Design and Analysis of A Secure Two-Phase Locking Protocol

Sang H. Son and Rasikan David

Department of Computer Science
University of Virginia
Charlottesville, VA 22903, USA

Abstract

In addition to maintaining consistency of the database, secure concurrency control algorithms must be free from covert channels arising due to data conflicts between transactions. The existing secure concurrency control approaches are unfair to transactions at higher access classes. In this paper, a secure two-phase locking protocol is presented, which is correct and free from covert channels. The protocol uses three different types of locks to support non-interference property and to provide reasonably fair execution of all transactions, regardless of their access class. The results of a performance evaluation of the protocol are provided, comparing it with secure optimistic concurrency control and secure multiversion timestamp ordering.

1. Introduction

Database security is concerned with the ability of a database management system to enforce a security policy governing the disclosure, modification or destruction of information. Most secure database systems use an access control mechanism based on the Bell-LaPadula model [Bell 76]. This model is stated in terms of subjects and objects. An object is understood to be a data file, record or a field within a record. A subject is an active process that requests access to objects. Every object is assigned a classification and every subject a clearance. Classifications and clearances are collectively referred to as security classes (or levels) and they are partially ordered. The Bell-LaPadula model imposes the following restrictions on all data accesses:

a) *Simple Security Property*: A subject is allowed read access to an object only if the former's clearance is identical to or higher (in the partial order) than the latter's classification.

This work was supported in part by ONR, by CIT and by IBM

b) *The *-Property*: A subject is allowed write access to an object only if the former's clearance is identical to or lower than the latter's classification.

The Bell-LaPadula model prevents direct flow of information from a higher access class to a lower access class, but the conditions are not sufficient to ensure that security is not violated indirectly through what are known as covert channels [Lamp 73]. A covert channel allows indirect transfer of information from a subject at a higher access class to a subject at a lower access class. An important class of covert channels that are usually associated with concurrency control mechanisms are timing channels. A timing channel arises when a resource or object in the database is shared between subjects with different access classes. The two subjects can cooperate with each other to transfer information.

Concurrency control is used in databases to manage the concurrent execution of operations by different subjects on the same data object such that consistency is maintained. A concurrency control algorithm is said to be *fair* if the same level of service is provided to all classes of transactions, where the definition of service varies with the application. For a real-time system, it could signify the guaranteeing of a deadline, in a conventional database system it could be the response time for each transaction, etc. In multilevel secure databases, there is the additional problem of maintaining consistency without introducing covert channels. In this paper, a secure two phase locking protocol is presented and its performance is evaluated, comparing it with secure optimistic concurrency control and secure multiversion timestamp ordering. There have been various other approaches to secure concurrency control [Thur 93], but we restrict our comparison to the three canonical cases. We do not consider approaches that make use of specialized architectures [Koga 90], or that assume weaker correctness criteria [Jajo 92].

Covert channel analysis and removal is the single most important issue in multilevel secure concurrency control. The notion of *non-interference* has been proposed [Gogu 82] as a

0730-3157/94 $04.00 © 1994 IEEE

simple and intuitively satisfying definition of what it means for a system to be secure. The property of *non-interference* states that the output as seen by a subject must be unaffected by the inputs of another subject at a higher access class. This means that a subject at a lower access class should not be able to distinguish between the outputs from the system in response to an input sequence including actions from a higher level subject and an input sequence in which all inputs at a higher access class have been removed [Keef 90a].

Locking will fail in a secure database because the security properties prevent actions in a transaction T_1 at a higher access class from delaying actions in a transaction T_2 at a lower access class (e.g. when T_2 requests a conflicting lock on a data item on which T_1 holds a lock). Timestamp ordering fails for similar reasons, with timestamps taking the role of locks, since a transaction at a higher access class cannot cause the abortion of another transaction at a lower access class.

Optimistic concurrency control for a secure database can be made to work by ensuring that whenever a conflict is detected between a transaction T_h at a higher access class in its validation phase and a transaction T_l at a lower access class, the transaction at the higher access class is aborted, while the transaction at the lower access class is not affected. A major problem with using optimistic concurrency control is the possible *starvation* of higher-level transactions. For example, consider a long-running transaction T_h that must read several lower-level data items before the validation stage. In this case, there is a high probability of conflict and as a result, T_h may have to be rolled back and restarted an indefinite number of times.

