Evolving a model of transaction management with concurrency control for multilevel secure distributed real-time database systems

S.Albert Rabara

Department of Computer Science, St.Joseph's College (Autonomous)
Tiruchirappalli – 620 002, India.

Abstract

The concurrency control in distributed database management systems is an important research problem. Several concurrency control algorithms have been proposed for secure distributed real time database systems, and several have been and are being implemented. Most of the concurrency control algorithms are the variations of the following basic techniques: Two-Phase Locking, Timestamp Ordering and Serialization Graph Testing. These algorithms may not be providing accurate performance model. In this paper, it is proposed an algorithm model to enhance the performance of concurrent transactions for multilevel security for distributed database. This model reduces the data access time and wait time for every transaction monitored by the sub-query analyzer. Simulation study reveals that the effective enhancement of performance for concurrent transactions with different levels of security.

.

Keywords : *Multilevel Security, Distributed Database, Concurrency, Transaction , Performance.*

1. Introduction

The importance of databases in modern businesses, public and private organizations, banks, educational institutions and in general day-to-day applications is already huge and still growing. Many critical applications requires databases. These databases contain data of different degree of importance and confidentiality , and are accessed by a wide variety of users. Integrity violations for a database can have serious impact on business processes and disclosure of confidential data. Traditional security provides techniques and strategies to handle such problems with respect to database servers in a non distributed environment. In a global enterprises the database access is required round the clock. The Database Management Systems (DBMS) has coincided with significant developments today in distributed computing and concurrent access technologies and this becomes the dominant data management tools for highly data intensive applications. A distributed database is a collection of multiple logically interrelated databases distributed over a computer network. A distributed DBMS is defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users[1]

In this scenario, Data Base management Systems (DBMS) are designed to meet the requirements of performance, availability, reliability, security and concurrency. The recent rapid proliferation of Web-based applications and information systems have further increased the risk exposure of databases in a distributed environment and hence the data protection is more crucial than ever. The data needs to be protected not only from external threats but also from insider threats[2]. The solution to data security are classified into three major categories : the protection of data against unauthorized disclosure, prevention of unauthorized and improper modification and prevention of denial of services. In addition to the different levels of security , the DBMS is to provide concurrent access of highly available data in the presence of large and diverse user populations. Therefore it is obvious that multilevel security must be provided to the DBMS mainly on distributed environment. 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 with consistency. Several methods have proposed to provide secure concurrency control to achieve correctness and at reduced cost of high security level transactions. One of the most important issues for concurrency control in MLS database system is the cover channel problem[3]. 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), and Unclassified(U) security levels encountered in the military and government sectors. In this paper, it is proposed a model to enhance the performance of concurrency for Multilevel Secure Distributed Database System. This model allows users to access a database concurrently from geographically dispersed locations through use of concurrency control locking algorithm.

The paper is organized as follows: The review of literature on MLS distributed database and related issues on concurrency control algorithms are presented in the Section-2. Section-3 presents the proposed multilevel secure distributed database model. Section-4 presents lock based protocol for the proposed concurrency model. Section-5 concludes paper.

2. Review of Literature

The operations of the database can be performed in the form of transactions. Several units of works that form a single logical unit of work can be called as a transaction. This can be performed under supervision of transaction manager. A valid transaction must be satisfied the Atomicity, consistency, Isolation, Durability properties [3]. The transaction manager can allow two or more transactions to access the same data called concurrent transactions. The un-controlled concurrent transaction leads to inconsistent database and violate the isolation property. Hence, the transaction manager needs to control the interactions between the transactions. To achieve

this task, transaction manager uses several concurrency control mechanisms, such as locks, time stamps, etc.

James M. Slack [3] presented two security mechanisms: the first mechanism is based on one–way protected group. One-way protected group is a set of one-way protected objects. Each one-way protected object in the group will accept messages only from a distinguished object in the group called the interface object. A one-way protected group supports data integrity and access integrity. The second mechanism is two-way protected group. It is an extension of one-way protected group. Each object in the group could only send the messages of that group. This model fails to address how security and integrity policies are implemented with protected groups.

