SECURE CONCURRENCY CONTROL ALGORITHM FOR MULTILEVEL SECURE DISTRIBUTED DATABASE SYSTEMS

Navdeep Kaur , Rajwinder Singh
Department of Electronics and Computer Engg., Indian Institute of TechnologyRoorkree, Roorkee, India

Hardeep Kaur Sidhu
Department of Electronics and Communication Engg.,Baba Ishar Singh Polytechnic,Kot-Ise khan,Moga,,India

Keywords: Distributed database, Multilevel secure database system, Concurrency control

Abstract: Majority of the research in multilevel secure database management systems (MLS/DBMS) focuses primarily on centralized database systems. However, with the demand for higher performance and higher availability, database systems have moved from centralized to distributed architectures, and the research in multilevel secure distributed database management systems (MLS/DDBMS) is gaining more and more prominence. Concurrency control is an integral part of database systems. Secure concurrency control algorithms proposed in literature achieve correctness and security at the cost of declined performance of high security level transactions. These algorithms infringe the fairness in processing transactions at different security levels. Though the performance of different concurrency control algorithms have been explored extensively for centralized multilevel secure database management systems but to the best of author's knowledge the relative performance of transactions at different security levels using secure concurrency control algorithm for MLS/DDBMS has not been reported yet. To fill this gap, this paper presents a detailed simulation model of a distributed database system and investigates the performance price paid for maintaining security with concurrency control in a distributed database system. The paper investigates the relative performance of transactions at different security levels.

1 INTRODUCTION

In applications such as military, data and transactions (users) are classified into different levels of security. For these applications security can be implemented by using a database system that can control the access to data based on the security level of users submitting the transactions and the security level of data. This is in-contrast to the traditional database systems where all data in the database and all users who access it belong to the same security level. Database system that can store and manage data with different classifications in a single system is called a multilevel secure database (MLS/DB) system.

In a Multilevel secure database system (centralized or distributed) a security level is assigned to each transaction and data. A security level for a transaction represents its clearance level and the security level for a data represents its classification level. A multilevel secure database management system (MLS/DBMS) restricts database operations based on the security levels.

Concurrency control is an integral part of the database systems. It is used to manage the concurrent execution of operations by different transactions on the same data item such that consistency is maintained. One of the most important issues for concurrency control in MLS database system is the covert channel problem (Lampson, 1973). It naturally comes due to the contention for the shared data items by transactions executing at different security levels. The most common instances of totally ordered security levels are the Top-Secret(TS), Secret(S), Confidential(C), and Unclassified(U) security levels encountered in the military and government sectors. In this paper, we use two security levels: high and low. A primary concern in multilevel security is information leakage, by a high security level transaction-to-transaction executing at a low security level. Covert channels are paths not normally meant for information flow. In multilevel secure databases, a low security level transaction can be delayed or aborted by a high security level transaction due to shared data access. Thus, by delaying low security level transactions in a predetermined manner, high security level information can be indirectly transferred to the lower security level. This is called a covert channel. Direct leakage can be prevented by mandatory access control policies such as the Bell-LaPadula (BL) model (Bell & LaPadula, 1976) but handling of covert channel needs modifications in conventional concurrency control schemes such as two-phase locking (2PL) and timestamp ordering (TO).

Most of the research efforts in the area of secure concurrency control are focused on centralized databases. Several approaches have been proposed for centralized secure concurrency control in MLS/DBMSs. Most of these are either extension of the 2PL protocol or of timestamp-based protocols (Atluri, Jajodia & Bertino, 1997). The performance of secure concurrency control algorithms has also been studied (Son & David, 1994 and Sohn & Moon, 2000.). However, to the best of author's knowledge the performance study of MLS/DDBS has not been yet reported.

The problem of covert channel makes secure concurrency control algorithms more complex than conventional concurrency control algorithms. In this paper, we concern ourselves with concurrency control algorithm that has to satisfy both security and consistency requirements and compare the performance of secure 2PL with non-secure 2PL for secure distributed database via simulation.

