Volume 2, Issue 6, June 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Transaction Management in Homogenous Distributed Real-Time Replicated Database Systems

Ashish Srivastava[1], Udai Shankar2, Sanjay Kumar Tiwari[3]
Deptt. of Computer Science & Engineering
Madan Mohan Malaviya Engineering College,
Gorakhpur, India.

Abstract— **A homogenous distributed real time replicated database system is a network of two or more DBMS that reside on one or more machines. A distributed system that connects two or more databases are Homogenous Distributed Database Systems (HDDBS) create different problems when accessing distributed and replicated databases. Particularly, access control and transaction management in HDDBS require different mechanism to monitor data retrieval and update to databases. Current trends in multi-tier client/server networks make DDBS an appropriated solution to provide access to and control over localized databases. This paper highlights the basic concepts underlying distributed database system including transaction management in in HDRTDBS. Distributed database systems (DDBS) create different problems when accessing distributed and replicated databases. Particularly, access control and transaction management in DDBS require different mechanism to monitor data. Retrieval and update to databases and reduces the communication traffic and also achieves a good transaction response time. An example is given to demonstrate the step involved in executing the two-phase commit.**

Keywords— **Database system, Real time system, replicated database system, Transaction management, two-phase commit, homogenous distributed database system etc.**

I. INTRODUCTION

A real-time system is one that must process information and produce a response within a specified time, else risk severe consequences, including failure. That is, in a system with a real-time constraint it is no good to have the correct action or the correct answer after a certain deadline: it is either by the deadline or it is useless. Database replication based on group communication systems has been proposed as an efficient and flexible solution for data replication. Distributed data base system is a technique that is used to solve a single problem in a heterogeneous computer network system. A major issue in building a distributed database system is the transactions atomicity. When a transaction runs across into two sites, it may happen that one site may commit and other one may fail due to an inconsistent state of transaction. Two-phase commit protocol is widely used to solve these problems. The choice of commit protocol is an important design decision for distributed database system. A commit protocol in a distributed database transaction should uniformly commit to ensure that all the participating sites agree to the final outcome and the result may be either a commit or an abort situation. Many real time database applications are distributed in nature [1,11,12] These include the aircraft control, stock trading, network management, factory automation etc . The real time performance of Real time distributed database system (RTDBS) depends on *several* factors such as the database system architecture ,the underlying processor, disk speed etc.

II. DISTRIBUTED DATABASE SYSTEMS (DDBS)

Distributed database systems (DDBS) are systems that have their data distributed and replicated over several locations; unlike the centralized data base system (CDBS), where one copy of the data is stored. Data may be replicated over a network using horizontal and vertical fragmentation similar to projection and selection operations in Structured Query Language (SQL). Both types of database share the same problems of access control and transaction management, such as user concurrent access control and deadlock detection and resolution. On the other hand however, DDBS must also cope with different problems. Access control and transaction management in DDBS require different rules to monitor data retrieval and update to distributed and replicated databases[2,7].Oracle, as a leading Database Management Systems (DBMS) employs the two-phase commit technique to maintain a consistent state for the databases. The objective of this paper is to explain transaction management in DDBMS and how to implements this technique. To assist in understanding this process, an example is given in the last section. It is hoped that this understanding will encourage organizations to use and academics to discuss DDBS and to successfully capitalize on this feature of Database. The next section presents advantages, disadvantages, and failures in Distributed Database Systems. Subsequent sections provide discussions on the fundamentals of transaction management, two-phase commit, homogenous distributed

database system implementation of the two-phase commit, and, finally, an example on how the two phases commit works.

A. Advantages of Distributed Database system(DDBS)

Since organizations tend to be geographically dispersed, a DDBS fits the organizational structure better than traditional centralized DBS. Improved Availability-A failure does not make the entire system inoperable and Improved Reliability-Data may be replicatedEach location will have its local data as well as the ability to get needed data from other locations via a communication network. Moreover, the failure of one of the servers at one site won't render the distributed database system inaccessible. The affected site will be the only one directly involved with that failed server. In addition, if any data is required from a site exhibiting a failure, such data may be retrieved from other locations containing the replicated data. The performance of the system will improve, since several machines take care of distributing the load of the CPU and the I/O. Also, the expansion of the distributed system is relatively easy, since adding a new location doesn't affect the existing ones.

