A Secure Dynamic Copy Protocol in Real-Time Database Systems

Soo-Yeon Park, Sungyoung Lee, Byung-Soo Jeong
Department of Computer Engineering, Kyung Hee University, Seoul, Korea
sylee@oslab.kyunghee.ac.kr

Hyon Woo Seung
Department of Computer Science, Seoul Women's University, Seoul, Korea
hwseung@cs.swu.ac.kr

Abstract

Concurrency control for real-time secure database systems must satisfy not only logical data consistency but also timing constraints and security requirements associated with transactions. These conflicting natures between timing constraints and security requirements are often resolved by maintaining several versions (or secondary copies) on the same data items. In this paper, we propose a new lock-based concurrency control protocol, Secure Dynamic Copy Protocol, ensuring both conflicting requirements. Our protocol aims for reducing the storage overhead of maintaining secondary copies and minimizing the processing overhead of update history. The main idea of our protocol is to keep a secondary copy only when it is needed to resolve the conflicting read/write operations in real time secure database systems. For doing this, a secondary copy is dynamically created and removed during a transaction's read/write operations. While comparing the existing real-time security protocol, we have also examined the performance characteristics of our protocol through simulation under different workloads. The results show that our protocol consumed less storage and decreased the deadline missing transactions.

1. Introduction

While conventional database systems must consider data consistency, high performance and quick response time, database systems for real-time applications require not only logical data consistency but also timing constraints associated with transactions[3][4]. Multi-level secure database systems are shared by concurrent transactions with different clearance levels and manage data objects with different classification levels[10]. Most of the multi-level secure database systems are based on

the Bell-LaPadula model[2], which imposes the following restrictions on all data accesses:

1) Simple Security Property: Read access is allowed if the transaction's clearance is identical to or higher than the data object's classification.
2) Restricted *- Property: Write access is allowed if the transaction's clearance is identical to the accessed data object [6].

It is also required in such a system to remove covert channels. Although security models prevents direct flow of information from a higher access class to a lower access class, information in a high-priority transaction may flow into a low-priority transaction through covert channels which are not designed in the system. Covert channels arise when conflicts occur among concurrent transactions with different security levels. They may be generated as the concurrent transactions share the same working areas (storage channel), or by measuring the time used for the common resources (timing channel). Multi-level secure database systems must be designed to avoid such covert channels, and security protocols in the systems must guarantee low-level transactions are not delayed or aborted by high-level transactions[9].

Mukkamala and Son's Secure Real-Time 2 Phase Locking (SRT-2PL)[1] is based on the idea of preventing covert channels from arising by maintaining the property of noninterference among transactions with different security levels. Two kinds of data objects are used to provide noninterference in SRT-2PL. One is the primary copy which is the target object for read and write operations of equivalent level transactions. The other is the secondary copy which is for read operations of high level transactions. This protocol, however, has a disadvantage of memory waste since it has to maintain copies of all data objects, and it also causes some extra cost since it has to use and manage an update queue to update data copies. Furthermore, the lack of predictability always exists since there can be high-level transactions waiting for the contents in the update queue to be updated on the secondary copy.

In order to overcome such problems, this paper proposes a dynamic copy protocol which prevents covert channels from arising by maintaining noninterference, and consumes less memory than the SRT-2PL by reducing the time to keep copies. The proposed model, which is based on the Bell-LaPadula model, dynamically manages zero or more copies for each data object and remarkably reduces the time for keeping copies by generating them only if needed and deleting them otherwise. Since all the copies can be accessed by maintaining them in a list, the unpredictability problem of high-level transactions' read-down operations does not exist.

The paper is organized as follows. In Section 2, related works are reviewed. The dynamic copy protocol is proposed in Section 3. In Section 4, the correctness issues are

discussed for the proposed algorithm. The performance evaluation is carried out by comparing it with the SRT-2PL in Section 5. Finally, Section 6 summarizes the results and future work.