The remainder of the paper is organized as follows. The next section presents MLS distributed database model. Section 3 presents the secure two-phase locking concurrency control algorithm that implemented in our simulation model. Section 4 gives the details of the simulation model. The results of simulation experiments are discussed in Section 5. Section 6 concludes the paper.

2 MLS DISTRIBUTED DATABASE MODEL

We use the MLS distributed database model given in (Ray, Mancini, Jajodia & Bertino, 2000). It consists of a set N of sites, where each site N є N is an MLS database. Each site has an independent processor connected via secure (trusted) communication links to other sites. Thus no communication between two sites is subject to eavesdropping, masquerading, reply or integrity violations.

The MLS distributed database is modeled as a quadruple $< D, T, S, L >$, where D is the set of data items, T is the set of distributed transactions, S is the partially ordered set of security levels with an ordering relation ≤, and L is a mapping from $D \cup T$ to S. Security level S_i is said to dominate security level S_j if $S_j \leq S_i$. For every x є D, $L(x)$ є S, and for every T є T, L(T) є S. Every data object x, as well as every distributed transaction T, has a security level associated with it.

Each MLS database N is also mapped to an ordered pair of security classes L_{min}(N) and L_{max}(N). Where L_{min}(N), L_{max}(N) є S, and $L_{min}(N) \leq L_{max}(N)$. In otherwords, every MLS database in the distributed database has a range of security levels associated with it. For every data item x stored in an MLS database N, $L_{min}(N) \leq L(x) \leq L_{max}(N)$ Similarly, for every transaction T executed at N, $L_{min}(N) \leq L(T) \leq L_{max}(N)$. A site N_i is allowed to communicate with another site N_j only if $L_{max}(N)_i = L_{max}(N)_j$. The security policy used is based on the Bell-LaPadula model and enforces the following restrictions:

Simple Security Property: A transaction T(subject) is allowed to read a data item(object) x only if $L(x) \leq L(T)$.

Restricted *- Property: A transaction T is allowed to write a data item x only if $L(x) = L(T)$.

Thus, a transaction can read objects at its level or below, but it can write objects only at its level. In addition to these two requirements, a secure system must guard against illegal information flows through covert channels.

3 SECURE TWO PHASE LOCKING PROTOCOL

Two-phase locking is the most widely used concurrency control algorithm in database systems for synchronizing accesses to shared data and has been realized in most of the commercial systems (Bernstein, Hadzilacos, & Goodman, 1987 and Mohan, Lindsay, & Obermarck, 1986). As the name indicates, two-phase locking (2PL) consists of two phases. The first phase is called expanding phase during which new locks can be acquired but none can be released. The second phase is called shrinking phase during which locks held by a transaction are released but no new locks can be acquired. For strict execution, strict two-phase locking additionally requires that all locks held by a transaction be released only after the transaction commits or aborts (Ceri & Pelagatti, 1984). If a transaction T_i is holding a lock on shared data item x, no other transaction T_j can get access to x if their operation on x conflict. As a result, the isolation of transactions is enforced.

In 2PL, the current lock holder is never aborted due to a conflicting request from another transaction. The new request is blocked until the current holders release their locks. Therefore, two-phase locking is not suitable for MLS databases because a low security level transaction can be delayed by a high security level transaction due to shared data access. Thus, by delaying low security level transactions, high security level transaction can indirectly transfer the information to lower security level transaction by establishing a timing covert channel. If low security level transactions are somehow allowed to continue with there execution in spite of the conflict with high security level transactions, covert channel can be prevented.

Let T_l denotes the low security level transaction and T_h denotes the high security level transaction, i.e., $L(T_l) < L(T_h)$. x and y are data items with low and high security level respectively. Read down is the only conflicting operation between T_l and T_h that can create covert channel

The security model allows a transaction (or sub-transaction) to issue read-equal, read-down and write-equal operations. This is sufficient to prove that security is not violated through data access (Sandhu, 1990).

Table 1 shows all permitted operations in MLS database system.