Keishi Tajimia [4] has developed a technique to statically detect the security flaws in OODBSs. He designed a framework to describe the security requirements and developed an algorithm to determine the security flaws. In this technique a user can bypass the encapsulation and abuse the primitive operations inside the functions and also the properties of aggregate are not given.

Linda M. Null et al. [5] has defined security policy for object-oriented data model. This security policy addresses mandatory as well as discretionary access controls. This model is based on the classes derive security classification constraints from their instances and logical instances. The implementation and suitability of these policies in the object-oriented environment is not presented in their proposal.

Sushil Jajodia et al. [6] has proposed a database security model for mandatory access control that details with the Object-Oriented Data model. This includes a message filtering algorithm that protects the illegal flow of information among objects of various security levels. Finally a set of principles is defined to design and implement security policies in Object-Oriented Database Management Systems. This model fails to address information flow are rendered explicitly.

Roshan. K. Thomas et al. [7] gives a kernelized architecture for multilevel secure Object-Oriented Database Management Systems (DBMS's) which support write-up. However, supporting write-up operations in object-oriented systems is complicated by the fact that such operations are no longer primitive, but can be arbitrarily complex and therefore can take arbitrary amounts of processing time. This architecture supports Remote Procedure Call (RPC) based write-up operations. Dealing with the timing of such write-up operations consequently holds less time.

Bertino et al.[8] proposed the practical relevance of nested transactions and for the theoretical issues related to the development of suitable locking mechanisms and serializability theories for nested transaction. In particular, the interactions between parent and child transactions in the same nested transactions can be executed concurrently. This require revise and extend primitives and the locking protocol, however as it mentioned. Serializability theory for nested transactions is substantially more complex.

Lin et al.[9] a simplified simulation model is used to compare the performance of basic timestamp, multi version timestamp, and two-phase locking algorithms. It doses not include different data distributions(partitioned, replicated, etc.), and simplified communication delay by combining CPU processing time, communication delays, and I/O processing time for each transaction.

Navdeep Karur et al.[10] have presented a simulation model of a multilevel secure distributed database system using secure concurrency algorithm. It addresses the performance price paid for maintaining security in a MLS/DDBMS, but performance of higher security level transactions in a replicated database has not been studied.

Having studied the above literature, it has been identified that most of the research efforts in the area of secure concurrency control are focused on centralized databases. Concurrent access of distributed database is mandatory for applications like banking, financial, enterprises, industry and institutes etc., Concurrent Transactions within a distributed database management system face several restrictions. The proposed model is designed to perform efficient, secured concurrent transactions based on sub-query analyzer.

3. Proposed Model

The data stored in a database should be secured from the unauthorized users. The data retrieval time is minimized by the Lock Manager (LM). This manager handles the locking mechanism for distributed sites. All transactions are sorted through the security manager (SM).. The security manager (SM) handles the authentication of the user using with proper verification. The different query levels proposed in the model are View level ($SL(D_u)$) , Secret level ($SL(D_s)$) and Top secret level ($SL(D_t)$). If the user is permitted access to the view level, the query is limited with the view level transaction ($SL(Q)<SL(D_u)$) . Otherwise, the query is roll back to Transaction Level security. Transaction level is also classified into secret and top secret respectively upon the user classifications.

The query optimizer further divides the query into various levels for distributed access and creates a new optimized query according to the data distribution. The Transaction Manager (TM) pools the query in the transaction queue and allows the transaction to be executed according to the load balance of the transaction concurrency. The network traffic is also considered by the Transaction Manager (TM). The Transaction Concurrency Manager (TCM) again analyzes the query and arranges the query for various levels of concurrency. The waiting time stamp is used for each transaction. The transactions are executed without concurrency when the waiting time stamp is expired. So the infinity waiting time of transaction is avoided. The waiting time stamp for each transaction is proportional to the security levels. The lock manager transmits the lock signal to the entire distributed database sites. It allows the transaction to update the data only when there is no objection from any other sites. Otherwise, it rolls back the transaction. If there is no objection from all other lock managers then it locks the data. The transaction is rolled back by the lock manager if the transaction exceeds the timestamp limit.. Thus the proposed model felicitates to access data only after passing through multiple security levels and also allowed to access concurrently without any access conflict. The proposed model is depicted in Figure1.