A secure version of the MVTO scheduler is presented in [Keef 90b]. The difference between basic MVTO and secure MVTO is that secure MVTO will sometimes assign a new transaction a timestamp that is earlier than the current timestamp. This effectively moves the transaction into the past with respect to active transactions. To be more precise, when a transaction begins, it is assigned a timestamp that precedes the timestamps of all transactions active at strictly dominated access classes and that follows the timestamps of all transactions at its own access class. This approach to timestamp assignment is what makes it impossible for a transaction to invalidate a read from a higher access class.This method has the drawback that transactions at a higher access class are forced to read arbitrarily old values from the database due to the timestamp assignment. This problem can be especially serious if most of the lower level transactions are long running transactions.

It is easy to see that the problems with any concurrency control mechanism are present because a higher level transaction cannot interfere with the execution of a lower level transaction. This makes the mechanism unfair, because higher level transactions are starved for service. The secure two phase locking protocol presented in this paper does not eliminate the problem of starvation for higher level transactions, but is fairer than the other existing secure concurrency control algorithms. Basic two phase locking does not work for secure databases because a transaction at a lower access class (say T_l) cannot be blocked due to a conflicting lock held by a transaction at a higher access class (T_h). If T_l were somehow allowed to continue with its execution in spite of the conflict, then non-interference would be satisfied. This is the basic principle behind the secure two phase locking protocol.

2. A Secure Two-Phase Locking Protocol

The system maintains two lists for each transaction T_i - *before*(T_i) which is the list of active transactions that precede T_i in the serialization order and *after*(T_i) which is the list of active transactions that follow T_i in the serialization order. The following additions are made to the basic two phase locking protocol:

1) When an action $p_i[x]$ sets a virtual lock on x because of a real lock $ql_j[x]$ held by T_j, then T_i and all transactions in *after*(T_i) are added to *after*(T_j), T_j and all transactions in *before*(T_j) are added to *before*(T_i).

2) When an action $w_i[x]$ arrives and finds that a previous action $w_i[y]$ (for some data item y) has already set a virtual write lock $vwl_i[y]$, then a dependent lock $dvwl_i[x]$ is set with respect to $vwl_i[y]$.

3) When an action $p_i[x]$ arrives and finds that a conflicting dependent lock $dvql_j[x]$ has been set by a transaction T_j which is in *after*(T_i), then $p_i[x]$ is allowed to set a lock on x and perform $p_i[x]$ in spite of the conflicting lock.

4) A dependent virtual lock $dvp_i[x]$, dependent on some action $q_i[y]$ is upgraded to a virtual lock when $vql_i[x]$ is upgraded to a real lock.

The semantics of the three different kinds of locks are explained below:

1) *Real Lock (of the form $pl_i[x]$):* A real lock is set for an action $p_i[x]$ if no other conflicting action has a real lock or a virtual lock on x. The semantics of this lock are identical to that of the lock in basic two phase locking.

2) *Virtual Lock (of the form $vpl_i[x]$)*: A virtual lock $vpl_i[x]$ is set for an action $p_i[x]$ if a transaction at a higher access class holds a conflicting lock on x ($p_i[x]$ has to be a write to satisfy the Bell-LaPadula properties). The virtual lock is non-blocking. Once a virtual lock $vpl_i[x]$ is set, $p_i[x]$ is added to *queue[x]* and the next action in T_i is ready for scheduling. When $p_i[x]$ gets to the front of queue, its virtual lock is upgraded to a real lock and $p_i[x]$ is submitted to the scheduler. No other conflicting lock (a read lock or a write lock by another transaction) can be set until $vpl_i[x]$ is upgraded to $pl_i[x]$ and subsequently released.

3) *Dependent Virtual Lock (of the form dvpli[x])*: A dependent virtual lock is set for an action $p_i[x]$ (where p is a write) if a previous write $w_i[y]$ in the same transaction holds a virtual lock. An action $p_i[x]$ which holds a dependent virtual lock with respect to another action $w_i[x]$ is not allowed to set a real lock or a virtual lock unless $w_i[y]$'s virtual lock is upgraded to a real lock. The dependent lock is non-blocking too. An action $p_i[x]$ holding a dependent lock cannot be superseded by an action at a lower access class. However, a conflicting action $q_j[x]$ at a higher access class can be granted precedence over $dvpl_i[x]$ if T_j is in *before*(T_i).