B. Disadvantages of Distributed DBS

On the other hand, DDBS has several disadvantages. A distributed system usually exhibits more complexity and cost more than a centralized one. Security-network must be made secure Integrity Control More Difficult This is true because the hardware and software involved need to maintain a reliable and an efficient system. All the replication and data retrieval from all sites should be transparent to the user. The cost of maintaining the system is considerable since technicians and experts are required at every site. Another main disadvantage of distributed database systems is the issue of security. Handling security across several locations is more complicated. In addition, the communication between sites may be tapped to.

C. Types of DDBMS

1) Homogeneous DDBMS

- All sites use same DBMS product (eg. Oracle)
- Fairly easy to design and manage.

2) Heterogeneous DDBMS

- Sites may run different DBMS products .
- Possibly different underlying data models .
- Occurs when sites have implemented their own databases and integration is considered later.
- We won't consider heterogeneous DDBMSs here.

D. Failures in Distributed DBS

Several types of failures may occur in distributed database systems:

Transaction Failures*:* When a transaction fails, it aborts. Thereby, the database must be restored to the state it was in before the transaction started. Transactions may fail for several reasons. Some failures may be due to deadlock situations or concurrency control algorithms.

Site Failures: Site failures are usually due to software or hardware failures. These failures result in the loss of the main memory contents. In distributed database, site failures are of two types:

1. *Total Failure* where all the sites of a distributed system fail,
2. *Partial Failure* where only some of the sites of a distributed system fail.

Media Failures: Such failures refer to the failure of secondary storage devices. The failure itself may be due to head crashes, or controller failure. In these cases, the media failures result in the inaccessibility of part or the entire database stored on such secondary storage.

Communication Failures: Communication failures, as the name implies, are failures in the communication system between two or more sites. This will lead to network partitioning where each site, or several sites grouped together, operates independently. As such, messages from one site won't reach the other sites and will therefore be lost. The reliability protocols then utilize a timeout mechanism in order to detect undelivered messages. A message is undelivered if the sender doesn't receive an acknowledgment. The failure of a communication network to deliver messages is known as performance failure.

III. FUNDAMENTALS OF TRANSACTION MANAGEMENT

Transaction Management deals with the problems of keeping the database in a consistent state even when concurrent accesses and failures occur, [6].

A. What is a Transaction?

A transaction consists of a series of operations performed on a database. The important issue in transaction management is that if a database was in a consistent state prior to the initiation of a transaction, then the database should return to a consistent state after the transaction is completed. This should be done irrespective of the fact that transactions were successfully executed simultaneously or there were failures during the execution,[5.7].

A transaction is a sequence of operations that takes the database from a consistent state to another consistent state. It represents a complete and correct computation. Two types of transactions are allowed in our environment: query transactions and update transactions. Query transactions consist only of read operations that access data objects and return their values to the user. Thus, query transactions do not modify the database state. Two transactions conflict if the read-set of one transaction intersects with the write-set of the other transaction. During the voting process ,Update transactions consist of both read and write operations. Transactions have their time-stamps constructed by adding 1 to the greater of either the current time or the highest time-stamp of their base variables.Thus; a transaction is a unit of consistency and reliability. The properties of transactions will be discussed later in the properties section. Each transaction has to terminate. The outcome of the termination depends on the success or failure of the

transaction. When a transaction starts executing, it may terminate with one of two possibilities:

1. The transaction **aborts** if a failure occurred during its execution

2. The transaction **commits** if it was completed successfully example of a transaction that aborts during process 2 (P2). On the other hand, an example of a transaction that commits, since all of its processes are successfully completed [8.9].