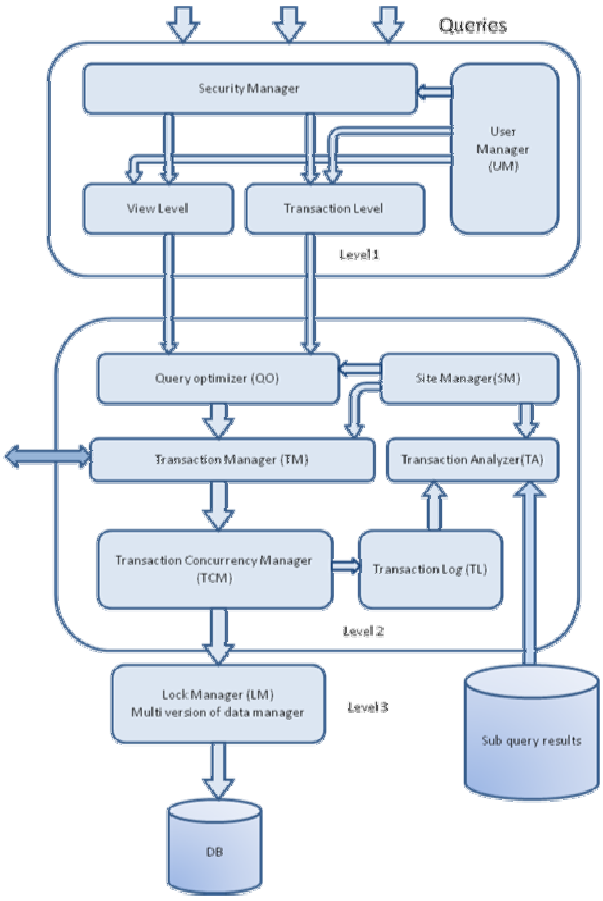

Figure 1. Proposed Model

3.1 Secure Concurrency Control Protocol

The various modules of the secure concurrency control protocol including the concurrency components are illustrated as follows:

The various modules proposed in the MLS framework including concurrency components are briefly presented here.

The User Module (UM) handles the data for the accepted users with their privileges. It is defined as a three tuple. U=(V_s, D, M), where V_s is a verification code set, D is the data which is associated with the user and M is the mode of accessibility. The set V_s may be any one of security mechanism such as Username and password, IP based security and other hybrid security. User Log (UL) is maintained by the User Module (UM).

The data stored in a database should be secured from the unauthorized users. The security manager (SM) handles the authentication of the user using proper verification. All transactions are sorted through the security manager (SM). The different query levels proposed in the model are View level ($SL(D_u)$) , Secret level ($SL(D_s)$) and Top secret level ($SL(D_t)$). If the user is permitted access to the view level, the query is limited with the view level transaction ($SL(Q)<SL(D_u)$) . Otherwise, the query is rolled back by the Security Manager (SM) . Transaction level is also classified into secret and top secret respectively based on the user classifications. In the transaction level one can update the data, but updation of data is not permitted in the view level. Each user having their own level of access limits to the data. Each transaction is permitted with the access limits controlled by the Security Manager (SM).

