Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 6, June 2014, pg.700 – 707

RESEARCH ARTICLE

Guaranteeing Global Conflict Serializability in Concurrent Distributed Database Systems using Commitment Ordering

Teresa K. Abuya[1], Cheruiyot W.K[2]

[1]Computer Science, Kisii University, Kenya

[2]Computer Science, Jomo Kenyatta University of Agriculture & Technology, Kenya

[1] tkwambokaa@gmail.com; [2] wilchery68@gmail.com

Abstract*— A distributed database system is a single logical database that is spread physically across computers in multiple locations connected by a data communication network. Transaction management has become a difficult issue in distributed systems. The difficult arises due to transmission of transactions concurrently on the same database without a proper order which has led to global conflicts. Delays between the initiation of transactions and their commitment for an unknown duration is the most common challenge in distributed database systems with client server models. The application of common methods of concurrency controls like Two Phase Locking and optimistic methods in client server models has lead to long blocking duration, high abortions of transactions and increased waiting time of transactions. In this paper a top down approach is presented to ensure proper execution of global transactions across client server systems. The main contribution of this paper is to explore Commitment ordering applicable to distributed systems environment in order to address latency between transactions. A simulation model is developed that will guarantee global serialization of multiple resource managers.

***Keywords*— Commitment ordering, concurrency control, distributed systems, transaction management, global serialization**

I. Introduction

A Distributed Database (DDB) is a collection of multiple, logically interrelated databases distributed over a data communication network. A Distributed Database Management System (DDBMS) permits the management of DDB and makes the distribution transparent to the users. Each Local database system (LDBS) is controlled by a local database management system (LDBMS).One of the goals of distributed system is to ensure distribution of data across networks and servers connecting to it. The combination of vast amounts of data, real time constraints and necessity of high availability creates challenges for many aspects of database technology including distributed databases, database transaction processing, storage and query optimization. The performance, reliability and availability requirements of data access are demanding.

In recent years, growth has been experienced with regard to processing enormous data in distributed database systems. Many transactions are being carried out in a million seconds and therefore need proper execution and commitment order to deal with the problem of high abortion rates and long unpredictable blocking times by other transactions.

Ravindran [23], said that the objective of distributed database management system is to control the management of database in such a way that it appears to the user as a centralized database and that concurrency control co-ordinates concurrent access to a database in a multi-user database management system.

Ostermann [12], said that an effective transaction management system serves to maintain the acidity properties of transactions. He added that extensive research done on concurrency control mechanisms like Two Phase Locking (2PL), timestamp and optimistic concurrency control to determine their performance have not properly addressed the problem of global transaction concurrency control.

Different concurrency control mechanisms have been implemented to assess their performance on concurrent distributed database environment but the studies reveal problems with serializability and order among transactions. For instance, Two phase locking (2PL) concurrency control mechanism has been implemented in most commercial database systems and studies have shown that 2PL possesses an overall performance advantage over the non-locking concurrency control mechanisms such as timestamp and optimistic concurrency control mechanisms. However it has the drawback that it is susceptible to thrashing when the degree of concurrency reaches a certain limit. Similarly serializability graphs have been used to enforce serializability of transactions to ensure that only transactions without data conflicts with the executing transactions are allocated the CPU. However, the most critical trade-off in this approach is the delay of the starting of some transactions which is due to the waiting of a transaction until it has no data conflict with the executing transactions. This has caused starvation for many transactions on the queue. Timestamp ordering has not helped the situation either as experiments shows that they have a starvation problem for long transactions not guaranteeing recoverability in concurrent distributed systems.

In Multi-version Concurrent Control (MVCC) with optimistic and pessimistic results which relies on validation and locking uses snapshot isolation for testing serializability. This method however does not guarantee serializability in distributes systems as read and write logically occur at different times, reads at the beginning of transaction and writes at the end of transaction.