B. Properties of Transactions

A Transaction has four properties that lead to the consistency and reliability of a distributed data base. These are Atomicity, Consistency, Isolation, and Durability[6].

Atomicity. This refers to the fact that a transaction is treated as a unit of operation. Consequently, it dictates that either all the actions related to a transaction are completed or none of them is carried out. For example, in the case of a crash, the system should complete the remainder of the transaction, or it will undo all the actions pertaining to this transaction. The recovery of the transaction is split into two types corresponding to the two types of failures: the transaction recovery, which is due to the system terminating one of the transactions because of deadlock handling; and the crash recovery, which is done after a system crash or a hardware failure.

Consistency. Referring to its correctness, this property deals with maintaining consistent data in a database system. Consistency falls under the subject of concurrency control. For example, "dirty data" is data that has been modified by a transaction that has not yet committed. Thus, the job of concurrency control is to be able to disallow transactions from reading or updating "dirty data."

Isolation. According to this property, each transaction should see a consistent database at all times. Consequently, no other transaction can read or modify data that is being modified by another transaction. If this property is not maintained, one of two things could happen to the data base.

a. Lost Updates: this occurs when another transaction (T2) updates the same data being modified by the first transaction (T1) in such a manner that T2 reads the value prior to the writing of T1 thus creating the problem of loosing this update.

b. Cascading Aborts: this problem occurs when the first transaction (T1) aborts, then the transactions that had read or modified data that has been used by T1 will also abort.

Durability. This property ensures that once a transaction commits, its results are permanent and cannot be erased from the database. This means that whatever happens after the COMMIT of a transaction, whether it is a system crash or aborts of other transactions, the results already committed are not modified or undone.

IV. TRANSACTION MANAGEMENT ON REPLICATED DATA IN THE DDBMS

The term replication refers to the operation of copying and maintaining database objects in multiple databases belonging to a distributed system. The terms distributed database system and database replication are related, yet distinct. In a pure (that is, not replicated) distributed database, the system manages a single copy of all data and supporting database objects. Typically, distributed database applications use distributed transactions to access both local and remote data and modify the global database in real-time. While replication relies on distributed database technology, database replication offers applications benefits that are not possible within a pure distributed database environment. Most commonly, replication is used to improve local database performance and protect the availability of applications because alternate data access options exist.[13,14,16] For example, an application may normally access a local database rather than a remote server to minimize network traffic and achieve maximum performance. Furthermore, the application can continue to function if the local server experiences a failure, but other servers with replicated data remain accessible. A new component, which is a *replication manager* module, has been recently added to the system, in order to maintain replicated data. In addition, the *local client* module was created to manage the local transactions operating on replicated data items[16,17]. The communication between these two modules is achieved by mechanisms of active databases: triggers, database procedures, database events and, via tuple space. At the present stage of the researches there have been two models of replicated data – primary/secondary and identical copies – implemented in the experimental system. In this method has been implemented in which write operations are executed on the primary copy and next, committed writes are collected and sent to all other copies as independent transactions. There have been two methods used in the second case, ROWA (Read One Write All) – a read operation may be executed on an arbitrary copy but a write operation has to be executed on every copy. To achieve this aim the two–phase commit protocol (2PC) [1] has been implemented in the experimental distributed database management system. Communication between a client and local servers goes through TS. All modules of the application processor have access to TS. The analysis of the 2PC protocol in the context of the tools used, made. The basic structure of the DDBMS. it possible to reduce the number of vote requesting messages and the number of messages with the client decision to commit or abort the transaction. Instead of sending messages to each local server the client outputs to TS only one tuple containing the decision. All participants read the tuple, without removing it from TS.IND (Independent) – copies are updated independently but the possibility of transaction commitment is consulted with the replication manager through tuple space. The replication manager queues all transactions referring to the replicated data. A new transaction at the given node may be committed if all the transactions from the replication manager queue had been realized prior to it. The analysis of replication manager work has shown that this module could be the "bottleneck" of the whole system. As the result of this, the second concept has been proposed, in which replication manager has been removed and its queue was placed in TS as a set of tuples . In connection with this, two terms have been suggested *global replicated data version* – being a number of transactions performed on these data in the whole system , and l*ocal replicated data version* – being a number of transactions performed at the local node. In the new concept the commitment of a transaction realized by a local client is possible if the global and given local version

of replicated data are equal. In the other case transactions making a difference between versions must be done at first. An access to these transactions goes through a FIFO queue.