The Query Optimizer is designed to optimize the quires received from distributed locations. Let the set of sites (S_i) and each site S is having the set of fields (F_j). The fields F is the sub set of sites S. Let T_i be any transaction which involves the set of sites S_i and output or condition fields TF_i. The site S_i is eliminated from the query, if any one of the fields F_j of S_i does not belongs to the set TF_i. This process will eliminate the unnecessary sub transaction ST_i . So the network and data computation cost is reduced. The procedure is presented below.

```
void queryOptimizer(transaction T)
        TF[]=getOutputFields(T)
        TF[]=getComputationalFields(T) // add to output
                                          fields
        S[]=getDistributedDatabase(T)
        F[][]=getFields(S) //two dimensional table

match the each row F[x][] with TF[]
If F[x][] is not match with any TF[]
        eliminate S from T
```

The Site Manager (SMR) looks after all the distributed data which is created or configured for the transaction. This helps to secure the data from the unauthorized network access to the services. The site manager sends the signal to its own network boundary about the various locations of which data distributed and its configuration. This helps to attach and detach of sites over the network. The site manager has the details about the sites (S_i) and the associated fields (F_i). A transaction T_i is rolled back , if the transaction wants to access field TF_i in site S_i , which is not belongs to S_i. The Site Manager (SMR) handles the distributed sites and their locations. It also handles the frequency of updating the records. So it assigns a time stamp for each site. This time stamp is used for the transaction analyzer for flush the Sub Query Results (SQR).

The Transaction Manager (TM) further divides the query into various levels for distributed access and creates a new optimized query according to the data distribution. The Transaction Manager (TM) pools the transactions T_i in the transaction queue and allows the transaction to be executed according to the load balance of the transaction concurrency. The network traffic is also considered by the Transaction Manager (TM). Let T_i be any transaction and ST_{ij} be any sub transaction which is derived from the parent transaction T_i. We can define dispatcher as a four tuple T_i= {ST, TS, DB, CL}. Let ST be the sub transaction, TS be the time stamp, DB be the database and CL be the listing port for the sub transaction in the network. The transaction T_i is rolled back, if its all sub transactions S_i is not finished with in the timestamp order. The transaction manager (TM) saves the sub query result in the disk until it receives the updated message from the database. So before sending the Sub Transaction (ST) it checks its own Sub Query Results (SQR). For that reason we can avoid repeated transaction for the same data. The Transaction Dispatcher (TD) sends the sub query to various sites. After completing that query it assembles the result and

produces the result. The dispatcher using the timestamp mechanism for getting the results back.

The Transaction Analyzer (TA) analyzes the pooled transactions arrived from various distributed locations using the transaction log and creates the Frequently Accessed Fields (FAF) table. It also set the expiry time for each record in Frequently Accessed Fields (FAF) table. Using the timestamp, the FAF table entry is flushed. Let T_i be any transaction, TF_i be a set of fields which is going to be access by the transaction T_i, FAF_i be any entry in FAF table, $FAFT_i$ be the time stamp for record FAF_i, $FAFS_i$ be the current state of the Sub Query Results (SQR). The sub transaction ST_i for any transaction T_i is allowed to access the other site, if there is no entry in FAF table or the current state of $FAFS_i$ is cleared. The Sub Query Results (SQR) is updated, if or the current state of $FAFS_i$ is cleared and the sub transaction ST_i for any transaction T_i is allowed to access the other site. Transaction Analyzer (TA) flushes the Sub Query Results (SQR) when it receives the updated message from any other site. The Sub Query Results (SQR) is only for the transactions which are frequently involved. SQR proposed in this model reduce the unnecessary data access time and wait time for every transaction.

```
Void Transaction Analyzer (Transaction T)
    TF[ ] = get output fields (T)
    TF[ ] = get computational fields (T) // add with output
                                          Fields
   S[ ] = get Distributed Database (T)
   For  i = 0 to n
      If (FAF [i] is found for TF [i] and
         FAF[i] is not cleared)
         data [i] = get Data (SQR)T)
      Else
         data[i] = get Data(S,T)
         updata (SQR, data)
   Assemble all Data[i]
```

The Transaction Concurrency Manager (TCM) analyzes the concurrent transactions T_i and arranges the transaction for various levels of concurrency TC_i. The waiting time stamp is used for each transaction. The transactions are executed without concurrency when the waiting time stamp is expired. Thus the infinite waiting time of transaction is avoided. The waiting time stamp for each transaction is proportional to the security levels.