2. Related Research

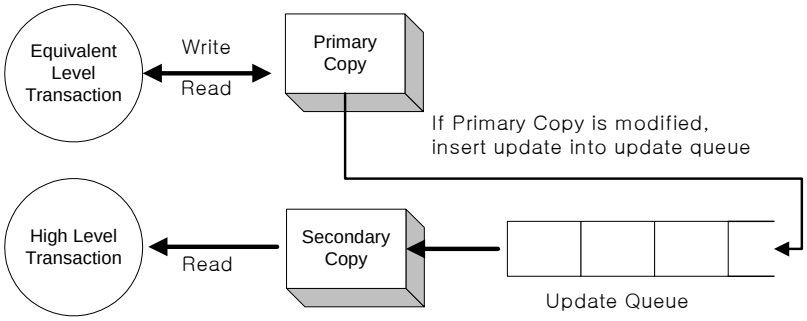
The real-time concurrency control protocol and the multi-level security concurrency control protocol have been studied independently. The former has been researched on the basis of the locking strategies or optimistic methods, including priority based scheduling approaches[3]. Typical examples are High Priority Algorithm, Conditional Restart Algorithm, OPT-SACRIFICE, and OPT-20. The latter protocol, whose research is mainly based on the locking or timestamping methods and single/multiple versioning techniques, aims to maintain noninterference among transactions with different security levels.

Recently, several research projects have reported on the concurrency control protocol for the multi-level security real-time database systems. David[7] proposed the feedback control mechanism which prevents information flow by restricting the capacity of covert channels based on the information theory. This method inputs, in advance, a capacity limit of the covert channel and an ideal deadline missing rate, and keeps adjusting the two values according to the actual capacity and rate monitored during the system execution, not only to meet the real-time requirements but also to avoid the covert channels. It, however, guarantees only the limited capacity of covert channels but not the deadline missing rate[7].

The basic principle behind the secure 2 Phase Locking (2PL) proposed by Son[8] is to try to simulate execution of basic 2PL without blocking the lower access class transactions by higher access class transactions. The blocking occurs when a transaction with an access class identical to the data object (say $\boldsymbol{T_L}$) asks for a write operation, in which case a transaction with an access class higher than the data object (say $\boldsymbol{T_H}$) holds a lock on the shared data. The secure 2PL allocates a virtual lock to T_L and allows it to execute the write operation, without delay, on the local area. Once the lock is released, the virtual lock is changed to a real lock, and an actual write operation is executed. A deadlock may occur with the protocol, but it can be resolved using a set of transactions, before ($\boldsymbol{T}$) and after ($\boldsymbol{T}$), that must be finished before and after some transaction T, respectively. It requires, however, that serializability be guaranteed and subordinate locks be maintained in order to prevent a virtual lock from being allocated without control by a low-level transaction[8].

Mukkamala and Son[1] introduced SRT-2PL which prevents covert channels by

maintaining the property of noninterference among transactions with different security levels. Since a covert channel arises when a high-level transaction aborts or delays a low-level transaction, the SRT-2PL removes possible conflicts by separating transactions with an access class identical to the data object from higher access class transactions. The protocol works as shown in Figure 1. Each data object has a primary and a secondary copy. The primary copy of an object is accessed (read/write) by transactions at the same security level as the object. The secondary copy is accessed (read) by transactions at a security level higher than the data object. Since the secondary copy is only for read operations, it cannot be updated by itself. The update queue contains updates that have been performed on the primary copy but have yet to be performed on the secondary copy. The secondary copy is updated when it does not hold a read-lock. This protocol, however, must maintain an update queue in addition to the two copies of data objects. Furthermore, it lacks the predictability of high-level transactions since they have to wait for contents in the queue to be updated on the secondary copy.