The placement of a new transaction in this queue must be preceded by obtaining a tuple, containing a global replicated data version. This mechanism protects from concurrent changing of the contents of the queue. A problem may occur if a process that has got the global version goes down. To deal with it transactions on TS have been used. They are designed to provide reversibility for a group of crucial operations in the event of some sort of problem before they are all completed successfully. If there is a problem anywhere along the line, then the Paradise server restores a tuple space to the state it was in before the transaction started; removed tuples will reappear, and no added tuples will appear.

V. TWO-PHASE COMMIT PROTOCOL

The 2-phase commit (2PC) protocol is a distributed algorithm to ensure the consistent termination of a transaction in a distributed environment. Thus, via 2PC an unanimous decision is reached and enforced among multiple participating servers whether to commit or abort a given transaction, thereby guaranteeing atomicity. The protocol proceeds in two phases, namely the prepare and the commit phase, which explains the protocol's name. The protocol is executed by a coordinator process, while the participating servers are called participants. When the transaction's initiator issues a request to commit the transaction, the coordinator starts the first phase of the 2PC protocol by querying—via prepare messages—all participants whether to abort or to commit the transaction

The master initiates the first phase of the protocol by sending ***PREPARE*** (to commit) messages in parallel to all the cohorts. Each cohort that is ready to commit first force-writes a ***prepare*** log record to its local stable storage and then sends a ***YES*** vote to the master. At this stage, the cohort has entered a ***prepared*** state wherein it cannot unilaterally commit or abort the transaction but has to wait for the final decision from the master. On the other hand, each cohort that decides to abort force-writes an ***abort log*** record and sends a NO vote to the master. Since a NO vote acts like a veto, the cohort is permitted to unilaterally abort the transaction without waiting for a response from the master.

After the master receives the votes from all the cohorts, it initiates the second phase of the protocol. If all the votes are ***YES,*** it moves to a ***committing*** state by force writing a ***commit*** log record and sending ***COMMIT*** messages to all the cohorts. Each cohort after receiving a ***COMMIT*** message moves to the committing state, force-writes a ***commit log*** record, and sends an ***ACK*** message to the master. If the master receives even one ***NO*** vote, it moves to the aborting state by force-writing an ***abort log*** record and sends ***ABORT*** messages to those cohorts that are in the prepared state. These cohorts, after receiving the ***ABORT*** message, move to the ***aborting*** state, force write an ***abort log*** record and send an ***ACK*** message to the master.

Finally, the master, after receiving acknowledgements from all the prepared cohorts, writes an ***end*** log record and then "forgets" the transaction. The 2PC may be carried out with one of the following methods: Centralized 2PC, Linear 2PC, and Distributed 2PC, [7,15].

C. The Centralized Two-Phase Commit Protocol

In the Centralized 2PC shown in Figure 3, communication is done through the coordinator's process only, and thus no communication between subordinates is allowed. The coordinator is responsible for transmitting the PREPARE message to the subordinates, and, when the votes of all the subordinates are received and evaluated, the coordinator decides on the course of action: either abort or COMMIT. This method has two phases:

1. First Phase: In this phase, when a user wants to COMMIT a transaction, the coordinator issues a PREPARE message to all the subordinates, (Mohan et al., 1986). When a subordinate receives the PREPARE message, it writes a PREPARE log and, if that subordinate is willing to COMMIT, sends a YES VOTE, and enters the PREPARED state; or, it writes an abort record and, if that subordinate is not willing to COMMIT, sends a NO VOTE. A subordinate sending a NO VOTE doesn't need to enter a PREPARED state since it knows that the coordinator will issue an abort. In this case, the NO VOTE acts like a veto in the sense that only one NO VOTE is needed to abort the transaction. The following two rules apply to the coordinator's decision, (Ozsu et al., 1991):

a. If even one participant votes to abort the transaction, the coordinator has to reach a global abort decision.

b. If all the participants vote to COMMIT, the coordinator has to reach a global COMMIT decision.

