Journal of Computer Science 3 (7): 561-565, 2007
ISSN 1549-3636

A Secure Time-Stamp Based Concurrency Control Protocol For Distributed Databases

[1]Shashi Bhushan, [2]R. B. Patel and [3]Mayank Dave
[1]Department of Computer Engineering, HEC Jagadhari, HR, India
[2]Department of Computer Engineering, M. M. E. C. Mullana, Ambala, HR, India
[3]Department of Computer Engineering, NIT Kurukshetra, HR, India

Abstract: In distributed database systems the global database is partitioned into a collection of local databases stored at different sites. In this era of growing technology and fast communication media, security has an important role to play. In this paper we presented a secure concurrency control protocol (SCCP) based on the timestamp ordering, which provides concurrency control and maintains security. We also implemented SCCP and a comparison of SCCP is presented in three cases (High, Medium and Low security levels). In this experiment, It is observed that throughput of the system decreases as the security level of the transaction increases, i.e., there is tradeoff between the security level and the throughput of the system.

Key Words: Transaction, timestamp, Protocol, and Concurrency

INTRODUCTION

A distributed database system is described as "one in which multiple database sites are linked by a communications system in such a way that the data at any site is available to users at other sites" [1]. This is a system which has several characteristics such as: (1) provides an interface to user which is transparent to where the data actually resides; (2) ability to locate the data; (3) a DBMS to process queries; (4) network-wide concurrency control and recovery procedures; and, (5) mediators to provide translation of queries and data between heterogeneous systems.

In a secure distributed database system 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 secure distributed database management system restricts database operations based on the security level and provides security classes. 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. 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.

Communications in a distributed system is a complicated and rapidly changing field [1]. There are three basic components in any data communications system: the source, the medium and the sink. The message originates at the source, the path that the message travels is the medium, and the mechanism that presents the data to the user is the sink. There are many different links, channels, or circuits over which the data can travel resulting in a complex communication medium. In addition, there are many characteristics that must be considered when transferring the data: path establishment time, network delay, transfer rate, and reliability [1].

In this paper, we use three security levels: high, medium and low. 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 problem [3]. To handle of covert channel needs modifications in conventional 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.

System Model: In a secure distributed database system, the global database is partitioned into a collection of local databases stored at different sites. It consists of a

Corresponding Author: R. B. Patel, Department of Computer Engineering, M. M. E. C. Mullana, Ambala, HR, India

set of N number of sites, where each site N_i is having a secure database, which is a partition of global database scattered on all the N sites. Each site has an independent processor connected via secure (trusted) communication links to other sites.

The secure distributed database is defined as a five tuples $< D_t, T_t, T_s, S_c, L_v >$, where D_t is the set of data items, T_r is the set of distributed transactions, T_s is the timestamp provided by coordinator as shown in Fig. 1, each transaction is provided a timestamp in ascending order, S_c is the partially ordered set of security levels with an ordering relation $\leq$, and L_v is a mapping from $D_t \cup T_r$ to S_c. Security level $S_c i$ is said to dominate security level $S_c j$ if $S_c j \leq S_c i$. For every $x \in D_t$, $L_v(x) \in S_c$ and for every $T \in T_r$, $L_v(T) \in S_c$. Every data object x, as well as every distributed transaction T has a security level associated with it. Each secure database N is also mapped to an ordered pair of security classes $L_v \min(N)$ and $L_v \max(N)$, where $L_v \min(N)$, $L_v \max(N) \in S_c$, and $L_v \min(N) \leq L_v \max(N)$. In other words, every secure database in the distributed database has a range of security levels associated with it. For every data item x stored in an secure database N, $L_v \min(N) \leq L_v(x) \leq L_v \max(N)$. Similarly, for every transaction T executed at N, $L_v \min(N) \leq L_v(T) \leq L_v \max(N)$. A site N_i is allowed to communicate with another site N_j only if $L_v \max(N_i) = L_v \max(N_j)$. The security policy used is based on the Bell-LaPadula model[4] and enforces the following restrictions:

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

Restricted Property: A transaction T is allowed to write a data item x only if $L_v(x) = L_v(T)$. Thus, a transaction can read objects at its level or below, but it can write objects only at its level. As in [6] we also disallow transactions that write to higher levels for the sake of database integrity [4, 5]. In addition to these two requirements, a secure system must guard against illegal information flows through covert channels.

A user at any site can issue a global transaction against the global schema. The global schema is accessible to all users by one of the following configurations:

1. Replicate the global schema on all sites.
2. Select only one site (called the coordinator) to maintain the global schema and the global transaction manager, and direct requests against the global schema to that site.
3. Select number of sites (coordinators) to maintain copies of the global schema and the global transaction manager, and direct requests against the global schema to the nearest coordinator.