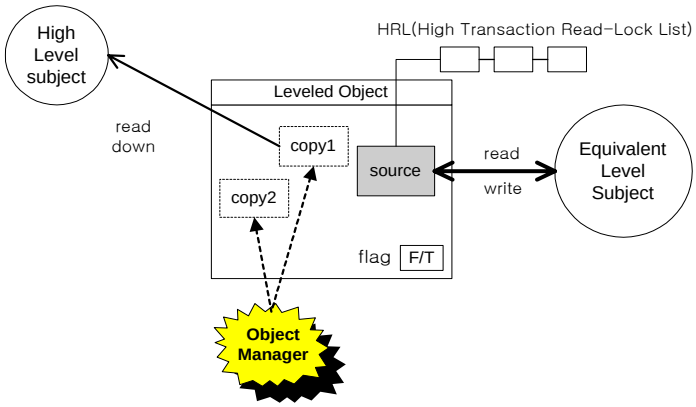
[Figure 1] SRT-2PL

3. Dynamic Copy Protocol

3.1 System Model

The Dynamic Copy Protocol proposed in this paper is an extension of SRT-2PL, and, therefore, works in a similar way to SRT-2PL. As in SRT-2PL, the secondary copy is for read operations of high-level transactions, and the primary copy is for read/write operations of equivalent level transactions. In this protocol, however, each data object may have more than one secondary copy, and a secondary copy is generated when a write operation is requested. A high-level transaction may read the primary copy if there is no write-lock on the primary copy, while SRT-2PL forces the high-level transaction to read the secondary copy even if the primary copy does not hold a lock.

Figure 2 illustrates how the Dynamic Copy Protocol works. The Object Manager is responsible for the management of secondary copies, and the High Transaction Read-Lock List (HRL) is maintained since high-level transactions may read the primary copy. There is a flag which indicates the current state of the secondary copy.



[Figure 2] System Model for Dynamic Copy Protocol

The object manager creates a secondary copy when a transaction requests a write operation and deletes the copy if it is not read-locked when the transaction is finished. The copies are added to or deleted from a list, and a set of transaction identifiers, TIDs is maintained for the transactions which hold read-locks on secondary copies. HRL is a list of high-level transactions executing read operations on the primary copy. When a transaction at the same security level as the data object requests a write-lock, the read-locks on the HRL move onto the secondary copies so that the low-level transactions may not be delayed. If the flag is true, the value of the recently generated copy is the same as that of the primary copy, otherwise it is false. The object manager generates a copy only when the flag is false. It is initialized as false and changed to true after a copy is generated. After the write operation is finished, it is set to false again.

The following are the assumptions made in this protocol:

A1) Each transaction needs to procure all the required locks before starting its execution.

A2) Among transactions at the same security level as the data object, the basic 2PL is applied.

A3) Only transactions at a security level higher than the data object are allowed to move locks onto the secondary copy.

3.2 Rules of the Protocol

Covert channels can be prevented from arising by removing the possibilities of interference among transactions with different security levels. The Dynamic Copy Protocol guarantees noninterference among security levels by keeping high-level transactions from aborting or delaying low-level transactions. As shown in Table 1, operations of the protocol can be classified into 5 categories according to the requested operation, conditions of the lock, flag, and HRL.

[Table 1] Operations of Dynamic Copy Protocol

Condition			Request	Action
Lock	Flag	HRL		
none	×	null	×	grant a lock on source
R	×	null	R or RD	
R	×	not null	R or RD	
R	F	not null	W	generate a copy and grant a lock on source
R	T	not null	W	move lock held by transactions in HRL to copy
W	×	null	RD	grant a lock on copy
R	×	null	W	execute transaction on the priority basis (block/abort)
W	×	null	R	
W	×	null	W	

* R : read(lock), W : write(lock), RD : read-down, ×: inapplicable

The rules according to which our protocol manages its locks and operations are as follows:

[***Rule 1***] When a transaction T requests a read operation on a low-level data object ***x***, check if ***x*** is write-locked. If it is not, T is allowed to execute the read operation on the primary copy of ***x***. Otherwise, the read operation is executed on the recently generated secondary copy.