2. Second Phase: After the coordinator reaches a vote, it has to relay that vote to the subordinates. If the decision is COMMIT, then the coordinator moves into the committing state and sends a COMMIT message to all the subordinates informing them of the COMMIT. When the subordinates receive the COMMIT message, they, in turn, move to the committing state and send an acknowledge (ACK) message to the coordinator. When the coordinator receives the ACK messages, it ends the transaction. If, on the other hand, the coordinator reaches an ABORT decision, it sends an ABORT message to all the subordinates. Here, the coordinator doesn't need to send an ABORT message to the subordinate(s) that gave a NO VOTE.

D. The Linear Two-Phase Commit Protocol

In the linear 2PC, as depicted in Figure 4, subordinates can communicate with each other. The sites are labeled 1 to N, where the coordinator is numbered as site 1. Accordingly, the propagation of the PREPARE message is done serially. As such, the time required to complete the transaction is longer than centralized or distributed methods. Finally, node N is the one that issues the Global COMMIT. The two phases are discussed below:

First Phase: The coordinator sends a PREPARE message to participant 2. If participant 2 is not willing to COMMIT, then it sends a VOTE ABORT (VA) to participant 3 and the transaction is aborted at this point. If participant 2, on the other hand, is willing to commit, it sends a VOTE COMMIT (VC) to participant 3 and enters a READY state. In turn, participant 3 sends its vote till node N is reached and issues its vote.

Second Phase: Node N issues either a GLOBAL ABORT (GA) or a GLOBAL COMMIT (GC) and sends it

to node N-1. Subsequently, node N-1 will enter an ABORT or COMMIT state. In turn, node N-1 will send the GA or GC to node N-2, until the final vote to commit or abort reaches the coordinator, node .

E. The Distributed Two-Phase Commit Protocol

In the distributed 2PC, all the nodes communicate with each other. According to this protocol, as Figure 5 shows, the second phase is not needed as in other 2PC methods. Moreover, each node must have a list of all the participating nodes in order to know that each node has sent in its vote. The distributed 2PC starts when the coordinator sends a PREPARE message to all the participating nodes. When each participant gets the PREPARE message, it sends its vote to all the other participants. As such, each node maintains a complete list of the participants in every transaction. Each participant has to wait and receive the vote from all other participants. When a node receives all the votes from all the participants, it can decide directly on COMMIT or abort. There is no need to start the second phase, since the coordinator does not have to consolidate all the votes in order to arrive at the final decision.

VI. HOMOGENOUS DATABASE MANAGEMENT SYSTEM: THE TWO-PHASE COMMIT

A homogenous distributed database system is a network of two or more homogenous Databases that reside on one or more machines. A distributed system that connects four databases. An application can simultaneously access or modify the data in several databases in a single distributed environment.

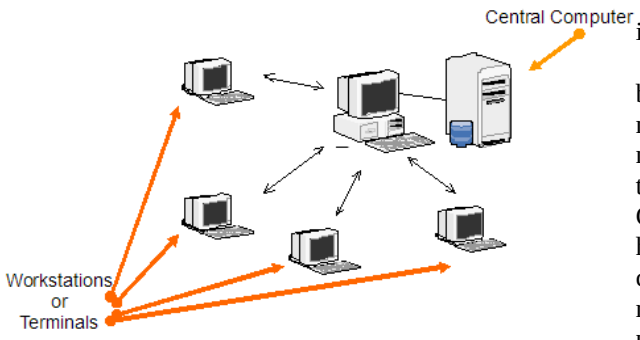

Fig.1.Homogenous Database Management System

For a client application, the location and platform of the databases are transparent. You can also create synonyms for remote objects in the distributed system so that users can access them with the same syntax as local objects. For example, if you are connected to database mfg but want to access data on database headquarters, creating a synonym on manufacturing for the remote dept table enables you to issue this query[4] ,he homogenous database is a distributed database management system, which employs the two-phase commit to achieve and maintain data reliability. The following sections explain homogenous's two-phase implementation procedures.