We favor the second or the third alternative over the first one because it is difficult to maintain a copy of the global schema at every site. It also hinders the expandability and simplicity of the system. The coordinator solves the problem of assigning timestamps in Step 2, which is responsible for assigning timestamps to all global transactions. In this article this case is considered, and in future we will consider a variation of the mechanism, which supports the configuration in Step 3.

System Architecture: In our architecture, coordinator (GT_r Global Transaction Manager) is a software module which translates and decomposes the transaction against the global schema into subtransactions against local schemas, and coordinates the execution of the subtransactions. GT_r is divided into various layers as shown in Fig. 1. The description of various layers is as follows: First layer Transaction Interface, this layer of architecture will receive the global transaction from the outer sites (requesting site), i.e., this is the only interface with which requesting sites send their requests in the form of transaction (Queries). This layer is responsible for the compatibility with the global transactions coming from the other sites. Second layer is Authentication Check Layer: This layer checks the authentication of the requester, i.e., whether the particular requester is authorized for the data items he is requesting for? This layer also checks security level of the requested data item. This may be checked according to user name and password and security level of the data items. Security Level Assignment layer: This layer of the coordinator will allocate the security class to the authorized transaction, received from the previous layer. This assignment is implemented with the help of a table. All the transactions are allotted a security class/level. This is the major task of the coordinator for securing the complete transaction.

Fourth layer is having two parts, transaction manager and data manager. Transaction Manager: This manager takes the security assigned and authorized transaction from previous layer. It is also responsible for resolving the global transaction into sub transaction.

Transaction Manager also allots the timestamp to all the sub transaction and decides the data, a requester need. This layer is also doing site allocation with the help of index, which contains the IP addresses of the sites that holds the partition of the database. The sub transaction further dispatched to the respected sites, where the data item resides. Data Manager: This manager is responsible for handling the complete data, i.e., all the data received and given to the requester. Data partitioning is also handled by this phase according to the information received from the transaction manager.

The Data Access Tracker: In the proposed scheme, timestamps are not maintained with the data items. Instead, the list of timestamps associated with each data item is stored in the Data Access Tracker (DAT) as a part of the GT_r. Each time a data item is added into a component database, a corresponding timestamp list is inserted in the DAT which is initially made empty. A list of timestamps associated with data item x comprises the timestamp of the last global subtransaction that has written that data item, denoted by $WT_s(x)$, and the timestamp of the last global subtransaction that has read it, denoted by $RT_s(x)$. Each global subtransaction, upon its initiation, is assigned a unique timestamp, and timestamps are assigned in ascending order. Using the DAT, the GT_r can detect the direct conflicts between Global transactions, and the GT_r uses this information to submit global subtransactions to each LT_r in a serializable order, as will be explained in the next section.

The LT_r is also divided into various layers as shown in Fig. 1, which are as follows: First layer is sub transaction interface layer. This layer resolves the transaction (which is a sub transaction), and decides the data required by the transaction at local Processing site. Sub Query Manager: this layer resolves the required data i.e., it calculates the actual data needed from the local database. All the relevant information is passed to the next Data Administrator Layer for further accessing of data.

Data Administrator Layer: This layer is fully responsible for the data management. This layer is also responsible for security checks on the data items. This layer also sends DAT update massage. The algorithm presented in this paper is a part of this layer.

Local Database: This is the actual database within which data item resides. This database is a partition of the global database.

Assumptions: The proposed transaction model is based on the following assumptions: There is only one global transaction manager (which works like coordinator) per federation of databases. No modifications are allowed to LT_r. A global transaction can have at most one sub-transaction. No site or communication failure is considered. The processing of a transaction requires the use of CPU and data items located at local site or remote site. No replication of data items at various sites is considered here. Arrivals of transactions at a site are independent of the arrivals at other sites. The model assumes that each global transaction is assigned a unique identifier. Each global transaction is decomposed into subtransactions to be executed by LT_r, and these subtransactions inherit the identifier of the global transaction. The problem of finding a correct decomposition for a given transaction will not be addressed in this paper. Transactions make requests for the data items and concurrency control is implemented at the data item level. A secure (trusted) communication network interconnects the sites. There is no global shared memory in the system and all sites communicate via messages exchange over the secure (trusted) communication network.