[***Rule 2***] When a transaction ***T*** requests a write operation on a data object ***x***, the object manager references the flag and generates a secondary copy if false. Then, the locks held by the transactions in the HRL are moved to the generated copy, and a write-lock is granted to T on the basis of the basic 2PL.

[***Rule 3***] When a transaction ***T*** releases the write-lock on ***x***, the object manager deletes the copy only if there is no read-lock on it.

[***Rule 4***] When a high-level transaction ***T*** releases the read-lock on ***x***, the object

manager checks if the transaction which generated the copy is finished and deletes the copy if so.

Figures 3 and 4 show two cases of pseudo-codes which represent how the dynamic copy protocol works according to the kinds of locks.

```
Case 1. Reauest Read-Lock
  1 : if( λ ;)<λ ⌜R) )
      // lock requester has higher level than holder
  2 :     if ( x has write-lock )
              execute read operation with copy and exit
  3 :     else { add TID to HRL
                 execute read operation with source }
  4 : else          // λ ;)=λ ⌜R)
          execute read operation with source by 2PL
```

[Figure 3] Pseudo-code of Dynamic Copy Protocol On Read-Lock Request

Case 1) When a read-lock is requested (Figure 3),
if the lock requester has a higher level than the data object (line 1) and the data object has a write-lock, then execute the read operation with the copy and exit (line 2). If the data object has no write-lock, add the lock requester's TID to HRL and execute the read operation with the source (line 3). If the lock requester is at the level of the data object, execute the read operation with the source according to the basic 2PL (line 4).

```
Case 2. Request Write-Lock
  1 : if ( flag == TRUE ) go to step 3
  2 : else  { create copy
              flag = TRUE }
  3 : subjects in HRL read copy
  4 : release lock source
  5 : execute write operation with source by 2PL
  6 : flag = FALSE;
```

[Figure 4] Pseudo-code of Dynamic Copy Protocol On Write-Lock Request

Case 2) When a write-lock is requested (Figure 4),
if the flag is false, create a new copy, insert it to the copy list, and set the flag to true (line 1). Then, move the locks held by the transactions in the HRL to the generated copy (line 2 and 3), and the lock requester executes write operation with the primary copy according to the basic 2PL (line 5). After the write operation, the flag is reset to false (line 6).

3.3 An Example Operation : SRT-2PL vs. Dynamic Copy Protocol

Let's take an example of the sequence of transactions input to a scheduler as shown in Figure 5 (the transactions arrived in the T_1, T_2, T_3, T_4 order). Small letters ***x*** and ***y*** are data objects at an UNCLASSIFIED level, and ***X*** is a SECRET level data object. ***R, r, w*** and ***C*** stand for Read-down, read, write, and Commit operations, respectively.

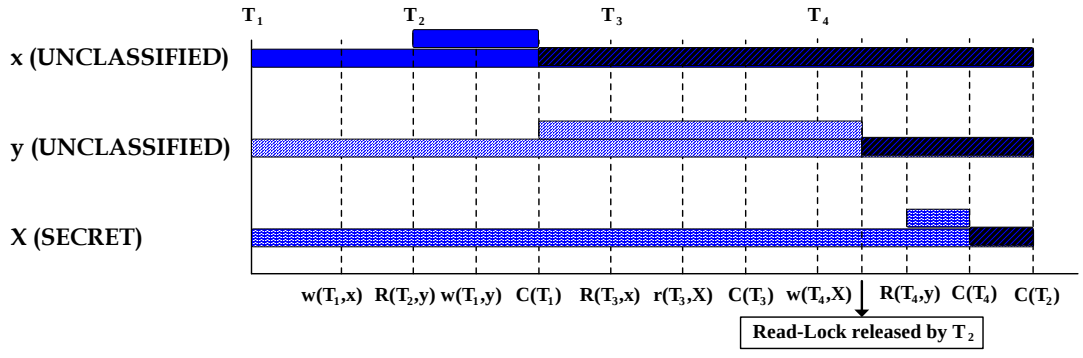
T1(UNCLASSIFIED):	***w[x] w[y]C***	
T2(SECRET)	***: R[y]***	***C***
T3(SECRET)	***: R[x]r[X]C***	
T4(SECRET)	***: w[X]R[y]C***	