Table 1: Permitted Operations in MLS Databases

Data Items / Transactions	$L(x_l)$	$L(y_h)$
$L(T_h)$	r[x]	r[y], w[y]
$L(T_l)$	w[x] ,r[x]	-

Based upon these permitted operation following conflicts may occur:

1. (Read-down conflict among different levels): Read-down conflict occurs between $L(T_h)$'s read operation, r[x], and $L(T_l)$'s write operation, w[x].
2. (Read-write conflict at same level): Read-write conflict occurs between $L(T_l)_i$'s read operation, r[x], and $L(T_l)_j$'s write operation, w[x]. Where $L(T_l)_i$, $L(T_l)_j$ and x are at the same security level.
3. (Write-write conflict at same level): Write-write conflict occurs between $L(T_l)_i$'s write operation, w[x], and $L(T_l)_j$'s write operation, w[x]. Where $L(T_l)_i$, $L(T_l)_j$ and x are at the same security level.

To close all covert channels, read-down conflict is the only case that needs to be treated differently from the conventional conflict in MLS database systems.

In this paper we extend the two phase locking high priority (2PL-HP) algorithm given in (Abbott & Molina., 1992) for distributed databases and study the performance of secure distributed 2PL. The rules according to which the algorithm manages its locks and operations are as follows:

- Every transaction in S2PL must obtain a read lock before reading a data item and a write lock before writing a data item. A transaction cannot request additional locks once it has issued an unlock action.
- A transaction holds on to all its locks (read or write) until it completes.

A high security level transaction must release its read lock on a low data item when a low security level transaction requests a write lock on the same data item and the aborted high security level transaction is restarted after some delay.

4 SIMULATION MODEL

To evaluate the performance of the concurrency control algorithms, we developed a detailed simulation model of MLS distributed database model. The model consists of MLS database that is distributed, in a non-replicated manner, over N sites connected by a network. Each site in the model has six components: a source which generates transactions workload of the system; database which models the data and its organization; a transaction manager which models the execution behavior of the transaction; a concurrency control manager which implements the concurrency control algorithm; a resource manager which models the physical resources (CPU and I/O); and a sink which collects statistics on the completed transactions of the site. In addition to these per site components, the model also has a network manager which models behavior of the communications network. Figure 1 shows detailed view of these components and their key interaction.

Source: The source is responsible for generating the workload for each data site. Transactions are generated as a Poisson stream with mean equal to *ArrivalRate*. Each transaction in the system is

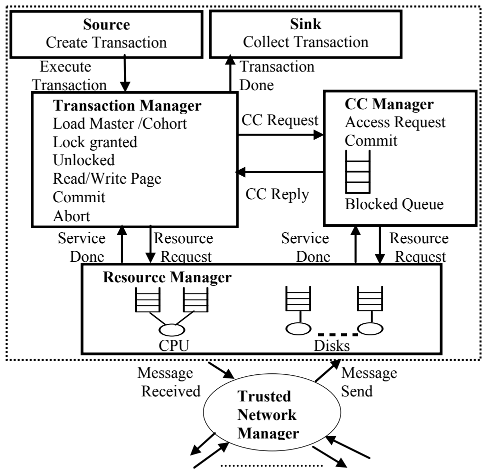


Figure 1: Simulation model of the DDBMS

distinguished by a globally unique transaction id. The id of a transaction is made up of two parts: a transaction number, which is unique at the originating site of the transaction and the id of the originating site, which is unique in the system. Each transaction has an associated security clearance level. A transaction is equally likely to belong to any of the *ClearLevel* security clearance levels. We assume that the clearance level remains constant throughout the life of transaction inside the system.

Database Model: The database is modeled as a collection of *DBSize* pages. These pages have been assigned ClassLevels and are uniformly distributed in a non-replicated fashion across all the *NumSites* sites. The database is equally partitioned into *ClassLevels* security classification levels. Table 2 summarizes the parameters of simulation model.

Transaction Manager: For each distributed transaction; there is one process, called the master or coordinator that executes at the originating site of the transaction and a set of other processes, called cohort that execute at the various sites where the required data pages reside. The number of pages to be accessed by the transaction is determined by the parameter *TransSize*.If there exists any local data in the access list of the transaction, one cohort will be executed locally. When a cohort completes its data access and processing requirements, it waits for the master process to initiate two-phase commit. The master process commits a transaction only if all cohorts of the transaction are ready to commit (all cohorts are voted yes); otherwise it aborts and restarts the transaction after a delay and makes the same data accesses as before.