Concurrent execution of different transactions in a distributed client-server model has faced problems of delay between the time transactions are entered and the time transactions are committed. Rahmani [11], adds that, the timing constraints of real-time database are specified in the form of deadlines that require a transaction to be completed by a specified time. Failure to meet a deadline can cause the results to lose their value, and in some cases a result produced too late may be useless or even harmful and so unlike traditional database Real-time databases (RTDBMS) must maintain Temporal Consistency of data. Research shows that conventional pessimistic concurrency control based on locking with higher priority to ensure transaction serialization have been used to address this problem. However because of high rate of restart transactions it cannot satisfy the need of real time database systems very well.

Over the years extensive research has been done on concurrency control mechanisms like Two Phase Locking (2PL), timestamp, Serial Graph Testing and optimistic concurrency control to determine their performance on concurrent distributed systems. The studies reveal a problem with serializability and order among transactions. The problem arises due to transmission of transactions concurrently on the same database management system without a proper order. These results in delay between the transaction initiation and the time it's fully committed. All the methods have not addressed global conflict serializability problems conclusively.

Thus, the question of what concurrency control mechanism that should be implemented to achieve global concflict serializability in concurrent distributed database systems accomplishing proper and ordered coordination and execution of ernomous data transactions remains a question for investigation.The need therefore remains to come up with a concurrency control mechanism that clearly gives a roadmap to its implementation in concurrent distributed client-server system environment and address global conflict serializability problems.

II. Distributed Database System Model in Concurrent Transactions

A distributed database (DDB) allows multiple, logically interrelated databases to be distributed over a computer network in a tightly coupled multiprocessor system which resides at one of the nodes of a network of computers. As depicted in **figure 1.0** below all data is managed by a global distributed database management system. All users access data through the global distributed DBMS which is a union of all local Database Management Systems (LDBMS).Concurrent transactions in

distributed systems pose difficult problems especially when transactions are not committed in a particular order. The figure below shows how transactions are distributed across different database systems on a communication network.

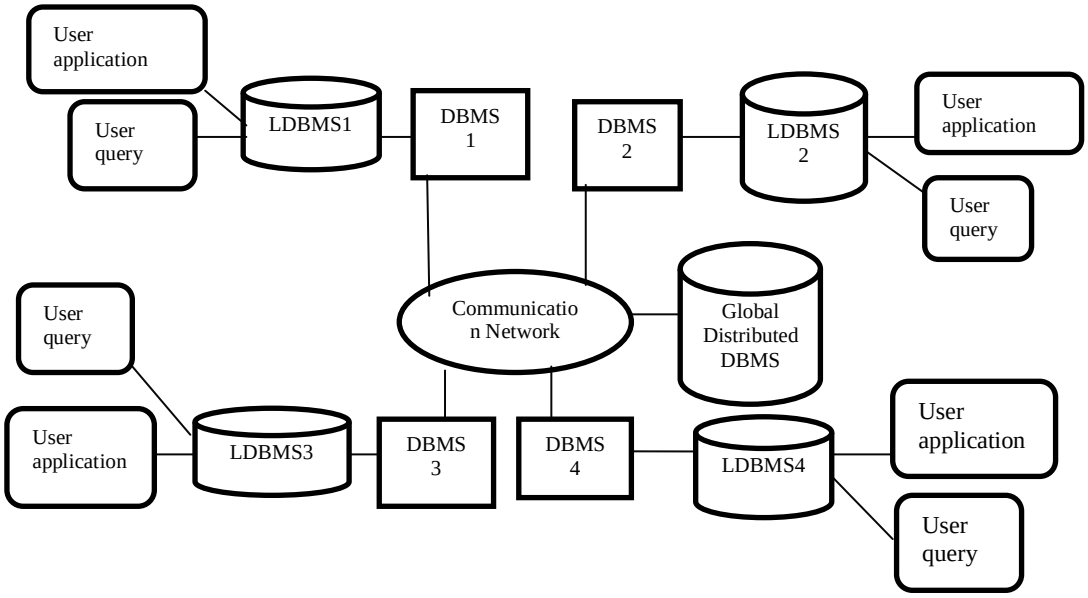

Fig 1.0 Overview of distributed database system model

The user query and user application are used in LDBMS for the client to query the systems what they intend to transacts and applications needed. Global distributed database systems ensure that all data is distributed accordingly depending on orders being executed.