[Figure 5] Example Scheduling

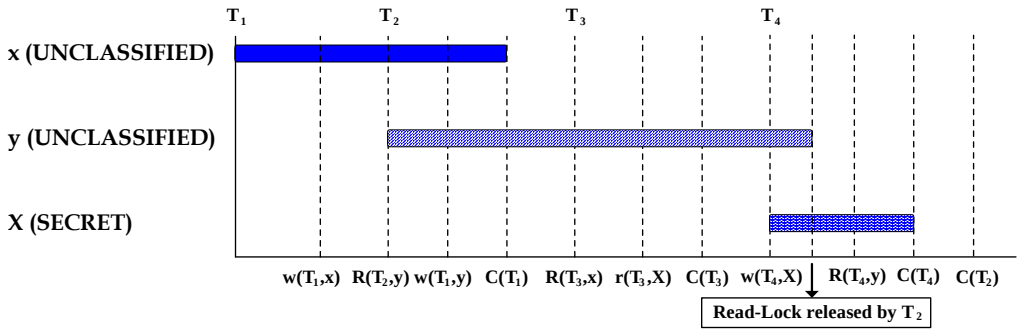
In SRT-2PL, a secondary copy is maintained for each data object ***x, y,*** and ***X*** until all the transactions are committed and every update is appended to the update queue and kept until it is reflected on the secondary copy. In the Dynamic Copy Protocol, the overall time to keep the copies is shorter since a secondary copy is generated only when a write operation is requested.

In SRT-2PL, the transaction T_2 reads the secondary copy of ***y*** since its level is higher than ***y***. T_4 also reads the secondary copy, but the operation must be delayed until the copy is updated (T_2 releases the read-lock), since it must read the value of ***y*** updated by T_1 in order to maintain the data consistency. Such a delay does not exist in the Dynamic Copy Protocol since a write operation is executed after a copy is generated and high-level transactions may read the primary copy.

The presence or absence of horizontal bars in Figures 6 and 7 show how secondary copies are generated/deleted during executions of the transactions scheduled by SRT-2PL and the Dynamic Copy Protocol, respectively. The slanted area in Figure 6 indicates that the secondary copy has been updated.



[Figure 6] Generation/Deletion of Copies when scheduled by SRT-2PL



[Figure 7] Generation/Deletion of Copies when scheduled by Dynamic Copy Protocol

In Figure 6, any read operation on x after $C(T_1)$ can be executed on the secondary copy since x is updated at the time of $C(T_1)$ when $w(T_1,x)$ is committed. The primary copy of y is updated by the operation $w(T_1,y)$, but the update is not reflected on the secondary copy nor appended to the update queue due to $R(T_2,y)$. $R(T_4,y)$ is granted a lock after T_2 releases its read-lock. Accordingly, read-down operations by high-level transactions may be unpredictable. In the case of the Dynamic Copy Protocol shown in Figure 7, T_4 reads-down the source regardless of the read-lock release by T_2. Since the source is not write-locked, T_4 is granted a read-lock and executes its operation without waiting. As soon as T_2 releases its lock, the secondary copy is deleted.

4. Proof of Serializability

In this section, we prove the correctness of the proposed protocol.

[**Lemma 1**] All transactions at a single security level are serializable.

Proof: Same as in [1]

[**Lemma 2**] All transactions committed by the Dynamic Copy Protocol are serializable.

