* No write down */
       Reply := DiscardData(LockRequest)
   ENDIF
ENDIF
END
```

### 3.2 Operation Instance of 2DL/MLS

In case donated data items are used under *XAL/MLS*, it is allowed to request data items which are donated by only one transaction. Under *2DL/MLS*, in contrast, short-lived transactions are treated to be given more freedom in accessing donated objects by eliminating the single-donation constraint. Short-lived transactions can access objects donated by two different long-lived transactions in multilevel secure database.

*2DL/MLS* permits short-lived transactions request data items which have been donated by two different long-lived transactions. A way to conduct a two-way donation is shown, in Example 2, with two separate long transactions and a single short transaction.

Example 2(Allowing Proceeding of Short Transaction with Two Concurrent Long Ones in Multilevel Secure Database) : Suppose that *T1(R, ts)*, a long transaction with Read/Top-secret secure level, attempts to access data items, *A(ts)*, *B(s)*, *C(c)*, *D(u)* and *E(ts)*, orderly in multilevel secure database. Presume that *T1(R, ts)* has already locked and successfully donated *A(ts)* and *B(s)*. *T1(R, ts)* now is supposed in the stage of accessing *C(c)*. Suppose also that there are two more concurrent transactions in execution along with *T1(R, ts)* : *T2(W, s)*, long transaction, wishing for data items, *F(s)*, *G(c)*, *H(u)*, *I(ts)* and *J(s)*, in an orderly manner and *T3(R, c)*, short with low level, wishing for *B(s)*, *G(c)* and *K(u)* similarly. Presume that *T2(W, s)* has already locked and successfully donated *F(s)* and skipped *G(c)* due to *-property in BLP model. *T2(W, s)* now is supposed in the stage of accessing *H(u)* (Figure 2).




(Figure 2) Execution of T3 with Two Concurrent Long-Lived Transactions

If we apply *XAL/MLS* for these transactions, a lock request for *B(s)* by *T3(R, c)* would be allowed to be granted but a lock request *G(c)* would not because *G(c)* has already been donated by another long-lived transaction. Only after *T2(W, s)* commits, *G(c)* can be tossed to *T3(R, c)*.

In case *2DL/MLS*, *T3(R, c)* could fortunately be allowed to access without any delay. This is made possible by simply including the wake of *T2(W, s)* into the wake of *T1(R, ts)*. End of Example 2.

### 3.3 Correctness of 2DL/MLS

In this section, we will show that 2DL/MLS satisfy both serialization and security requirement. To do so, we will make use of the serializability theorem [3], the definition of Crest Before [4] and a lemma used in proving the correctness of AL [4]. The serializability theorem states that a history H is serializable iff its serialization graph is acyclic, and the definition of Crest Before state that for two transactions, say Ti uTj if Ti unloaks some data items before Tj locks some data items.

We use $o_i[x]$, $p_i[x]$ or $q_i[x]$ to denote the execution of either read or write operation issued by a transaction $T_i$, on a data item x. Reads and writes of data items are denoted by $r_i[x]$ and $w_i[x]$, respectively. Locking operation is also represented by $ol_i[x]$, $pl_i[x]$, $ql_i[x]$, $rl_i[x]$ or $wl_i[x]$. Unlock and donate operations are denoted by $u_i[x]$ and $d_i[x]$, respectively. H represents a history which may be produced by *2DL/MLS* and O(H) is a history obtained by deleting all operations of aborted transactions from H. The characteristics of histories which may be produced by *2DL/MLS* are as follows.

**Property 1(Two-Phase Property)** : If $ol_i[x]$ and $u_i[y]$ are in O(H), $ol_i[x] < u_i[y]$.

**Property 2(Lock Property)** : If $o_i[x]$ is in O(H), $ol_i[x] < o_i[x] < u_i[x]$.

**Property 3(Donate Property)** : If $ol_i[x]$ and $d_i[x]$ is in O(H), $o_i[x] < d_i[x]$.

**Property 4(Unlock Property)** : If $d_i[x]$ and $u_i[x]$ is in O(H), $d_i[x] < u_i[x]$.

**Property 5(Security Property)** : If level($T_i$) level($r_i[x]$) in O(H), $rl_i[x] < u_i[x]$, If level($T_i$) level($w_i[x]$) in O(H), $wl_i[x] < u_i[x]$.