Resource Manager: The resource manager manages the physical resources of each site. The physical resources at each site consist of *NumCPUs* processors and *NumDisks* disks. There is a single common queue for the processors whereas each of the disks has its own queue. All queues are processed in an FCFS order except that the message processing is given higher priority than data processing at the CPUs. The *PageCPU* and *PageDisk* parameters capture the CPU and disk processing times per data page, respectively.

Concurrency Control Manager: It is responsible for handling concurrency control requests made by the transaction manager, including read and write access requests, requests to get permission to commit a transaction, and several types of master and cohort management requests to initialize and terminate master and cohort processes. Concurrency Control Manager uses strict two-phase locking (2PL) protocol. We have implemented non-secure 2PL and secure 2PL concurrency control managers.

Table 2: Simulation model parameters and values

Parameter	Meaning	Value
NumSites	Number of sites in the database	8
DBSize	Number of pages in the database	4000
ClassLevels	Number of Classification Levels	2
ArrivalRate	Transaction arrival rate / site	Varies
ClearLevel	Number of Clearance Levels	2
TransSize	Average transaction size	4
WriteProb	Page write probability	0.2
NumCPUs	Number of processors per site	2
NumDisks	Number of disks per site	4
PageCPU	CPU page processing time	5ms
PageDisk	Disk page access time	20ms
MsgCPU	Message send / receive time	5ms

Network Manager: We assumed a reliable system, in which no site failures or communication network failures occur. The communication network is simply modeled as a switch that routes messages without any delay since we assume a local area network that has high bandwidth. However, the CPU overheads of message transfer, message transfer are taken into account at both the sending and the receiving sites. This means that there are two classes of CPU requests - local data processing requests and message processing requests. We do not make any distinction, however, between these different types of requests and only ensure that all requests are served in priority order. The CPU overheads for message transfers are captured by the *MsgCPU* parameter.

Sink: The sink module receives both completed and aborted transactions from transaction manager. It collects statistics on these transactions of the site.

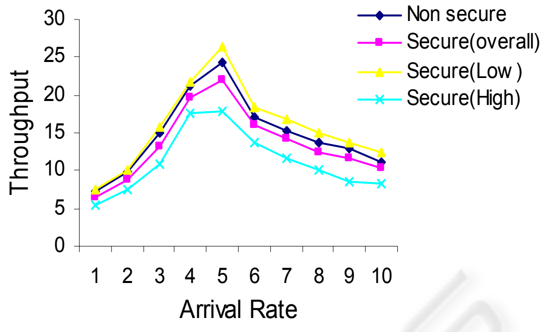
5 EXPERIMENTS AND RESULTS

In this section, we present the performance results of our simulation experiments. The aim of the experiments was to obtain a measure of the performance price that needs to be paid to provide security in a distributed database system. This price was measured as a comparison between the throughput of transactions of non-secure 2PL and that of secure 2PL at two security levels, (i.e., high and low). The throughput is the number of transactions committed per second.

5.1 Experiment 1: Performance under Resource and Data Contention

The workload and system parameter settings taken for this experiment ensure that there are both data contention (DC) and resource contention (RC) in the system. The parameter settings used for this experiment are shown in Table 2. There are two security levels (classification levels), high and low. Correspondingly, there are two transaction security levels (clearance levels).

Graph 1 shows the transaction throughput as a function of the transaction arrival rate per site. It can be seen that the throughput of both concurrency control algorithms initially increases with the increase in arrival rate then decreases when arrival rate becomes more than 5. However the overall throughput of secure 2PL is always less than non-secure 2PL.We also observes that the throughput of high security level transactions is lower than that of low security level transactions as arrival rate increases. This is because higher priority is given to low security level transaction. The high security level transaction is aborted and restarted after some delay whenever a data conflicts occur between a high security level transaction and low security level transaction.