III. Concurrency Control Anomalies in Distributed Systems

According to Bhowmick [16], the following anomalies arise during transaction processing:

Anomaly 1: The lost update problem: A second transaction writes a second value of a data item on top of a first value written by a first concurrent transaction, and the first value is lost to other transactions running concurrently which need, by their precedence, to read the first value. The transactions that have read the wrong value end with incorrect results.

Anomaly 2: The dirty read problem: Transactions read a value written by a transaction that has been later aborted. This value disappears from the database upon abort, and should not have been read by any transaction. The reading transactions end with incorrect results.

Anomaly 3: The incorrect summary problem: While one transaction takes a summary over the values of all the instances of a repeated data item, a second transaction updates some instances of that data item. The resulting summary does not reflect a correct result for any precedence order between the two transactions, but rather some random result, depending on the timing of the updates, and whether certain update results have been included in the summary or not.

Bhowmick [16],said that our main concern in designing a concurrency control algorithm is to correctly process transactions that are in conflict. Each transaction has a read set and a write set and two transactions conflict if the read set of one transaction intersects with the write set of the other transaction and/or the write set of one transaction conflicts with the write set of the other transaction. Jog'Schek[4], used the illustration in **fig.1.1** below to explain read and write sets in transactions. If read set $S(R_1)$ and write set $S(W_2)$ have some database entities in common, we say that the read set of T_1 conflicts with the write set of T_2.This is represented by the diagonal edge in the figure. Similarly if $S(R_2)$and $S(W_1)$have some database items in common, we draw the other diagonal edge. If $S(W_1)$ and $S(W_2)$ have some database items in common, we say that the write set of T_1 conflicts with the write set of T_2.This situation is represented by the horizontal edge at the bottom. We do not need to worry about the conflict between the read sets of the two transactions because read actions do not change the values of the database entities. It must be noted that transactions T_1 and T_2 can conflict only if both are executing at the same time. If, for example, T_1 has finished before T_2 was submitted to the system, even if their read and write sets intersect, they are not considered to be in conflict.

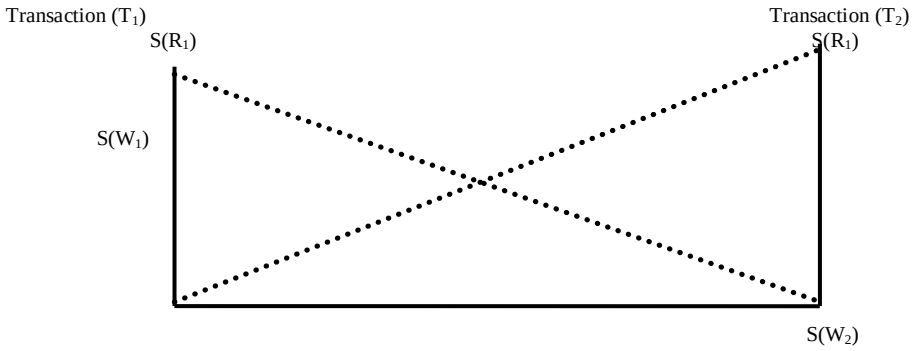

Fig.1.1 types of conflict for two transactions Jog'Schek [4]

Jog'Schek[4], further said that transactions may be online queries expressed in a self-contained query language, or application programs written in a general purpose programming language. However Ali[20], says that the concurrency control algorithms pay no attention to the computations performed by transactions. Instead, these algorithms make all of their decisions on the basis of the data items a transaction reads and writes, and so details of the form of transactions are unimportant in our analysis. However we do assume that transactions represent complete and correct computations; each transaction, if executed alone on an initially consistent database, would terminate, produce correct results, and leave the database consistent. The logical read set of a transaction is the set of logical data items the transaction reads or writes. Similarly, stored read sets and stored write sets are the stored data items that a transaction reads and writes.