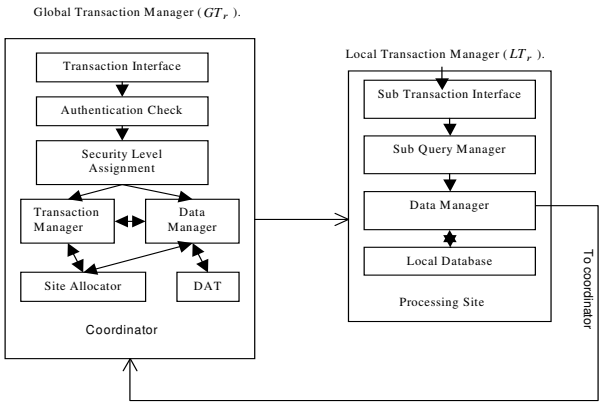

Fig. 1: System Architecture

Implementation

Serializing Global Subtransactions: For the concurrency control of global transactions, the GT_r uses the information available in the DAT to produce a correct serialized order for global transactions. This order is enforced at local sites by the interface process at each site. For example, let GS be the set of global subtransactions to be executed at a local site. Global database consistency is guaranteed if there exists a total ordering of subtransactions from GS such that, if a subtransaction S_i precedes a subtransaction Sj in this ordering, then for every pair of atomic operations O_i and O_j, from S_i and Sj, respectively, O_i proceeds O_j in each local schedule [7]. Therefore, if the GT_r submits global subtransactions to the involved LT_r in a serializable order, we can guarantee the concurrency control of the global transactions. Serializing the global

subtransactions can be done by applying the timestamp ordering protocol using the information in the DAT.

Modifying the Local Transaction Managers: Global serializability can be guaranteed if Local Transactions LT_r provide the local serialized orders to the Global Transactions (GT_r). Since some LT_r do not provide the serialized order, Sugihara [2] suggested the creation of a local controller at each site. The local controller maintains the serializability graph of that site and is responsible for detecting a cycle. A global schedule is serializable if the global graph does not have a cycle. Concurrency control based on distributed cycle detection solves the global concurrency control problem and achieves higher degree of concurrency at the expense of violating the local autonomy.

Simulation Model: To evaluate the performance of developed concurrency control algorithm, we have developed simulation model for the distributed database. Architecture of the simulation model is shown in Fig. 2. The model consists of a global database which is partitioned into a collection of local databases. These local databases stored at N sites which is connected through network. There is no replication of data items. There are one coordinator which generates the transactions and dispatch to the relevant sites (for which that request is). This coordinator is responsible for generating the workload for each data site and assigning time stamp to each transaction.

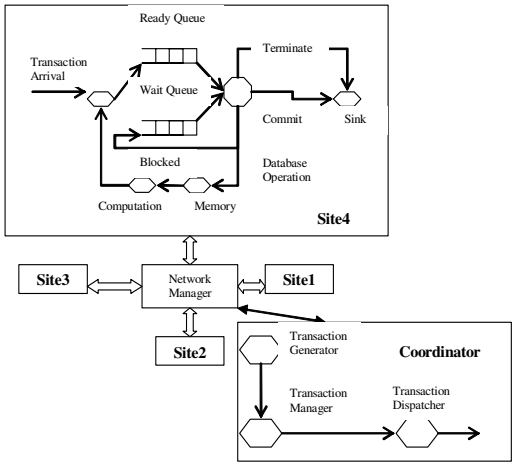

Fig. 2: Simulation Model

Each transaction in the system is distinguished by a globally unique transaction id. Each other processing site consists of transaction manager, a concurrency controller, a CPU, a ready queue, a local database, a communication interface, a sink, a wait queue. For every operation on the data object, it has to go through the concurrency control component to obtain a lock on the object. If the request is denied, the transaction is placed in to the wait queue. The waiting transaction will be awakened when the requested lock is released and all the locks are available. If the request of all the locks is granted, the transaction will access the memory and perform computation on data items. Transactions may commits or aborts and release all the locks it has been holding. The sink component of the model is responsible for gathering the statistics for the committed or terminated transactions.

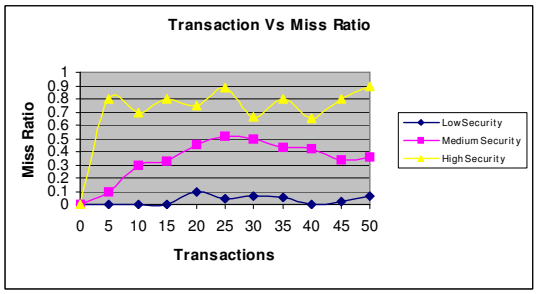

Fig. 3: Transactions Vs Miss Ratio

A Secure Concurrency Control Protocol: This section presents a global concurrency control mechanism based on the timestamp ordering. A transaction against the global schema issued at any component is handled by the GT_r. This is also act as coordinator. A GT_r is a software module which translates and decomposes the transaction against the global schema into subtransactions against local schemas, and coordinates the execution of the subtransactions.