Let $T_1,T_2,\ldots T_n$ be n transactions which can be executed concurrently. Let U_i be the unit of a transaction for any transaction T_i. The schedule S is created for the transactions $T_1,T_2,\ldots T_n$. The schedule S can be defined as a two tuple. S= (U , T). where U is a unit and T is a transaction. Let Unit Tree (UT) be the executed unit of transaction which are waiting for commit. The node for the unit tree is constructed when each unit transaction U is executed by the concurrency manager. The Transactions (T_i) are associated with the one-to-one mapping to the unit tree UT . The state for the Unit Tree (UT) is marked as roll back tree when the unit transaction of Schedule is failed. The Transaction associated with UT_i will not be executed in the Schedule if the Unit Tree UT_i is set by the Concurrency Manager(CM) . The final commit is only after finishing the entire schedule. The commit is only for the Unit Tree which are not marked as a roll back tree by the Concurrency Manager (CM). The According to this new concurrency algorithm, If the schedule S is failed then there is no need to roll back the entire transaction. The concurrent transactions of the proposed model are shown in Figure 2.

The Lock Manager (LM) transmits the lock signal to the entire distributed database sites. It allows the transaction to update the data only when it passes all the clearance from all security levels. Otherwise, it rolls back the transaction. If there is no objection from all other lock managers then it locks the data. The transaction is rolled back by the lock manager if the transaction exceeds the timestamp limit. The locking mechanism follows a **non cyclic** tree structure. If here any cyclic tree is formed by the transaction T_i then the T_i is rolled back. Hence the deadlock is prevented. Thus the proposed model felicitates to access data only after passing through multiple security levels and also allowed to access concurrently without any access conflict.

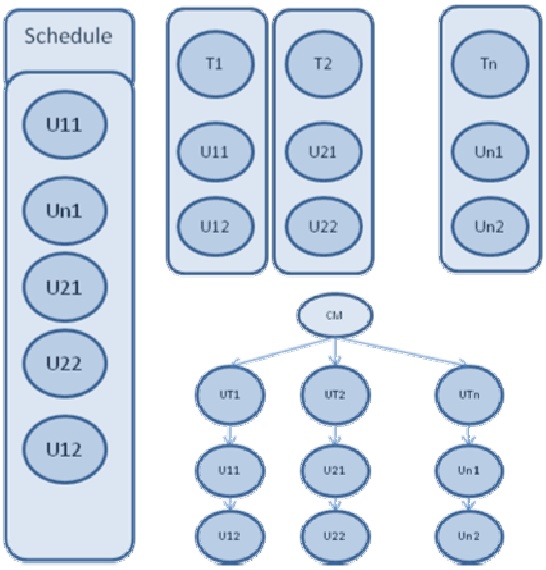

Figure 2. Concurrent Transactions

3.2. Concurrency Control Algorithm

Distributed database management system (DDBMS) allows users to access a database concurrently from geographically dispersed locations interconnected by a network. Concurrent accesses to the database have to be synchronized in order to maintain data consistency and to ensure correctness. This is achieved through use of distributed locking protocol which is applied in this proposed model. The efficient method of implementing concurrency is depicted in Figure 3 and presented in this section.

Let T_u denotes the unclassified security level transaction. Let T_s denotes the secret security level transaction and T_{ts} denotes

the top secret security level transaction, i.e. $SL(T_u) \leq SL(T_s) \leq SL(T_{ts})$.

Let r(x) and w(x) be the read and write of data item x respectively.

Based upon the above assumption, the following conflicts may occur:

(i) (Read-down conflict among different levels): Read-down conflict occurs between $SL(T_s) \leq SL(T_{ts})$*'s* read operation, *r[x]*, and $SL(T_{ts})$*'s* write operation, *w[x]*.