IV. Analysis of Concurrency Control Mechanisms

Serialization Graph Testing (SGT)

Alonso[3], said that, Serializability graph enforces the serializability of transactions to ensure that only transactions without data conflicts with the executing transactions will be allocated the CPU. Therefore, the basic principle is that a transaction is not scheduled until it has no data conflict with the executing transactions in the system. Consequently, no conflict resolution is needed for the executing transactions and the problem of transaction restart is eliminated. Moreover, a transaction can be executed without blocking such that the time spent in the system can be minimized. Obviously, the immediate benefit is that, all the system resources can be used more effectively to contribute to the commitments of transactions within deadlines Rajamani[18]. Since there is no data conflict among the executing transactions, the system can enjoy more flexibility in scheduling the transactions. The most critical tradeoff in this approach is the delay of the starting of some transactions. The delay is due to the waiting of a transaction until it has no data conflict with the executing transactions. This has caused starvation for many transactions on the queue.

B. Timestamp ordering Concurrency Control

According to Rajamani [18], timestamp concurrency control uses timestamps for synchronization instead of lock. From the outside it seems that the transactions are executed sequential according to their starting time. In other words, the scheduler generates serializable schedules that are equal to the serial execution of the transactions ordered by their starting time. To achieve this, the transaction manager assigns a timestamp TS (Ti)to each transaction Ti at its start and guarantees that the timestamp of transactions that started later is always higher than the timestamps of all earlier transactions. These timestamps are used to guarantee the Timestamp Ordering (TO) rule: if two operations mi(x) and nj (x) are in conflict, i.e. they access the same tuple x and at least one operation is a write operation, then the operation of the transaction with the lower timestamp is always executed first. Thereby, the resulting schedule is equal to the serial execution of the transactions ordered by their timestamp and, as a consequence, it is serializable, Bornea[17].According to Lingam[9], Multi-version Timestamp Ordering (MVTO) technique is one type of optimistic concurrency Control (OCC) mechanism which provides a large degree of concurrency for the transactions by maintaining multiple versions of data items and so it is more appropriate for real-time database systems where the transaction has a low rate of restart and delay of cut-off time but a high degree of concurrency. According to Triplett [7], timestamp ordering concurrency control mechanisms were considered to be quite suitable for distributed database systems, since transactions to be rolled back can be determined locally at each site. Experiments, however, have shown that timestamp ordering mechanisms do not seem to be efficient and have a starvation problem for long transactions not guaranteeing recoverability in concurrent distributed systems.

c.Two phase locking (2PL): Two-phase locking (2PL) concurrency control mechanism has been implemented in most commercial database systems. Studies carried out for its performance including both simulation and analytical modeling revealed that 2PL possesses an overall performance advantage over the non-locking concurrency control mechanisms, such as timestamp and optimistic concurrency control mechanisms. However, it has the drawback that it is susceptible to thrashing when the degree of concurrency reaches a certain limit therefore not achieving global serialization, Laiho & Laux[10]. To overcome this drawback, many attempts have been made to enhance the performance of the two-phase locking mechanism. These attempts include introducing new variants of 2PL which are classified as one being concerned with conflict resolution which can be resolved by either blocking the requesting transaction or aborting the requesting transaction. Between these two extremes are mechanisms that balance between the two, using either blocking, or restarts to resolve conflicts. Examples of these mechanisms are, wait-die, wound-wait, running priority, cautious waiting, and the wait-depth-limited concurrency control mechanism. However, 2PL has some inherent problems such as possibility of deadlocks as well as long and unpredictable blocking times and it is susceptible to thrashing when the degree of concurrency reaches a certain limit, Nizamuddin & Sattar[19].

The conventional pessimistic concurrency control mechanism based on locking e.g. two phase locking with higher priority can assure the transactions serializability, so as to strongly assure the consistency of data. However, because of a high rate of restart of transactions, it cannot satisfy the need of the real-time database systems very well as it assumes that the probability of any two concurrent transactions requesting the same data is not often. So it allows all operations to be performed directly whenever transactions request Palmieri[21].