2.1. The Protocol

The rules according to which the secure 2PL scheduler manages its locks are as the following:

```
When an action $p_i[x]$ is submitted, one of the following four
possible cases can arise:

Case 1 ($p_i[x]$ is a read and $wl_i[x]$ or $vwl_i[x]$ or $dvwl_i[x]$
        have already been set):
    Read temporary value of x written by $T_i$;

Case 2 ($p_i[x]$ is a write and $rl_i[x]$ has already been set):
    Upgrade $rl_i[x]$ to $wl_i[x]$;

Case 3 ($p_i[x]$ conflicts with some $ql_j[x]$ holding a lock on x
        or waiting to set a lock on x):
    if ($p_i[x]$ is a write and for some data item z, $vwl_i[z]$ or
      $dvwl_i[z]$ has already been set)
        Set $dvpl_i[x]$;
        Let $w_i[z]$ be the most recent action before $p_i[x]$ in
        $T_i$ that holds a virtual lock or a dependent virtual
        lock:
            Mark $p_i[x]$ as dependent on $w_i[z]$ and add
            $p_i[x]$ to queue[x];
    else if (sec_level($T_i$) < sec_level($T_j$))
        Set $vpl_i[x]$;
    else
        Add $p_i[x]$ to queue[x];
        Add $T_i$ and after($T_i$) to after($T_j$);
        Add $T_j$ and before($T_j$) to before ($T_i$);

Case 4 (None of the above cases are satisfied):
    Set $pl_i[x]$; Submit $p_i[x]$ to the Data manager;

Let X be the set of data items for which locks are released
when a transaction $T_i$ completes;
for every data item x in X do
{
    while queue[x] not empty
    {
        Let $p_i[x]$ be the first action on queue[x];
        if a conflicting lock $ql_j[x]$ has been set
        then
            exit loop;
        Remove $p_i[x]$ from queue[x];
        if ($p_i[x]$ is a write and $p_i[x]$ is marked virtual)
        then
            Upgrade $vpl_i[x]$ to $pl_i[x]$;
            Submit $p_i[x]$ to the Data Manager;
            for every lock $dvq_i[z]$ dependent on $pl_i[x]$
            {
                if ($q_i[z]$ is the first action in queue[z] and
                    $q_i[z]$ does not conflict with a lock $q_j[z]$
                    that has been set)
                then
                    Set $ql_i[z]$;
                    Submit $q_i[z]$ to the Data Manager;
                else
                    Upgrade $dvql_i[z]$ to $vql_i[z]$;
                    Mark $q_i[z]$ as virtual in queue[z];
            }
        else if ($p_i[x]$ is marked real)
            Set $pl_i[x]$; Submit $p_i[x]$ to the Data Manager;
    }
}
```

In addition, the following two conditions are to be satisfied by the transaction manager:

1) A transaction is allowed to commit, even if virtual writes performed by the transaction have not been committed to stable storage, i.e the writes are still on some queue[x] waiting to set a real lock on x.

2) Once an action $p_i[x]$ acquires a lock on a data item, the lock is not released until T_i commits and $p_i[x]$ is performed on the data item x in stable storage. Therefore, in most cases, all the locks that a transaction holds are not released when it commits.

2.2. Correctness

In this section, we prove the correctness of the Secure 2PL protocol. Its correctness rests on the fact that a conflict can arise between an action $p_i[x]$ at a higher access class and an action $q_j[x]$ at a lower access class if and only if p is a read and q is a write (by the Bell-LaPadula properties). To close all covert channels, this is the only case that needs to be treated differently from that of the strict two phase locking protocol [Bern 87].