F. The Branch Tree

In each transaction, Oracle constructs a branch tree for the participating nodes. The session tree describes the relations between the nodes participating in any given transaction. Each node plays one or more of the following roles[10]:

1. Client(C): A client is a node that references data from another node.

2. Database Server(DS): A server is a node that is being referenced by another node because it has needed data. A database server is a server that supports a local database.

3. Global Coordinator(GC): The global coordinator is the node that initiated the transaction, and thus, is the root of the branch tree. The operations performed by the global coordinator are as follows:

• In its role as a global coordinator and the root of the branch tree, all the SQL statements, procedure calls, etc., are sent to the referenced nodes by the global coordinator. Instructs all the nodes, except the COMMIT point site, to PREPARE

• If all sites PREPARE successfully, then the global coordinator instructs the COMMIT point site to initiate the commit phase

• If one or more of the nodes send an abort message, then the global coordinator instructs all nodes to perform a rollback.

4.Local Coordinator: A local coordinator is a node that must reference data on another node in order to complete its part. The local coordinator carries out the following functions (Oracle8):

• Receiving and relaying status information among the local nodes

• Passing queries to those nodes

• Receiving queries from those nodes and passing them on to other nodes

• Returning the results of the queries to the nodes that initiated them.

5. Commit Point Site: Before a COMMIT point site can be designated, the COMMIT point strength of each node must be determined. The COMMIT point strength of each node of the distributed database system is defined when the initial connection is made between the nodes. The COMMIT point site has to be a reliable node because it has to take care of all the messages. When the global coordinator initiates a transaction, it checks the direct references to see which one is going to act as a COMMIT point site. The COMMIT point site cannot be a read-only site. If multiple nodes have the same COMMIT point strength, then the global coordinator selects one of them. In case of a rollback, the PREPARE and COMMIT phases are not needed and thus a COMMIT point site is not selected. A transaction is considered to be committed once the COMMIT point site commits locally.

G. Two-Phase Commit and the Homogenous Database Implementation

The transaction manager of the homogenous Oracle8 database necessitates that the decision on what to do with a transaction to be unanimous by all nodes. This requires all concerned nodes to make one of two decisions: commit and complete the transaction, or abort and rollback the transaction. The Oracle engine automatically takes care of the commit [18].

or rollback of all transactions, thus, maintaining the integrity of the database. The following will describe the two phases of the transaction manager.

1. PREPARE Phase (PP): The PP starts when a node, the initiator, asks all participants, except the commit point site, to PREPARE. In the PP, the requested nodes have to record enough information to enable them either to commit or abort the transaction. The node, after replying to the requestor that it has PREPARED, cannot unilaterally perform a COMMIT or abort. Moreover, the data that is tied with the COMMIT or abort is not available for other transactions.

Each node may reply with one of three responses to the initiator. These responses are defined below:

a. Prepared: the data has already been modified and that the node is ready to COMMIT. All resources affected by the transaction are locked.

b. Read-only: the data on the node has not been modified. With this reply, the node does not PREPARE and does not participate in the second phase.

c. Abort: the data on the node could not be modified and thus the node frees any locked resources for this transaction and sends an abort message to the node that referenced it.

2. COMMIT Phase (CP): Before the CP begins, all the referenced nodes need to have successfully PREPARED. The COMMIT phase begins by the global coordinator sending a message to all the nodes instructing them to COMMIT. Thus, the databases across all nodes are consistent.