Proof: In the Dynamic Copy Protocol, each transaction procures all the required locks before starting its execution. Therefore, it is not possible for any two transactions to have contradicting dependency relationships at different security levels.
Consider levels L and L' where L>L'. Since transactions at each level are serialized(by Lemma 1), let $T_1 \rightarrow T_2 \rightarrow \ ... \rightarrow T_{m-1} \rightarrow T_m$ be the serialization order at level L and $t_1 \rightarrow t_2 \rightarrow \ ... \rightarrow t_{n-1} \rightarrow t_n$ be the order at level L' of committed transactions.

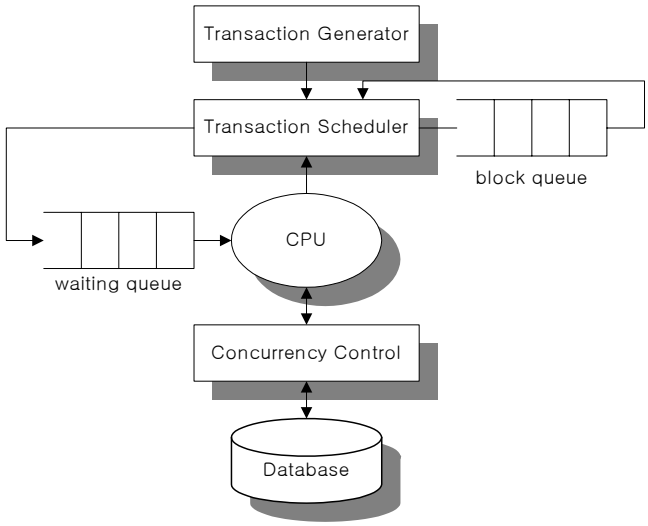
Suppose by way of contradiction that there is no serialization order among all these transactions. Then there is a cycle in the corresponding serialization graph so that one of the following two cases is possible:

Case 1) $T_i \rightarrow t_j \rightarrow t_k \rightarrow T_i$: Since only read-down operations are allowed for higher level transactions on lower level objects, $T_i \rightarrow t_j$ implies that there is a data object x (at level L') that was read by T_i and modified later by t_j. Similarly, $t_k \rightarrow T_i$ implies that there is a data object y (at level L') that was written by t_k and later read by T_i. Since all locks were obtained at the beginning of its execution, it had obtained locks on the secondary copies of x and y also at the beginning. Hence, $T_i \rightarrow t_j$ implies that t_j was committed after T_i started, and $t_k \rightarrow T_i$ implies that t_k was committed prior to T_i's start. From the above, we can conclude that t_k was committed before t_j. However, this contradicts our assumption that $t_j \rightarrow t_k$.

Case 2) $t_i \rightarrow T_j \rightarrow T_k \rightarrow t_i$: $t_i \rightarrow T_j$ implies that there is a data object x (at level L') that was modified by t_i and later was read by T_j. Similarly, $T_k \rightarrow t_i$ implies that there is a data object y (at level L') that was read by T_k and later modified by t_i. Using the same argument in case 1, it lead to a contradiction.
The same proof can be extended even when transactions from more than two levels are considered. Thus, all committed transactions by the Dynamic Copy Protocol are serializable. ■

5. Performance Evaluation and Comparisons

Figure 8 illustrates the simulation model designed for the performance evaluation of the Dynamic Copy Protocol and for the comparison of the protocol with the SRT-2PL. The transaction generator generates transactions according to the given factors. The transaction scheduler is responsible for scheduling the generated transactions and managing the block queue and waiting queue. A transaction in the waiting queue is transmitted to CPU and executed according to the given concurrent control protocol. Under the simulation model, the performance of both SRT-2PL and Dynamic Copy Protocol is evaluated by comparing the number of delayed or aborted transactions as well as the length of update queues and the number of copies, according to the size of the slack or transaction.