Define a history H to be a partial order of operations, with ordering relation $<_H$ that represents the execution of a set of transactions, such that for all conflicting operations $p,q \in H$, either $p <_H q$ or $q <_H p$. The serialization graph for H, denoted by $SG(H)$, is a directed graph whose nodes are committed transactions in H and whose edges are all of the form $T_i \rightarrow T_j\ (i \neq j)$ such that one of T_i's operations precedes and conflicts with one of T_j's operations in H. To prove a history H serializable, we only have to prove that $SG(H)$ is acyclic [Bern 87]. For the purpose of the proof, we also define a binary relation *before* on transactions, such that T_i *before* T_j if T_i occurs before T_j in the serialization order. For convenience of expression, we also represent an action $p_i[x]$ waiting for a lock in lock queue *queue[x]* as $pw_i[x]$.

To prove the correctness of Secure 2PL, we characterize the set of all Secure 2PL histories, that is, those that represent possible executions of transactions that are synchronized by a 2PL scheduler. We include the Set Real Lock, Set Virtual Lock, Set Dependent Lock, Waiting for Lock and Unlock operations into histories, since examining the order in which these operations were processed will help establish the order in which Reads and Writes are executed. This, is turn, will be used to prove that the *SG* of any history produced by Secure 2PL is acyclic. The proof outline is as follows:

- Characterize the Secure 2PL histories by listing all the orderings of operations that must hold and how the serialization order is maintained.

- Prove that the *before* relation is anti-symmetric, i.e., if T_i *before* T_j then T_j is not *before* T_i. From this fact, it is an obvious corollary that the serialization graph SG of any history is acyclic.

Proposition 1: Let H be a history produced by a Secure 2PL scheduler. For every operation $o_i[x]$ in H, $dvol_i[x] < vol_i[x] < ol_i[x] < o_i[x] < ou_i[x]$ or $ow_i[x] < ol_i[x] < o_i[x] < ou_i[x]$.

The virtual lock and the dependent virtual lock are not set by all actions, i.e., one or the other or both could be missing from the history.

Proposition 2: Let H be a history produced by a Secure 2PL scheduler. If $p_i[x]$ and $q_j[x]\ (i \neq j)$ are conflicting operations in H and $pl_i[x] < q_j[x]$ or $vpl_i[x] < q_j[x]$ or $dvpl_i[x] < q_j[x]$ or $pw_i[x] < q_j[x]$ and T_j is not *before* T_i, then T_i *before* T_j is set to TRUE.

Proposition 2 enumerates all possible cases where a *before* relation between two transactions could be enabled.

Proposition 3: If T_j *before* T_i is TRUE, then there do not exist actions $p_i[x]$ in T_i and $q_j[x]$ in T_j such that $ql_j[x] < pl_i[x]$ or $ql_j[x] < vpl_i[x]$ or $ql_j[x] < dvpl_i[x]$ *or* $ql_j[x] < pw_i[x]$.

Proposition 3 implies that once T_j is set before T_i in the serialization order, no action in T_i can be performed before an action in T_j, i.e., T_i *before* T_j can never be true at any subsequent point in time. This result leads directly to Proposition 4.

Proposition 4: The relation *before* is anti-symmetric.

Note that there is a one to one correspondence between the *before* relations across multiple transactions and the serialization graph. Therefore, in the serialization graph *SG*, for any pair of transactions, T_1 and T_2, a path $T_1 \rightarrow T_2 \rightarrow T_1$ cannot be present. Extending the argument to n transactions, suppose $SG(H)$ had a cycle $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow \ldots \rightarrow T_n \rightarrow T_1$, i.e., for each transaction T_i, $2 \leq i \leq n$, T_1 *before* T_i and T_i *before* T_1, which contradicts the proposition that the *before* relation is anti-symmetric.Therefore, the serialization graph of any history H generated by the Secure 2PL protocol is serializable.

2.3. Deadlock Detection and Resolution

The secure two phase locking protocol is not free from deadlocks. For example, consider the input schedule below:

T_1 (SECRET)	:	*r[x]*		*r[y]*	
T_2 (UNCLASSIFIED)	:		*w[y]*		*w[x]*

The sequence of operations performed are $rl_1[x]\ r_1[x]\ wl_2[y]\ w_2[y]\ vwl_2[x]\ c_2$. After these operations, deadlock would occur because $r_1[y]$ waits for $w_2[y]$ to release its lock and $vw_2[x]$ waits for $r_1[x]$ to release its lock. Deadlocks can be detected by constructing a waits-for graph [Bern 87]. A scheduler recovers from a deadlock by aborting one of the transactions in the cycle. A secure scheduler must ensure that recovery security is not violated by the choice of a victim [Keef 90a]. This can be guaranteed by aborting a transaction at the highest access class in the cycle detected in the waits-for graph.