H. Failure of the Two-Phase Commit(2PC)

A major problem with the two-phase commit occurs when one of the nodes participating in a distributed transaction fails while the transaction is in the PREPARED state. When the failure is for a prolonged period of time, then the data locked on all the other nodes won't be available for other transactions. This will cause a lot of transactions to rollback due to deadlocks. Oracle DBMS, in a new version, introduced an advanced queuing technique to deal with the problem of deadlock. The authors hope to address this technique in another paper in the near future.

VII. AN EXAMPLE OF A DISTRIBUTED DATABASE SYSTEM

Fig. 2 illustrates the steps homogenous distributed database performs in order to PREPARE, Select the COMMIT Point Site, and COMMIT. The example in the figure depicts a company that has several branches located in different cities numbered A to G. Each site has to have access to most of the data in the company in order to check on the status of purchase orders, material acquisition, and several other issues. Since new projects are awarded and older projects are completed, project sites tend to change locations. Also, depending on the size and duration of a project, different COMMIT point strength can be assigned and thus, in the same area, different COMMIT point sites can be chosen, for a given location, over a period of time. In this example, City E is the head office and thus posses the highest COMMIT point strength. The other sites are assigned the COMMIT point strength based on the rupee volume of the project. Higher monetary value for a project requires more resource allocation, and as such, will lead to more transactions executed against the data for that project. Since the amount of data involved is large, each site will have the portion of the database pertaining to its operations replicated and stored on a local server. Any transaction will at least affect the database at the head office and one of the sites. If, for example, a material rate, description of an item, accomplished progress, or purchase order is entered, a transaction is initiated that will affect the database at the head office and the database at the concerned site.

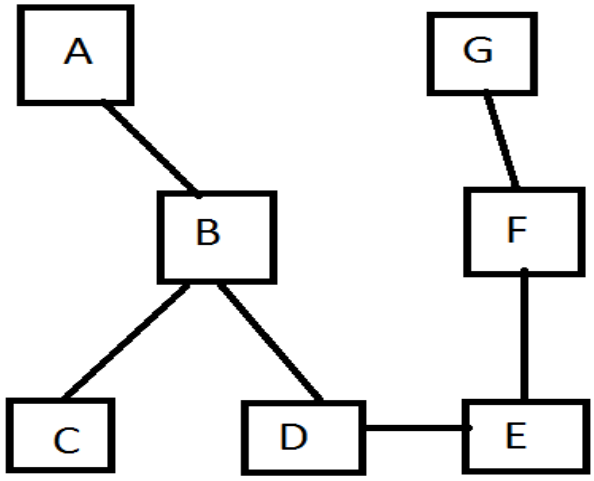

Fig.2-Distributed Database System over several node

Additional modifications, such as those involving employee transfer or equipment transfer from one site to another, will affect two or more sites. The following discussion explains the steps that entail in processing a distributed transaction:

An employee is to be transferred from City F to City B. The transaction is initiated by City E by a personnel employee. The affected sites need to participate in the transaction. The processes that transfer one employee from one site to another should be grouped under one transaction so that either all or none of the processes are carried out. An explanation of these steps follows:

1. Since City E is initiating the transaction, it becomes the root of the session tree, i.e. the global coordinator. Since City 1 updates data in City F and City B, it becomes a client. Since City E updates data on City G and City B, the two nodes become database servers.

2. When the application issues the COMMIT statement, the two-phase commit is started.

3. The global coordinator determines the COMMIT point site.

4. The global coordinator issues the PREPARE statement to all nodes except the COMMIT point site. If any of the nodes cannot PREPARE, the transaction is aborted; otherwise, a PREPARED message is sent to the node that referenced it.

5. The global coordinator instructs the COMMIT point site to COMMIT. The COMMIT point site commits the transaction locally and records the transaction in its local redo log.

6. The COMMIT point site informs the global coordinator that it has committed and the global coordinator informs the other nodes by sending the COMMIT message.

7. When all the transactions have committed, the global coordinator informs the COMMIT point site to "forget" about the transaction. The COMMIT point site, after "forgetting" about the transaction, informs the global coordinator, and the global coordinator, in turn, "forgets" about the transaction.