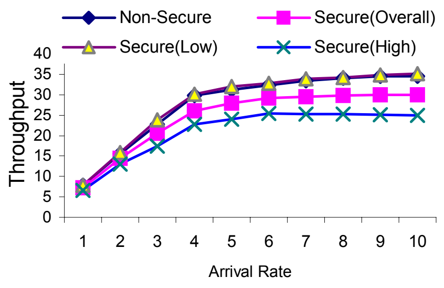


Graph 1: Throughput Vs Arrival Rate (RC+DC)

5.2 Experiment 2: Performance under Pure Data Contention

The goal of our next experiment was to isolate the impact of data contention (DC) on the performance of the concurrency control algorithms. For this experiment, the physical resources (CPUs and disks) were made “infinite”, i.e., there is no queuing for these resources (Agrawal, Carey & Livny, 1987). The other parameter values are the same as used in Experiment 1.

Graph 2 shows the transaction throughput as a function of the transactions arrival rate per site. With infinite physical resources, the throughput should be a non-decreasing function of arrival rate in the absence of data contention. However, for a given database size, the probability of data conflicts increases as arrival rate increases. The initial increase is due to the fact that there is no resource contention. In this experiment the throughput is limited only by data contention and transaction response times are typically smaller than that of under RC+DC. We again observed that the throughput of a secure algorithm is less than that of non-secure algorithms for all arrival rates. In addition, the performance of high security level transaction is significantly lower than that of low security level transaction at high arrival rate.



Graph 2: Throughput Vs Arrival Rate (DC)

6 CONCLUSION

In this paper, we made a preliminary study of the performance price paid for ensuring the covert channel free security in a multilevel secure distributed database system. Using a detailed simulation model of a distributed database system, we studied the performance of two- level (High and Low) secure concurrency control algorithm against an equivalent non-secure concurrency control algorithm. Within secure concurrency control algorithm, our experiment show that the performance of high security level transaction is significantly worst than that of the low security level transaction, highlighting the price that has to be paid for ensuring that there are no covert channels. In our future work, we plan to investigate the schemes by which the performance of high security level transactions can be improved without compromising security.

REFERENCES

Bell, D.E., & LaPadula, L.J., 1976. Secure computer systems: unified exposition and multics interpretation. *The MITRE Corp.*

Atluri, V., Jajodia, S., & Bertino, E., 1997. Transaction processing in multilevel secure databases using kernelized architecture: challenges and solutions, *IEEE Transaction on Knowledge and Data Engineering*, vol. 9, no. 5.

Son, S. H., & David, R, 1994. Design and analysis of a secure two-phase locking protocol. In *18th Int'l Computer Software and Applications Conference*, 374-379, IEEE Computer Society Press

Sohn, Y., & Moon, S., 2000. Verified order-based secure concurrency controller in multilevel secure database management system, *IEICE Transaction of Information & System,* vol. E83-D, no.5.

Ray, I., Mancini, L. V , Jajodia S., & Bertino, E., 2000. ASEP: A secure and flexible commit protocol for MLS distributed database systems, *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 6.

Abbott, R. K. & Garcia-Molina, H., 1992. Scheduling real-time transactions: a performance evaluation, *ACM Transactions on Database Systems*, vol. 17, no. 3, 513-560.

Bernstein P., Hadzilacos, V. & Goodman, N., 1987. *Concurrency control and recovery in database Systems*, Addison Wesley Publishing Company.

Mohan, C., Lindsay, B., & Obermarck, R., 1986. Transaction management in the r*distributed database management system, *ACM Transactions on Database Systems*.

Ceri, S., & Pelagatti, G., 1984. *Distributed databases principles and systems*, McGraw-Hill, New York.

Lampson, B.W., 1973. A note on the confinement problem. In *Communications of the ACM,* vol. 16, no. 10, 613-
615, ACM Press.

Sandhu, R., 1990. Mandatory controls for database integrity. In *DATABASE SECURITY III: Status And Prospects*, 143-150.

Agrawal, R., Carey, M. J., & Livny, M., 1987. Concurrency control performance modeling: alternatives and implications. *ACM Transactions on Database Systems.*