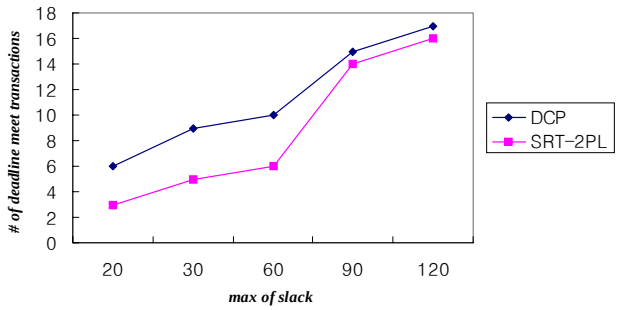
[Figure 8] Simulation Model

The experiments were performed under the single-processor and memory-based database environment in order to minimize other influences. The Earliest Deadline First (EDF) algorithm was used for the priority assignment of transactions. The simulation factors are given in Table 2.

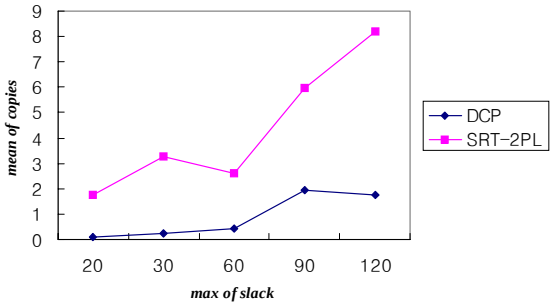
[Table 2] Simulation Factors

Factor	Value
execution Time (Read, Write)	3.0 ms
# of data objects	10
security level	3

In Figures 9 through 12, DCP stands for the Dynamic Copy Protocol. For DCP, the length of the copy list was considered as the number of copies, and one (secondary copy), added to the length of update queue, was used for SRT-2PL.

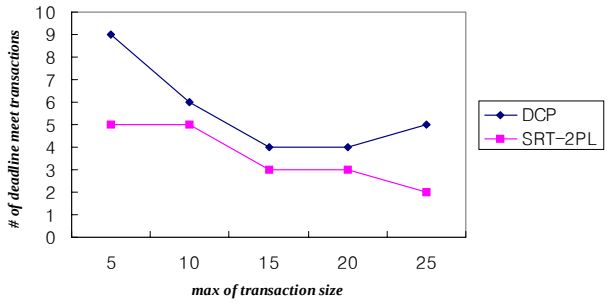


[Figure 9] Slack vs. Number of Deadline Met Transactions

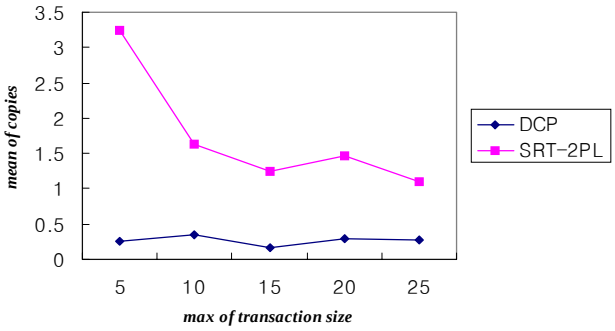


[Figure 10] Slack vs. Number of Copies

Figures 9 and 10 show how the number of deadline meeting transactions and average number of copies change, as the slack, which is the time left to the deadline, varies. The number of deadline meeting transactions increases as the slack becomes larger. The reason why SRT-2PL has a smaller number of deadline meeting transactions than DCP is because some transactions missed deadlines due to the delay of high-level transactions. The number of copies also increases as the slack becomes larger, since less transactions get aborted and more number of active transactions use copies. Many more copies are generated in SRT-2PL since it keeps a secondary copy even if the primary copy has no lock.