From the above concurrency control mechanisms experimented and extensively researched on have not been able to solve the problem of global conflict serializability.

V. Critical Analysis of Concurrency Control Mechanisms

The following key points have been noted from the above concurrency control mechanisms. The mechanisms reveal that achieving global conflict serializability with reasonable performance especially across Resource Managers has been considered a difficult problem.

In serialization graph testing (SGT), no conflict resolution is needed for the executing transactions and the problem of transaction restart is eliminated. Though a challenge is that the most critical tradeoff in this approach is the delay of the starting of some transactions which is due to the waiting of a transaction until it has no data conflict with the executing transactions. This has led to too much starvation of next transactions or low completion rates.

Timestamp ordering concurrency control mechanisms were considered to be quite suitable for distributed database systems, since transactions to be rolled back can be determined locally at each site. Experiments, however, have shown that timestamp ordering mechanisms do not seem to be efficient and have a starvation problem for long transactions not guaranteeing recoverability in concurrent distributed systems. Two-phase locking (2PL) concurrency control mechanism has been implemented in most commercial database systems. Studies carried out for its performance including both simulation and analytical modeling revealed that 2PL possesses an overall performance advantage over the non-locking concurrency control mechanisms, such as timestamp and optimistic concurrency control mechanisms. However, 2PL has some inherent problems such as possibility of deadlocks as well as long and unpredictable blocking times and it is susceptible to thrashing when the degree of concurrency reaches a certain limit.

Osterman[12], proposed a site queue concurrency control model in distributed systems to address global serializability problem as shown in fig.1.2 below.

Key
GTM: Global Transaction Memory
LTM: Local Transaction Memory
LDBS: Local Database System

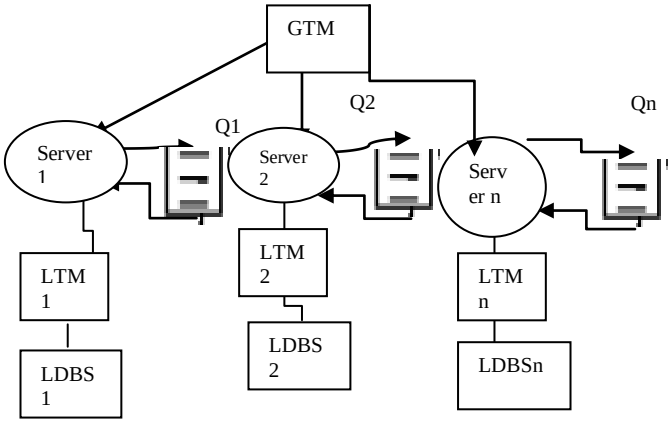

Fig 1.2 The site queue of a Heterogeneous Distributed Database Systems (HDBS)

This concurrency control approach has a server which is created to maintain a sub-transaction queue. The server and the queue constitute the global-local interface between the LDBS and the GTM. Each global transaction is decomposed into sub-transaction whereby the GCC determines an order among global transactions to the proper server according to some predetermined way. If a server receives a sub-transaction it will insert the sub-transaction to the rear of its queue and previously submitted transactions are serialized or aborted. Finally the results of all transactions are submitted to the GTM. The site queue concurrency control approach used in heterogeneous distributed systems shown in **fig.1.2** above is a bottom up approach showing how transactions are carried out in LDBS and studies generally show that the bottom up approach suffers from a high abortion rate of the global transactions therefore not guaranteeing global conflict serialization.

VI. The Proposed Solution

The main purpose of this study is to guarantee global conflict serialization in concurrent distributed database systems by decreasing long blocking duration, reducing high abortion rates of transactions and address latency between multiple resource managers.

This is achieved by using a serializability technique called Commitment Ordering(CO) which was introduced by Raz[2]from digital equipment corporation. Commitment Ordering provides an effective general solution for global conflict serializability across any collection of database systems and other transactional objects.