Algorithm for write operation on data item x issued by subtransaction S_i with timestamp $T_s i$:

```
If ( RTs(x) > Ts i ) {
        Abort ( Si ) ;
} ElseIf ( WTs(x) > Ts i ) {
        Ignore ( Si );
}ElseIf( Lv(x) == Lv(Si) )        /* Lv(x) & Lv(Si) is
security level of data item x & transaction Si
*/
{
        WritelockTo( x );
        Execution( x );
        WTs(x) = Ts i ;
        Update DAT to Ts i ;
}Else{
        Abort( Si );/* access denied due to
security */
}
```

Algorithm for read operation on data item x issued by sub transaction S_i with timestamp $T_s i$:

```
If ( WTs(x) > Ts i ){
        Abort( Si );
        Rollback( Si );
}ElseIf( Lv(x) ≤ Lv(Si) ){
        ReadlockTo( x );
        ExecuteOn( x );
        RTs(x) = Ts i ;
        Update DAT to Ts i ;
}Else{
    Abort( Si );
    Rollback( Si );
}
```

If a global subtransaction is rolled back by the mechanism, it will cause all other subtransactions pertaining to the same global transaction to roll back as well. By rolling back a global transaction at the coordinator site, before sending its subtransactions to the relevant LT_r, the execution autonomy of LT_r will be enhanced [8]. This is the result of maintaining the DAT with the GT_r. However, global transactions are not likely to be rolled back frequently.

Performance Study: This section presents the performance results of our simulation experiments. The aim of the experiments is 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 concurrency protocol and that of secure concurrency protocol at three security levels, (i.e., high, medium and low). The throughput is the number of transactions committed per second. Thus, our primary performance measure is the proportion of missed deadlines or miss ratio (MR) which is defined as the percentage of input transactions that system is unable to complete on or before their deadlines, i.e., MR = number of transactions aborted / number of transactions submitted to system for processing.

Fig. 3 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 protocols initially increases with the increase in arrival rates then decreases when arrival rate becomes more than 5. However the overall throughput of secure concurrency protocol is always less than non- secure concurrency protocol. We also observe 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.

CONCLUSIONS

In this paper we have presented an algorithm for controlling the concurrency secure mode. This algorithm provides security to the data while providing a concurrent access to the data from database. It is observed that throughput of the system decreases as the security level of the transaction increases, i.e., the probability of successful execution of transactions is decreasing. It means there are a tradeoff between the security level and the throughput of the system.

We are in the process of investigating schemes by which the performance of high security level transactions can be improved without compromising with the security. Further we are looking to secure real time distributed systems by which the performance of high security level transactions can be improved without compromising the security.

REFERENCES

1. Ricardo, Catherine, 1990. Database Systems: Principles, Design and Implementation. MacMillan Publishers, New York.
2. K. Sugihara, 1987. Concurrency Control Based on Distributed Cycle Detection, In Proceedings of International Conference on Data Engineering, pp. 267-274.
3. N. Kaur, R. Singh, A. K. Sarje, M. Misra, 2005. Performance Evaluation of Secure Concurrency Control Algorithm for Multilevel Secure Distributed Database Systems. In IEEE Proceedings of the International Conference on Information technology: Coding and Computing (ITCC'05), 1: 249 – 254.
4. D.E. Bell and L.J. LaPadula, 1976. Secure Computer Systems: Unified Exposition and Multics Interpretation. The MITRE Corp.
5. T.F. Keefe, W.T. Tsai, J. Srivastava, 1993. Database Concurrency Control In Multilevel Secure Database Management Systems. IEEE Transaction on knowledge and Data Engineering, 5 (6) 1039-1055.
6. I. Ray, L. V. Mancini, S. Jajodia and E. Bertino,2000. ASEP: A Secure and Flexible Commit Protocol for MLS Distributed Database Systems. IEEE Transactions on Knowledge and Data Engineering, 12(6): 880 – 899.
7. S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz, 1992. The Concurrency Control Problem in Multidatabases: Characteristics and Solutions. ACM SIGMOD International Conference on Management of Data, San Diego, California, June 2-5, 1992, pp: 288-297.
8. S.M. Chung, K.A. Elghayegh, 1993. A Timestamp-Based Concurrency Control Algorithm for Heterogeneous Distributed Databases. In Proceedings of the Fifth International Conference on Computing and Information, May 27 - 29, 1993, pp: 438-442.