3. Performance Evaluation

Much research has been done into the development of concurrency control algorithms for secure databases [Thur 93]. In this section, we present the results of our performance comparison of the Secure 2PL protocol with two other secure concurrency control protocols. The two main goals of our performance analysis are:

- To obtain a measure of fairness across security levels - this can be done with the help of two measures - *response time* and *staleness* at different access classes for the various concurrency control algorithms.
- To determine the cost (in terms of overhead) of providing security. This is achieved by comparing the response times of Secure 2PL with those of Basic 2PL.

3.1 Simulation Model

Central to the simulation model is a single-site disk resident database system operating on shared-memory multiprocessors [Lee 94]. The system consists of a disk-based database and a main memory cache. The unit of database granularity is the *page*. When a transaction needs to perform an operation on a data item it accesses a page. If the page is not found in the cache, it is read from disk. CPU or disk access is through an M/M/k queueing system, consisting of a single queue with *'k'* servers (where *'k'* is the number of disks or CPUs).

3.2 Summary of Results

For each experiment, we ran the simulation with the same parameters for 6 different random number seeds. Each simulation run was continued until 200 transactions at each access class were committed. For each experiment the required performance measure was measured over a wide range of workload. All the data reported in this paper have 90% confidence intervals, whose endpoints are within 10% of the point estimate. Due to space limitation, we only present the summary of major results. The following conclusions can be drawn from the simulation study results:

- The Secure 2PL protocol exhibits a much better response time characteristic than Secure OCC. Its operating region (the portion of the curve before the saturation point) is much larger than that of Secure OCC (Figure 1). In addition, even after the saturation point, the response time for Secure 2PL does not increase as rapidly as it does for Secure OCC. This fact is not apparent from Figure 1, which concentrates only on the region close to the saturation point, but is borne out by the performance results for arrival rates greater than that at which saturation takes place.
- Secure MVTO has the best response time and throughput characteristics. If staleness is not an issue or if average transaction sizes are small, Secure MVTO may be the secure concurrency control algorithm of choice.
- Although staleness in Secure MVTO is not very high for smaller transaction sizes, as the average transaction size and arrival rate increase, staleness reaches unacceptable values.

4. Conclusions

In this paper, a secure version of the two phase locking protocol was provided. We also presented our simulation model and evaluation results of the relative performance of Secure 2PL, Secure OCC and Secure MVTO.

Throughput this paper, we have discussed secure concurrency control algorithms with no other constraints, other than to maintain the consistency of the database. As a result, fairness and response time were our main performance criterion. However, real-time databases in which transactions have timing constraints associated with them are finding increased applicability. The performance goals of a real-time database system are different from those of conventional database systems. While conventional database systems aim to minimize average response times, real-time database systems seek to minimize number of transactions missing their deadline. In [Son 93], the security impact on real-time databases is discussed. In such an environment, the performance results discussed in this paper are no longer applicable, since good average case performance does not necessarily imply acceptable worst case performance as is required by a real-time system. Our future work involves the problem of integrating the security and the real-time requirements of a database system. We started to investigate how to define the capacity of covert channels and how to use it to control the trade-offs between timing and multilevel security requirements [Son 94].

References

[Agra 87] R. Agrawal, M. J. Carey, and M. Livny. "Concurrency Control Performance Modeling: Alternatives and Implications", ACM Transactions on Database Systems, Vol. 12, No. 4, pp 609-654, December 1987.

[Bell 76] D. E. Bell and L. J. LaPadula. "Secure Computer Systems: Unified Exposition and Multics Interpretation", The Mitre Corp., March 1976.

[Bern 87] P. A. Bernstein, N. Goodman & V. Hadzilacos. "Concurrency Control and Recovery in Database Systems", Reading, MA: Addison Wesley, 1987.