A. The concept of commitment ordering.

Newcomer [13], in his principles of transaction processing said that, Commitment Ordering (CO) is a serializability concept, that allows global serializability to be effectively achieved across multiple distributed Resource Managers (RMs) which may use different concurrency control mechanisms. This paper presents a conflict-free Transaction Model which is a vehicle for exploring benefits of using commitment ordering to enforce global serialization in concurrent distributed systems. Commitment Ordering intent is that transactions' sequential direction of effect happenings is held harmoniously with their individual right-of-way direction. Thus, Commitment Ordering provides a solution for the long standing global serializability problem in concurrent distributed systems. It ensures that no concurrency control information is shared with other entities, except Atomic Commitment Protocol (ACP) and it is a necessary condition for guaranteeing global serializability across Resource Managers. Commitment Ordering exhibits a deadlock free execution when implemented with global serializability graphs on client server systems. Since Commitment Ordering can be enforced both locally and globally by effective algorithms that only order commit events and do not interfere with the databases and other Concurrency Control mechanisms operations it is able to put to rest the unsolved issues related to delay in transaction execution. Thus it can significantly improve the concurrency of transactions as well as increase the number of transactions, Raz[2].

B. Top down approach

A top down approach of concurrency control is presented to achieve both global concurrency control (GCC) and local concurrency control (LCC).In topdown approach a GCC determines a global serialization order of global transactions before submitting them to local sites.This order is enforced at all local sites by either LCC or the GCC or both.

The reason why top down approach is preffered is that it has an important property i.e the serializability of global executions.**Fig 1.3** presents a top down model solution that guarantees global conflict serializability.