**Property 6(Indebtedness Property)** : If $T_j$ is indebted to $T_i$ for every $o_j[x]$ in O(H), either $o_j[x]$ is in the wake of $T_i$ or there exists $u_i[y]$ in O(H) such that $u_i[y] < o_j[x]$.

**Lemma 1(Altruism)** : If $p_i[x]$ and $q_j[x]$ $(i \neq j)$ are conflicting operations in O(H) and $q_i[x] < q_j[x]$, then $u_i[x] < ql_j[x]$ or $d_i[x] < ql_j[x]$.

Proof : A data item must be locked before and unlocked after it is accessed by Property 1. In Wake Expansion Rule of *2DL/MLS*, a conflict lock on the data item, say $a$, is allowed only when no transaction locks $a$ or the transactions which hold locks on $a$ has donated it. Thus, the history, O(H), satisfies Lemma 1. End of Lemma 1.

**Lemma 2(Complexity-In-Wake)** : If $T_1 \rightarrow T_2$ is in serialization graph, then either $T_1 \rightarrow_u T_2$ or $T_1 \rightarrow_d T_2$.

Proof : $T_1 \rightarrow T_2$ in serialization graph means that there exist conflicting operations, say $p_1[x]$ and $q_2[x]$, in H such that $p_1[x] < q_2[x]$. There are only two cases that may occur for this by Lemma 1. One is that there is $p_1[x] < d_1[x] < ql_2[x] < q_2[x]$ in O(H), i.e., $T_2$ accesses the data items donated by $T_1$.

A transaction $T_2$ has to access only wake of another transaction $T_1$ , once $T_2$ makes conflict locks on the data items donated by $T_1$. $T_2$ must be completely in the wake of $T_1$ if $T_2$ has accessed any of the wake of $T_1$. This is ensured by the first else if condition in algorithm. Even if $T_2$ has already accessed any data items which do not belong to the wake of $T_1$ , such data items would be included into the wake of $T_1$ as long as $T_1$ does not access any of such data items at all for its execution. If the data items locked by $T_2$ will be accessed by $T_1$, the access of $T_2$ to the data items donated by $T_1$ is not allowed by the second foreach condition. Thus, $T_1 \rightarrow T_2$ corresponds to $T_1 \rightarrow_d T_2$ in the case that $p_1[x] < d_1[x] < ql_2[x] < q_2[x]$ in H, or in the case that $p_1[x] < u_1[x] < ql_2[x] < q_2[x]$ in O(H) by Lemma 1. Thus, $T_1 \rightarrow T_2$ corresponds to $T_1 \rightarrow_u T_2$ in the case.

End of Lemma 2.

**Lemma 3(Correctness of *AL*)** : Consider a path $T_1 \rightarrow \cdots T_{n-1} \rightarrow T_n$ in O(H). Either $T_1 \rightarrow_u T_2$, or there exists some $T_i$ on the path such that $T_1 \rightarrow_u T_i$.

Proof : We will use induction on the path length n. By Lemma 2, the lemma is true for n = 2. Assume the lemma is true for paths of length n-1, and consider a path of length n. By the inductive hypothesis, there are two cases :

① There is a $T_I$ between $T_1$ and $T_{n-1}$ such that $T_1 \rightarrow_u T_k$. The lemma is also true for paths of length n.

② $T_1 \rightarrow_d T_{n-1} \rightarrow T_n$ and $T_{n-1}$ conflicts on at least one object, x. Since $T_{n-1}$is completely in the wake of $T_1$, we must have $d_1[x] < ql_{n-1}[x]$ in O(H). By Property 1, $T_n$ must lock x. By Property 4, $T_1$ must unlock x. Either $u_1[x] < ol_n[x]$ or $ol_n[x] < u_1[x]$. In the first case, we have that $T_1 \rightarrow_u T_n$, i.e., $T_n$ is the $T_k$ of the lemma. In the second case, $T_n$ is indebted to $T_1$. By Property 6, $T_n$ is completely in the wake of $T_1(T_1 \rightarrow_d T_n)$ or $T_1 \rightarrow_u T_n$.