(ii) (Read-write conflict at same level): Read-write conflict occurs between $SL(T_s)_i$*'s* read operation, *r[x]*, and $SL(T_s)_j$*'s* write operation, *w[x]*. Where $SL(T_s)i$, $SL(T_s)_j$ and *x* are at the same security level.

(iii) (Write-write conflict at same level): Write-write conflict occurs between $SL(T_{ts})_i$*'s* write operation, *w[x]*, and $SL(T_{ts})_j$*'s* write operation, *w[x]*. Where $SL(T_{ts})i$, $SL(T_{ts})_j$ and *x* are at the same security level.

Every transaction in this security model must obtain a read lock before reading a data item and a write lock before writing a data item. The security model allows a transaction to issue read-equal, read-down and write-equal operations. This is sufficient to prove that security is not violated through data access. The execution of a distributed transaction T is divided into sub-transactions T_i, where i=1 to n. A sub-transaction T_i is sent to the node N_i where the data is available and executed under the local security and concurrency transaction manager. If a sub-transaction fails, then the parent transaction is rolled-back and restarts after some delay to avoid repeated restart. The TLM (Transaction Lock Manager) determines at which node data items requested by a transaction are located. If the data is available in the parent node N_i, it is accessed in the same node N_i otherwise, if there is no local copy and multiple copies exist at more than one node, then one copy is randomly selected and locks other copies of the same data. It creates one sub-transaction for each node N_i that needs to be visited and acts as the coordinator in the distributed two-phase commit process. Even though the dispatches of sub-transactions of a transaction appear sequential, they are dispatches concurrently. Parent transactions originate from a fixed number of terminals and their number in the system is the sum of terminals connected to each node. This methodology is diagrammatically depicted in Figure 3.

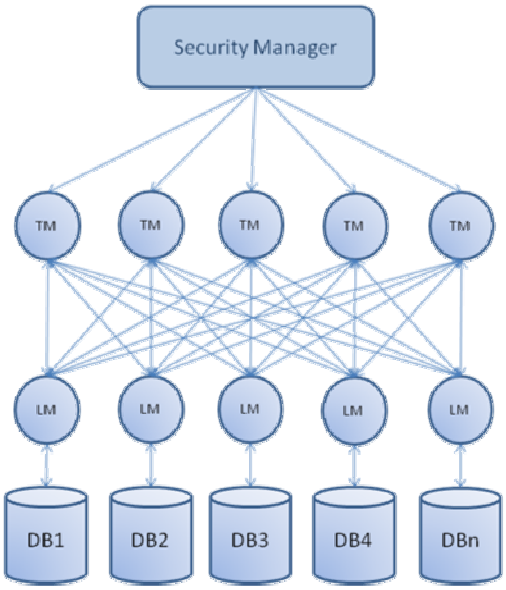

Figure 3. Secured Concurrency Control

The CTM (Concurrency Transaction Manager) coordinates concurrency control activities with other nodes. In the case of data replication, it implements a read-one-write-all policy for read requests. For a write request, it consults all nodes that hold a copy of the desired data item. A DM (Data Manager) at every node contains information about data distribution and replication.

Let V1 be the set of transactions V1={T1,T2, …, Tn} and V2 be the set of Distributaded Databases V2={DB1,DB2,…,DBn}. The set of all links E={e1,e2,…e_m} connecting from T_i where $T_i \in$V1 to DBj where $DB_j \in$V2 denotes the transaction T_i can access the data on the distributed database DB_j as illustrated in the Fugure 3.

A bipartite graph is a triple G = (V1, V2, E) where V1 and V2 are two disjoint sets of vertices, respectively the top and bottom vertices, and E $\subseteq$ V1 X V2 is the set of edges.

The difference with classical graphs lies in the fact that edges exist only between top vertices and bottom vertices.