[Figure 11] Transaction Size vs. Number of Deadline Met Transactions



[Figure 12] Transaction Size vs. Number of copies

Figures 11 and 12 show how the number of deadline meeting transactions and average number of copies change as the number of operations in a transaction varies. The larger the size of the transaction becomes, the more the number of deadline meeting transactions decreases. The number of copies also decreases as the size of the transaction becomes larger, since more transactions get aborted and less number of active transactions use copies.

From the experimental results, DCP showed an average of 14% higher than SRT-2PL in the number of deadline meeting transactions, and an average of 80% less in memory usage. Such a remarkable difference in memory saving comes from the fact that DCP generates copies only when needed and allows high-level transactions to read the primary copy, if it is not write-locked, without generating secondary copies, while at the same time SRT-2PL has a secondary copy for each data object and the size of the update queue grows for every update.

6. Conclusion

In addition to maintaining data consistency, database systems for real-time applications must satisfy timing constraints associated with transactions. Multi-level secure database systems with real-time transaction processing requirements are called multi-level secure real-time database systems (MLS/RT DBMS). Such a system is required to meet deadlines to maintain data consistency, and to prevent covert channels. It is, however, not easy to design a system which satisfies all the criteria since there are trade-offs among them.

We proposed a novel dynamic copy concurrency control protocol that can meet the real-time, security, and serializability conditions. The proposed protocol is similar to SRT-2PL since both protocols have primary and secondary copies. However, it allows high-level transactions to perform read operations on the primary copy and dynamically generates and deletes secondary copies as needed. Consequently, it consumes less memory space and makes the read-downs more efficient as well as more predictable. It also prevents covert channels from arising by maintaining noninterference among transactions with different security levels. Our experimental results showed that the proposed protocol used much less memory and made more transactions met deadlines than SRT-2PL. The proposed protocol has some defects. Many copies must be maintained when many write operations are requested on a data object. We are currently in the process of improving our protocol.

References

[1] R. Mukkamala and S. H. Son, "A Secure Concurrency Control Protocol for Real-Time Databases", Annual IFIP WG 11.3 Conference of Database Security, Rensselaerville, New York, Aug. pp 235-253, 1995.

[2] Bell.D.E. and LaPadula.L.J, "Secure Computer Systems : Unified Exposition and Multics Interpretation", The Mitre Corp., 1976.

[3] Soek-Hee Hong, Myung-Ho Kim, Yoon-Joon Lee, "Methods of Concurrency Control in Real-Time Databse", Korean Information Science Society Review, Vol. 11, No. 1, pp 26-36, 1993.

[4] Abbott. R. K. and Garcia-Molin. H, "Scheduling Real-Time Transactions : A Performance Evaluation", ACM Transactions on Database Systems, 17, pp513-560, 1992.

[5] Young-kuk Kim and Sang H. Son, "Predictability and Consistency in Real-Time Database Systems", Advances in Real-Time Systems, S. H. Son (ed.), Prentice Hall, pp 509-531, 1995.

[6] Thomas F. Keefe, W.T.Tsai, Jaideep Srivastava, "Database Concurrency Control in Multilevel Secure Database Management Systems", IEEE Transaction on knowledge and Data Engineering vol 5, no. 6, pp1039-1055, 1993.

[7] Rasikan David, Sang H. Son and Ravi Mukkamala, "Supporting Security Requirements in Multilevel Real-Time Databases", IEEE Symposium on Security and Privacy, Oakland, CA, pp 199-210, 1995.

[8] S. H. Son and R. David, "Design and Analysis of a Secure Two-Phase Locking Protocol," 18th International Computer Software and Applications Conference (COMPSAC'94), Taipei, Taiwan, pp 374-379, 1994.

[9] Ira S. Moskowitz and Myong H. Kang, "covert channels - Here to Stay?", Proc. COMPASS 94, pp235-243, 1994.

[10] Chanjung Park, Seog Park, "Multiversion Concurrency Control Protocol with Freezing Method in Multilevel Secure Database Systems", Journal of KISS(B): Software and Applications, Vol. 25, No. 8, pp.1159-1169, 1998.