**Theorem 1(Serializability of *2DL/MLS*)** : If O(H) is acyclic, O(H) is serializable and satisfies security rules.

Proof : Assume that there exists a cyclic $T_1 \rightarrow \cdots T_{n-1} \rightarrow T_n$ in serialization graph. By Lemma 3, $T_1 \rightarrow_d T_1$, or $T_1 \rightarrow_u T_i$. By Property 3, only $T_1 \rightarrow_u T_i$ is possible. By Property 5, $T_i$ in H satisfies security property. Since $T_i$ is prohibited to lock any more data items once $T_1$ unlocks any one, $T_i$ cannot be $T_1$. Again, by applying Lemma 3 to the same cycle $T_1 \rightarrow T_{i+1} \rightarrow \cdots T_i$ , we get $T_i \rightarrow_u T_k$.for the same reason and thus we get $T_1 \rightarrow_u T_i$ $_u T_k$ in all. Since the relation $_u$ is transitive, $T_1 \rightarrow_u T_k$ is satisfied. Thus, $T_k$ cannot be any of $T_1$ and $T_i$. If we are allowed to continue to apply Lemma 3 to the given cycle n-3 times more in this manner, we will get a path $T_1 \rightarrow_u T_{iu} \rightarrow T_k \rightarrow_u \cdots \rightarrow_u T_m$ containing all transactions, i.e., $T_1$ through $T_n$. If we apply Lemma 3 to the given cycle starting from $T_m$ one more time, we are enforced to get a cycle $T_l \rightarrow_u T_i \rightarrow_u T_k \rightarrow_u \cdots \rightarrow_u T_m \rightarrow_u T_l$ and we get a contradiction of violating Property 1 or Lemma 3. Thus serialization graph is acyclic and by the serializability theorem O(H) is serializable and satisfies security rules. End of Theorem 1.

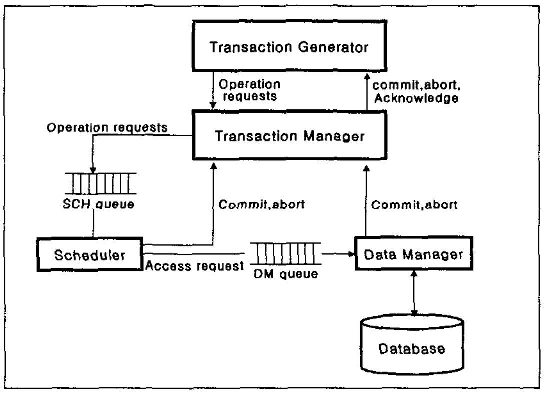
## 4. Performance Evaluation

### 4.1 Simulation Model

#### 4.1.1 Queuing System Model

The simulation model, in (Figure 3), consists of subcomponents in charge of fate of a transaction from time of inception to time of retreat : *transaction generator* (TG), *transaction manager*(TM), *scheduler* (SCH), *data manager*(DM), *database*(DB).

TG generates user transactions one after another and sends their operations to TM one at a time in a way of interleaving. TM receives transactions from terminals and passes them SCH queue.




(Figure 3) Simulation Model

DM analyzes an operation from SCH to determine which data item the operation is intended to access, and then sends the operation to the disk where the requested data item is stored. Whenever an operation is completed at the server, it sends to TM the message informing that the requested operation has been completed successfully.

This simulation model has been implemented using *Scheme* [5] discrete-event simulation(DEVS) language. In DEVS formalism one must specify basic models from which larger ones are built, and describe how these models are connected together in hierarchical fashion[7].

#### 4.1.2 Experimental Methodology

<Table 1> summarizes the model parameters and shows the range of parameter values used in our experiments. Values for parameters were chosen by reflecting real world computing practices.

<Table 1> Parameters Setting for Simulation

| Parameters | Values |
|---|---|
| *db_size* | 100 |
| *num_cpus* | 2 |
| *num_disks* | 4 |
| *short_tran_size* | 2, 3, 4 |
| *long_tran_size* | 5, 6, 7, 8, 9 |
| *tran_creation_time* | 2 units |
| *sim_leng* | 100, 300, 500, 700, 900, 1100, 1300, 1500 |

To see performance tradeoff between *2PL/MLS* and *2DL/MLS*, average transaction length represented by number of operation in transaction were treated to vary. The shortest one is assumed to access 20 percent of the entire database, while it is 80 percent for the longest one.

The number of CPUs and disks, *num_cpus* and *num_disks*, are set to 2 and 4, respectively. The idea behind this status of balance by 1-to-2 ratio has been consulted from[6].