VIII. CONCLUSIONS

Scheduled the present time Transaction management is an fully grown thought in distributed data base management systems (DDBMS) for research area. Our Homogenous Distributed Database Systems based replication proposal is able to inherit and reduces the communication traffic the best characteristics of the Database Systems. However, Oracle was the first commercial DBMS to implement a method of transaction management: the two-phase commit. Though it was very difficult to obtain in order on homogenous DBMS implementation of this method were able to pull together sufficient in sequence to put in writing homogenous transaction for the database system. Many associations do not implement distributed databases because of its difficulty. They simply resort to centralized databases. However, with global organizations and multi-tier network architectures, distributed implementation becomes a necessity. It is hoped that this paper to will assist organization in the implementation of distributed databases when installing homogenous DBMS, or give confidence organizations to journey from centralized to distributed DBMS. Organisations could also contribute to this process by having graduates with the knowledge of homogeneous DBMS capability. With DBMS making so much effort on incorporating this and other advanced features in its database software, academicians should also play a major role in exposing beneficiary to these superior element transaction management.

REFERENCES

[1] R. Abbott and H. Garcia-Molin a, "Scheduling 'Real-Time Transactions: a Performance Evahration", F&c. of 14th VLDB Conj., August 1988.

[2] M, Valduriez P,1991, Principles of Distributed Database Systems, Prentice-Hall.P. Bernstein, V. Hadzilacos and N. Goodman, .

[3] E. Cooper, "Analysis of Distributed Commit Protocols", Proc. of ACM Sigmod Conj., June 1982.

[4] Ghazi Alkhatib, Transaction Management in Distributed Database: the Case of Oracle's Two-Phase Commit, Vol. 13(2).

[5] Mohan, C.; Lindsay, B.; and Obermarck, R. [1986],"Transaction Management in the R* Distributed Database Management System." ACM Transactions on Database Systems, Vol. 11, No. 4, December1986, 379-395.

[6] Ozsu, Tamer M., and Valduriez, Patrick [1991], Principles of Distributed Database Systems,Prentice H.

[7] G. Coulouris, J. Dollimore, T. Kindberg: Disributed Systems, Concepts and Design, Addison–Wesley, 1994.

[8] S. Ceri, M.A.W. Houtsma, A.M. Keller, P. Samarati: A Classification of Update Methods for Replicated Databases, via Internet, May 5, 1994.

[9] D. Agrawal, A.El. Abbadi: The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. in Proc. of VLDB Conf. pp 243-254, 1990.

[10] Ramamritham,Son S. H, and DiPippo L,2004, Real-Time Databases and Data Services, Real-Time Systems J., vol. 28, 179-216.

[11] Robert A and Garcia-Molina H,1992, Scheduling Real-Time Transactions, ACM Trans. on Database Systems, 17(3).

[12] Jayant. H, Carey M, Livney,1992, "Data Access Scheduling in Firm Real time Database Systems", Real Time systems Journal, 4 .

[13] Jayanta Singh and S.C Mehrotra et all, 2010,"Management of missed transaction in a distributed system through simulation", Proc. Of IEEE .

[14] Udai Shanker, "Some Performance Issues In Distributed Real Time Database System", PhD Thesis, December,2005 .

[15] Jayanta Singh and S.C Mehrotra, 2006,"Performance analysis of Real Time Distributed Database System through simulation" 15th IASTED International Conf. on APPLIED SIMULATION & MODELLING, Greece.

[16] Jayanta Singh and S.C Mehrotra,2009 "A study on transaction scheduling in a real-time distributed system",EUROSIS"s Annual Industrial Simulation Conference, UK.

[17] Jayant H. 1991, "Transaction Scheduling in Firm Real-Time Database Systems", Ph.D. Thesis, Computer Science Dept. Univ. of Wisconsin, Madison.

[18] Oracle8 Server Distributed Database Systems, Oracle, 3-1 – 3-35.