The proposed conceptual model

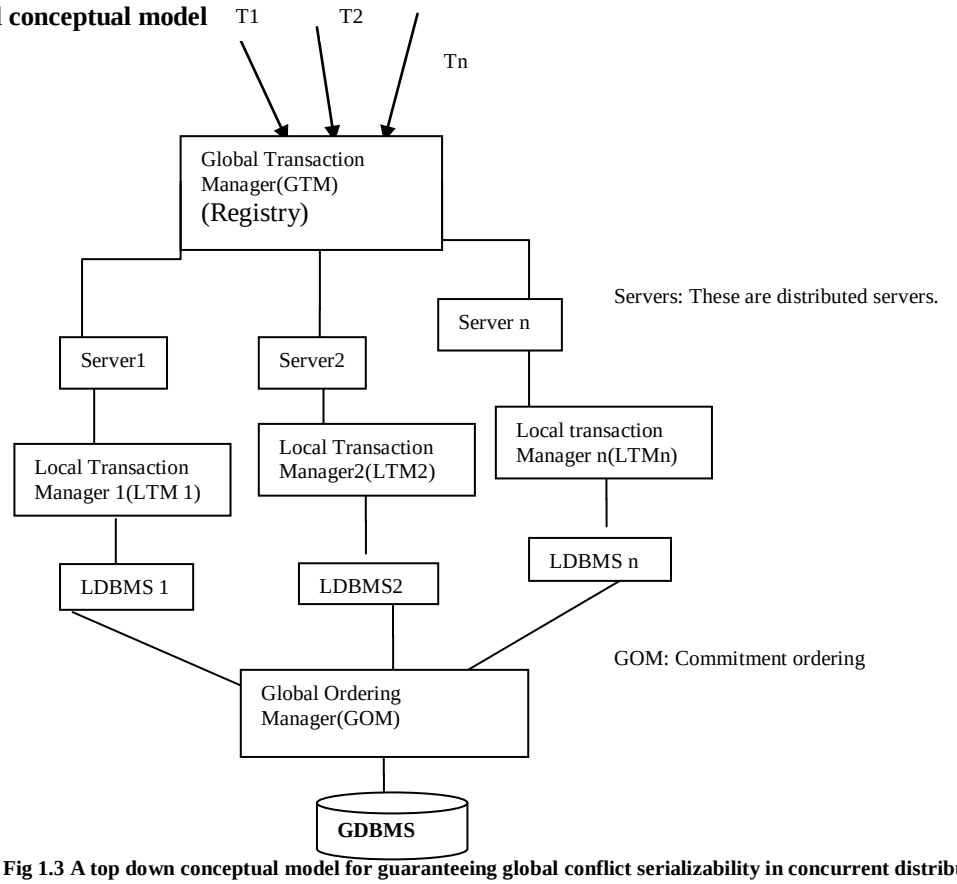

Fig 1.3 A top down conceptual model for guaranteeing global conflict serializability in concurrent distributed database systems

The figure above shows a conceptual model for global transactions in a distributed database environment. A global transaction is a set of global subtransactionswhere each global subtransaction is a transaction accessing data items at a single local site.A local transaction is a transaction that accesses the data items at a single local site..GTM is responsible for global transactions to ensure global serializability and Local Transaction Manager(LTM) is responsible for local transactions and subtransactions of global transactions executing at its site.A server is created to maintain a subtransaction queue.Each server serves a particular department where it gets data from the Global Transaction Manager (GTM).Each sub-query is transported to Local Transaction Manager checked for local correctness and then passed down to Local Database Management System.The server consists the global-local interface between the LDBS and the GTM.Each global transaction is decomposed into a set of subtransactions.The GCC first determines an order among global transactions and then submits the subtransactions of global transactions to the proper server according to the pre-determined global transaction order.If a server receives a subtransaction,it will submit it to the LTM and then to the LDBS and finally the results are sent to the Global Ordering Manager(GOM).
At the Global Ordering Manager transaction records are in no particular order.They are then put in a particular order of committing by assigning them address numbers such that the final information at the GDBMS is arranged in either ascending or descending order. Any potential problem, e.g. global dead lock, is resolved by GOM after gathering information from all the Participating sites.This is very important because concurrent distributed systems provides different values which cannot be committed at the global database at the same time since the database has a single read/write head. They then commit to the Global Database Management system(GDBMS).

VII. Conclusion

Concurrency is when multiple activities occur during the same time interval. It is a natural phenomenon; in the real world many things are happening simultaneously. Global concurrency control is required in order to allow concurrent global updates in concurrent distributed database systems. Two phase locking (2PL), timestamp, optimistic and pessimistic concurrency control mechanisms have some inherent problems such as possibility of deadlocks as well as long and unpredictable blocking times and high rate of abortions which do not guarantee serializability. To overcome this problem, a top down approach emerges as the viable alternative for ensuring proper concurrent execution of global transactions in concurrent distributed systems. This is crucial in addressing latency between transactions on a client server system. The main contribution of this paper is to implement

Commitment ordering in a distributed database systems environment in order to address latency between transactions. A conceptual model is developed that guarantees global conflict serialization of multiple resource managers.

References

[1] Elmasri, R.,and S. Navathe. 2006. *Fundamentals of Database Systems*, 5th ed. Menlo Park, CA: Benjamin Cummings.

[2] Yoav Raz (2009): *Theory of Commitment Ordering* - Summary Google Sites - Site of Yoav Raz. Retrieved 1 Feb, 2011.

[3] R. Alonso, H. Garcia Molina, and L. Salem. *Concurrency control and recovery for global procedures in federated database systems*. In IBBE Data Engineering Bulletin, pages 5-11, September 2007

[4] Laura Cristiana Voicu, Heiko Schuldt, Fuat Akal, Yuri Breitbart, Hans Jörg Schek (2009): "Re:GRIDiT-Co-rdinating Distributed update transactions on replicated data in the grid", 10th IEEE/ACM International Conference on Grid Computing (Grid 2009), Banff, Canada, 2009/10.

[5] Thomas Neumann. *Efficiently Compiling Efficient Query Plans for Modern Hardware*. PVLDB, 4(9):539–550, 2011.

[6] Yoav Raz (2006): *Theory of Commitment Ordering* - Summary GoogleSites - Site of Yoav Raz. Retrieved 1 Feb, 2011.