### 4.2 Simulation Results and Interpretations

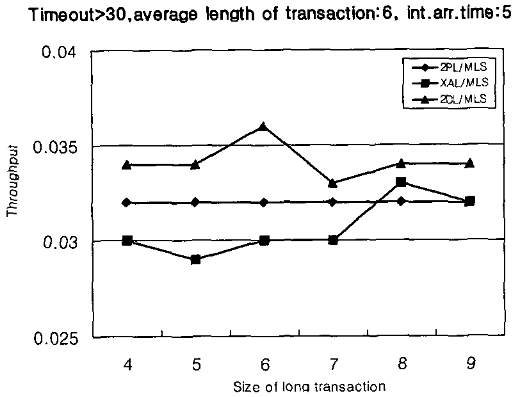
#### 4.2.1 Effect of Security Requirement Level

This experiment has been revealed that 2DL/MLS satisfied the security requirement by Bell and LaPadula model. We have counted the processing ratio data item which satisfy the security requirement against total ones. Each transaction has Read/Write option, four clearance level, and data items which they process. Each data items have four sensitivity levels. If the transaction satisfy the security requirement which it wish to process the data item, it process the data item the next time slice. Otherwise, the transaction discards the data item, and it remains the current time slice of operating system. In this experimental, the entire processing ratio was 61.4 percent. So this model satisfies the security requirement by BLP model.

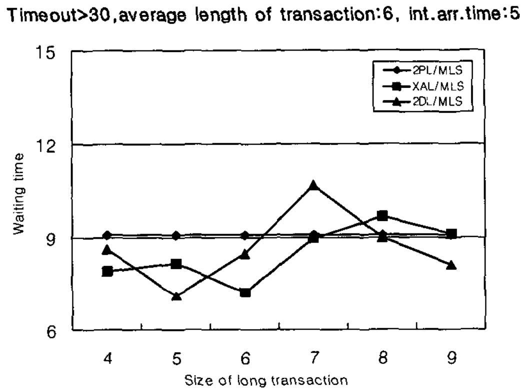
#### 4.2.2 Effect of Multiprogramming Level

This experiment shows that *2DL/MLS* generally appears to outperform *2PL/MLS* in terms of average waiting time. The best throughput performance is also exhibited by *2DL/MLS* and the worst average waiting time is portrayed by *XAL/MLS*.

Performance gain of *2DL/MLS* against *2PL/MLS* is from 103 to 113 percent increment in terms of throughput. And *2DL/MLS* outperforms *2PL/MLS* from 99 to 78 percent decrease of performance at transaction waiting time except long transaction size is 7. This is because *2DL/MLS* has the 2PL/MLS plus the donation of data items of long transaction.



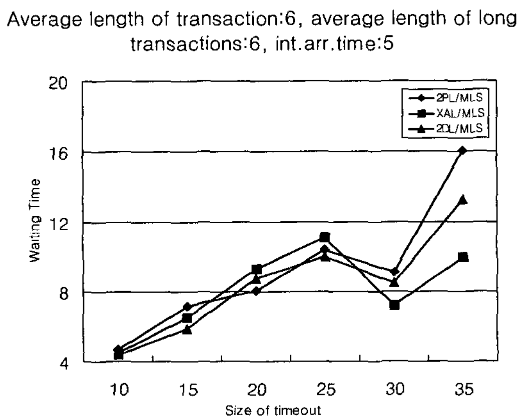

(Figure 4) Throughputs




(Figure 5) Average Waiting Time

4.2.3 Effect of Timeout

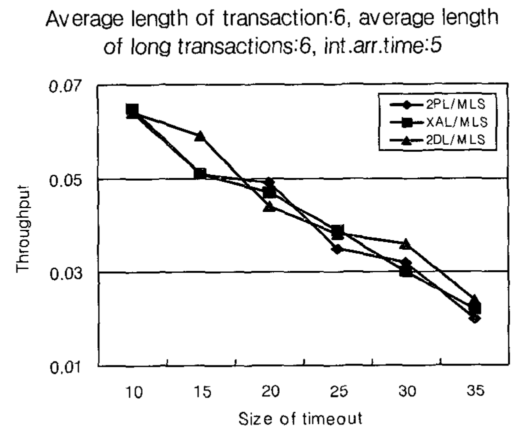
At a higher range of *timeout*, *2DL/MLS* shows a higher throughput and a medium transaction waiting time for three schemes. Throughputs of *2PL/MLS* and *XAL/MLS* show the same value from timeout size 10 through 35.




(Figure 6) Throughputs with Longer Timeout

Throughput of *2DL/MLS* outperforms *XAL/MLS* and *2PL/MLS* when timeout size is 15, 30 or 35. We can observe that average waiting time curve of 2PL/ MLS rapidly increase from 30 to 35 in (Figure 10). As 2DL/MLS's result, this phenomenon again shows us higher throughput gives lower average waiting time.