Two degree distributions can naturally be associated to such a graph, namely the **top degree distribution** ($V1_k$)

$$V1_k = \frac{|\{t \in V1: d(t) = k\}|}{|V1|}$$

Where d(t) denotes degree of a vertex t

and the **bottom degree distribution** ($V2_k$)**,**

$$V2_k = \frac{|\{t \in V2: d(t) = k\}|}{|V2|}$$

A transaction cannot request additional locks once it has issued an unlock action. It holds on to all its locks(read or write) until it completes. A top secret security level transaction must release its read lock on a low data item when a unclassified security level transaction requests a write lock

on the same data item and the aborted top secret security level transaction is restarted after some delay. Thus multiple transactions are performed simultaneously with minimum cost.

Table-1 shows the access permissions in the proposed model

Data Items / Transactions	$SL(x_{ul})$	$SL(y_{sl})$	$SL(z_{ts})$
$SL(T_{ts})$	r[x]	r[y]	r[z],w[z]
$SL(T_{sl})$	r[x]	r[y],w[y]	-
$SL(T_{ul})$	r[x]	-	-

Table 1

Concurrency Control Algorithm

void concurrency(transaction T)
U[]=divide the transaction into various unit of transaction
Analyze the transactions, which are in the queue
Create the Schedule for the various units of transaction
While(schedule finish)
U= next unit to be executed
UT=the Unit Tree associated with U
If (UT is not marked as roll back tree)
Execute the unit (U) in the schedule
If(U is success)
Place U in the Unit Tree (UT)
else
mark the Unit Tree (UT) as roll back tree
while(all unit tree UT in CM is traversed)
if(UT is not marked as roll back tree)
commit the Unit Tree UT
else
reject the transaction Ti which is associated with Unit Tree UT

4. Simulation Results

The protocols for evaluating the performance of concurrency control is tabulated. This evaluation is based on the performance presented in [11]. The aim of this experiment is to test the transaction performance with the proposed concurrency algorithm. The model is simulated in a real time environment presented in Table 2.

Parameter	Value
NumDBS	5
No. Query / sec	31
No. CPU	5
Disk for each site	5
Log disk	1
Concurrent Transaction / sec	11
Write ratio	6
Read ratio	25
Waiting time out transaction	4
Execution time	0.0715 s
Schedule Execution time	0.2015 s
Transaction size	10 records
CPU time / unit	0.0119 s
Network delay	0.01 s
Time for optimize	0.002
Time to partition	0.014
Unit / transaction	6

Table 2. Simulation Parameters

The performance study is carried out with varying time factor. The number of transactions is directly proportional to the increased level of security with varying time factor. Let NT be the number of transactions and TT be the time to finish the transactions then,

$$NT \propto 3.525\ TT$$

The time for executing the transaction is directly proportional to the security level also. Let SL be the security level and TT be the time to finish the transaction then,

$$TT \propto 1.1928\ SL$$

The simulation results graphically represented in Figure 4,5 and 6 shows that the performance of concurrent transactions increased with the transactions arrival rate is increased. The performance of the transaction is high when the number of distributed database is increased.