[Gogu 82] J. A. Goguen and J. Meseguer. "Security Policy and Security Models", Proceedings of the IEEE Symposium on Security and Privacy, pp 11-20, 1982.

[Jajo 92] Sushil Jajodia and Vijayalaksmi Atluri. "Alternative Correctness Criteria for Concurrent Execution of Transactions in Multilevel Secure Databases", Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, May 1992.

[Keef 90a] T. F. Keefe, W. T. Tsai and J. Srivastava. "Multilevel Secure Database Concurrency Control", Proceedings of the Sixth International Conference on Data Engineering, pp 337-344, Los Angeles, CA, February 1990.

[Keef 90b] T. F. Keefe and W. T. Tsai. "Multiversion Concurrency Control for Multilevel Secure Database Systems", Proceedings of the 11th IEEE Symposium on Security and Privacy, Oakland, California, pp 369-383, April 1990.

[Koga 90] Sushil Jajodia & Boris Kogan. "Concurrency Control in Multilevel Secure Databases Based on a Replicated Architecture", Proceedings of the 11th IEEE Symposium on Security and Privacy, Oakland, California, pp 360-368, April 1990.

[Lamp 73] Butler W. Lampson. "A Note on the Confinement Problem", Communications of the ACM, Vol. 16, No. 10, pp 613-615, October 1973.

[Lee 94] Juhnyoung Lee and Sang H. Son. "Performance of Concurrency Control Algorithms for Real-Time Database Systems" in "Performance of Concurrency Control Mechanisms in Centralized Database Systems", V. J. Kumar (editor), Prentice Hall 1994.

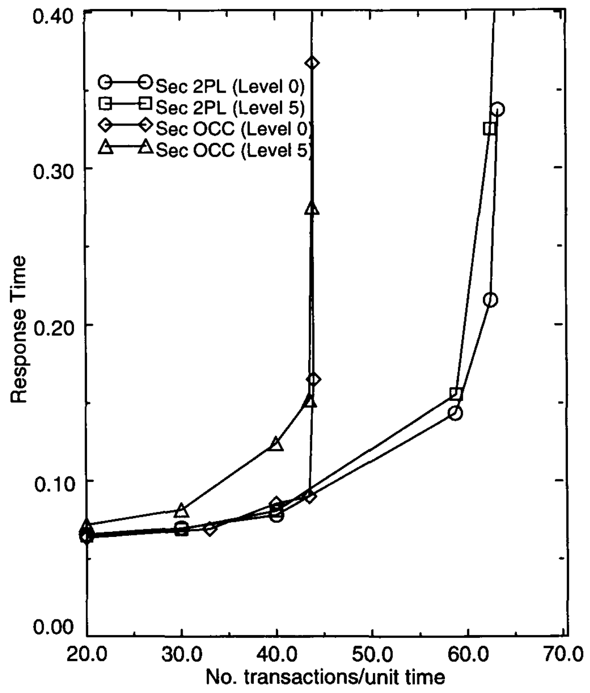
[Son 92] Sang H. Son, Juhnyoung Lee and Yi Lin. "Hybrid Protocols Using Dynamic Adjustment of Serialization Order for Real-Time Concurrency Control", Real-Time Systems Journal, Vol. 4, No. 3, pp 269-276, September 1992.

[Son 93] Sang H. Son and Bhavani Thuraisingham. "Towards a Multilevel Secure Database Management System for Real-Time Applications", IEEE Workshop on Real-Time Applications, New York, New York, May 1993.

[Son 94] Sang H. Son and Rasikan David. "Synthesis of Multilevel Security and Timing Requirements in Complex Real-Time Database System", Complex Systems Engineering Synthesis and Assessment Technology Workshop (CSESAW '94), Washington, D.C., July 1994.

[Thur 93] Bhavani Thuraisingham and Hai-Ping Ko. "Concurrency Control in Trusted Database Management Systems: A Survey", SIGMOD RECORD, Vol. 22, No. 4, December 1993.

Fig 1: Secure 2PL vs Secure OCC


Sec 2PL (Level 0)
Sec 2PL (Level 5)
Sec OCC (Level 0)
Sec OCC (Level 5)
Response Time
No. transactions/unit time