*2DL/MLS* performs better than *2PL/MLS* between 100 percent and 120 percent of performance at transaction throughput. If the timeout size is far extended beyond a certain point, say 30, the average waiting time curve of *2PL/MLS* increase than other two schemes. *2DL/MLS* outperforms *2PL/MLS* with 82.55 percent of performance at transaction waiting time when the timeout size is 35.




(Figure 7) Average Waiting Time with Longer Timeout

Overall behaviors have been revealed that as the size of timeout increases, *2DL/MLS* generally outperforms in terms of throughput and waiting time. This shows a possibility that performance gain of *2PL/MLS* against *2DL/MLS* could be deteriorated sharply if the timeout size is far extended beyond a certain point, say 30.

## 5. Conclusions

In this paper we proposed that the two-way donation locking for multilevel secure database(2DL/MLS) is a protocol improving concurrency control and satisfying the security requirements. 2DL/MLS showed a more satisfying performance compared to any other scheme methods, and in multilevel secure database when Long-lived transaction lead to abort overhead, 2DL/MLS is recommended to improve the concurrency degree for wireless mobile network environment. *2DL/MLS* is considered to be a practical solution to take where short-lived transactions quickly access database without any delay by long-lived ones for multilevel secure database.

Our protocol in this paper is limited to the BLP model for multilevel secure database. As part of our future work, we would like to prevent covert channels by ensuring that transactions at lower security levels are never delayed by the actions of a transaction at a higher security level.

## References


[1] T. F. Keefe, W. T. Tsai and J. Srivastava, "Multilevel Secure Database Concurrency Control," Data Engineering, Proceedings. Sixth International Conference on, 1990.

[2] D. E. Bell, and L. J. LaPadula, "Secure Computer Systems : Unified Exposition and Multics Interpretations," T echnical Report MTR-2997, Mitre Corp., March 1976.

[3] P. A. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems," Addison-Wesley, Massachusetts, U.S.A., 1987.

[4] K. Salem, H. Garcia-Molina and J. Shands, "Altruistic Locking," ACM Transactions on Database Systems, Vol.19, No.1, pp.117-169, March 1994.

[5] H. Bartley, C. Jensen and W. Oxley, "Scheme User's Guide and Language Reference Manual," Texas Instruments, Texas, U.S.A., 1988.

[6] R. Agrawal, M. J. Carey and M. Linvy, "Concurrency Control Performance Modeling : Alternative and Implications," AC M Transactions on Database Systems, Vol.12, No.4, pp. 609-654, December 1987.

[7] Zeigler, B. P., "Object-Oriented Simulation with Hierarchical, Modular Models : Intelligent Agents and Endomorphic Systems," Academic press, San Diego, CA, 1990.



### 김 희 완

e-mail : hwkim@syu.ac.kr

1987년 광운대학교 전자계산학과 졸업(이학사)

1995년 성균관대학교 정보공학과 졸업(공학석사)

1999년 성균관대학교 전기전자 및 컴퓨터공학부 박사과정 수료

1996년 정보처리 기술사(정보관리 부문) 취득

1996년 삼육의명대학 전산정보과 조교수

2001년 삼육대학교 컴퓨터과학과 조교수

관심분야 : DB보안, 동시성제어, 분산DB, Mobile Computing



### 이 혜 경

e-mail : rheehk@dove.kyungin-c.ac.kr

1979년 숭실대학교 전자계산학과 졸업(공학사)

1985년 미국 일리노이대학교(Urbana-Champaign) 전산학과 졸업(공학석사)

2000년 성균관대학교 정보공학과 졸업(공학박사)

1988년 국립 천안공업전문대학 전자계산과 전임강사

1993년~현재 경인여자대학 멀티미디어정보전산학부 조교수

관심분야 : 동시성제어, 분산DB, DB보안, Mobile Computing



### 김 응 모

e-mail : umkim@yurim.skku.ac.kr

1981년 성균관대학교 수학과 졸업(이학사)

1986년 Old Dominion University 전산학과 졸업(공학석사)

1990년 Northwestern University 전산학과 졸업(공학박사)

1990년~현재 성균관대학교 전기전자 및 컴퓨터공학부 교수

관심분야 : DB보안, 데이터마이닝, 웹DB, 공간DB, 동시성제어