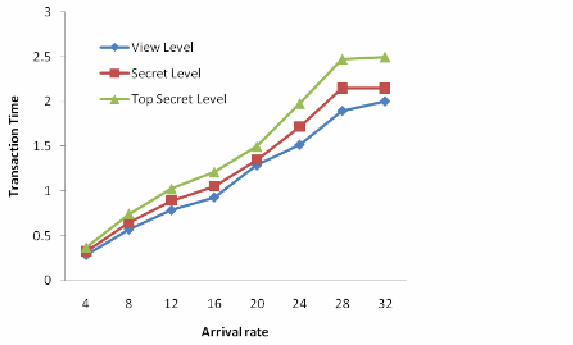

Figure 4. Transaction time vs Arival rate

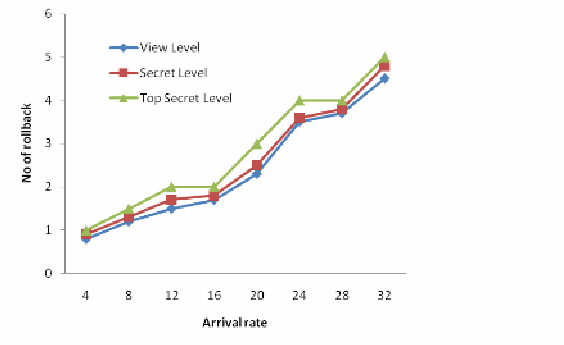

Figure 5. No of rollback vs Arival rate

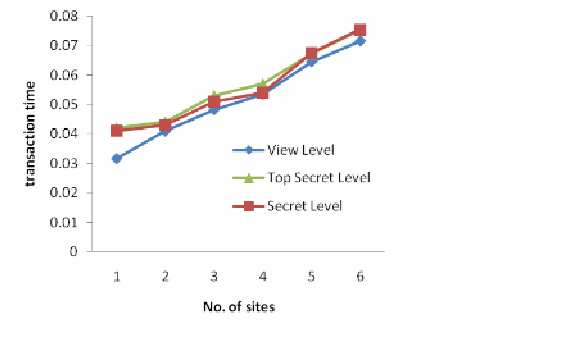

Figure 6. Transaction time vs No. of sites

5. Conclusion

Distributed database systems today play a reality. Several organizations are now deploy distributed database systems. Security is a serious concern a while accessing data. Different security levels need to be integrated into the database to avoid access conflict. We have proposed a new model for Multilevel Secure Real-time Database Systems(MLSDD) and deployed concurrency control for the secured access of data. The simulation study reveals that the performance of concurrent transactions is enhanced with multiple levels of security over the distributed database. This model can be applied for any real-time environment such as, corporate, financial enterprises, academic institute etc., performing day-to-day transactions in a distributed environment cutting edge to different levels of security.

6. References

[1] Elisa Bertino, Ravi Sandhu, "Database Security – Concepts, Approches, and Challenges" IEEE Transactions on Dependable and Secure Computing, Vol 2, No 1, Jan-Mar 2005.

[2] M.Tamper Ozsu, Patrick Valduriez, "Distributed and Parallel Database Systems", ACM Computing Surveys , Vol – 28, No. 1, March 1996.

[3] James M. Slack, "Security in an Object-Oriented Databases," *Proc. Of ACM Transactions*, pp 155-159, 1993.

[4] Keishi Tajimia RIFMS, Kyoto University Japan, "Static Detection of Security Flaws in OODBMS," *ACM SIGMOD international conference on Management of data SIGMOD '96,* Volume 25 Issue 2, pp 341 – 352, June 1996.

[5] Linda M.Null, "The DIAMOND Security Policy for Object-Oriented Databases", ACM annual Conferece on Communicaions, page 49-56, 1992

[6] Sushil Jajodia, Boris, Kogan, and Ravi S.Sandhu, "A Multilevel Secure Object-Oriented Data Model," *Proceedings of ACM SIGMOD,* pp 596-616.

[7] Roshan K. Thomas and Ravi S. Sandhu, "A Kernelized Architecture for Multilevel Secure Object-Oriented Databases Supporting Write-Up", *Journal of Computer Security,* 1993.

[8] E.Bertino, B. Catania And E.Ferrari, "A Nested Transaction Model for Multilevel Secure Database Management Systems", ACM Transactions on Information and System Security, Vol. 4, No. 4, November 2001, pp. 321–370.

[9] W. Lin and J.Nolte, Basic Timestamp, Multiple version Timestamp, and Two-phase Locking, In Proceedings of 9th International Conference on VLDB Conference, Florence, Italy, 1983.

[10] Navdeep Kaur, A.K.Sarje and Manoj Misra, Performance Evaluation of Concurrency Control Algorithm for Multillevel Secure Distributed Databases, IEEE Compter Society, 2004.

[11] Ming Xeoing et al.., "Mirror : A State Consciious Concurrency Control Protocol for Replicated Real Time Database", Journal of Information Systems, No 27, 2002, pp. 277-297.