[7] Josh Triplett, Paul E. McKenney, and Jonathan Walpole. *Scalable Concurrent Hash Tables via Relativistic Programming.*ACM SIGOPS Operating Systems Review,44(3):102–109, August 2010

[8] Alfons Kemper and Thomas Neumann. HyPer: A hybrid OLTP&OLAP *main memory database system based on virtual memory snapshots*. InICDE, pages195–206, 2011

[9] K. M. Prakash Lingam, *"Freezing as a correctness measure for multi-version timestamp ordering protocol*", Proceedings of 2nd International Conference on Compute r Engineering and Technology,vol.3, pp.312-316, 2010.

[10] M. Laiho , F. Laux, "*Implementing Optimistic Concurrency control for persistence Middleware using row version verification"*, Proceeding of the Second International Conference on Advances in databases, knowledge and Data Applications, pp. 45-50, 2010

[11] M. Hedayati, S. H. Kamali, R. Shakerian, M. Rahmani , *"Evaluation of performance concurrency control algorithm for secure firm real- time database systems via simulation model",*Proceeding of the International conference on Networking and Information Technology, pp. 260-264, 2013.

[12] W.Du,A. Elmagarmid, Y. Leu, and S. Ostermann. Effects of autonomy on global concurrency control in heterogeneous distributed database systems.Gaithersburg, MD, October 2009.

[13] Philip A. Bernstein, Eric Newcomer (2009): *Principles of Transaction Processing*, 2nd Edition, Morgan Kaufmann (Elsevier), June 2009, ISBN 978-1-55860-623-4 (pages 145, 360)

[14] W.Du, A. Elmagarmid,Y. Leu, and S.Ostermann. *Effects of autonomy on global concurrency control in heterogeneous distributed database systems*. In Proceedings of the Second Inter-national Conference on Data and Knowledge Systems for Manufacturing and Engineering, Gaithersburg, MD, 2009

[15] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Databases Systems*. Addison- Wesley Publishing Co., 2009

[16] Sanjay Kumar Madria, Mohammed Baseer,Vijay Kumar, Sourav S. Bhowmick: *A Transaction Model and Multiversion Concurrency Control for Mobile Database Systems, Dis-tributed and Parallel Databases,* 22(2-3), 2007.

[17] Mihaela A. Bornea, Orion Hodson, Sameh Elnikety, Alan Fekete: *One-Copy Serializability with Snapshot Isolation un-der the Hood*. ICDE, 2011.

[18] Salman Abdul Moiz ,Dr. Lakshmi Rajamani, "*Concurrency Control Strategy to Reduce Frequent Rollbacks in Mobile Environments*," CSE, vol.2, pp.709- 714, 2009 International Conference on Computational Science and Engineering, 2009.

[19] Mohammed Khaja Nizamuddin, Syed Abdul Sattar, "*Adaptive Valid Period Based concurrency Control In: Recent Without Locking in Mobile Environments" Trends in Networks and Communications,* Springer CCIS, Vol.90, Part 2, pp 349- 358, 2010, Springer Heidelberg.

[20] K.Ali and b.Ahmad,"*A Concurrency Control Method Based on Commitment Ordering in MobileDatabases", International Journal of Database Management Systems* IJDMS) vol.3, no.4, November 2011 .

[21] R. Palmieri, F. Quaglia, and P. Romano, "Osare: *Opportunistic speculation in actively replicated transactional systems*," in SRDS, 2011.

[22] M. M. Saad and B. Ravindran, "Transactional forwarding: *Supporting highly-concurrent in asynchronous distributed systems*," inSBAC-PAD, 2012.

[23] A. Turcu and B. Ravindran, "*On Open Nesting in Distributed Transactional Memory,"* in SYSTOR, 2012.

[24] J. Kim, R. Palmieri, and B. Ravindran, "*Scheduling open-nested transactions in distributed transactional memory,"* in COORDINATION. Springer,2013.

