ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 4, Issue 2, pp: (328-337), Month: April - June 2016, Available at: www.researchpublish.com

PARTIALLY DELAYED REPLICATION IN DATABASE SYSTEMS

Christos Papanastasiou

Technological Educational Institution Stereas Elladas, Department of Electrical Engineering, 35100 Lamia, GREECE

Abstract: **Distributing and replicating data are techniques that are used to improve performance on information systems. Most applications such as Web-based services and electronic commerce use the method of data replication. In this paper I will present a partially delayed replication database management system that can be applied on computer applications. Files used in a computer application can be separated as master files and transaction files and be treated in different way as to achieve a better performance of the system.**
The management system for a distributed database system that I am suggesting on this paper I believe it can provide very good feasibility and applicability of DDBMSs for most of business applications.

Keywords: **delayed replication, master files, transaction files, distributed database.**

1. INTRODUCTION

Distributed databases are more complex than centralized databases but if we find the best way to implement them on business applications we will provide great benefits to the companies that they will support.

Centralized systems are basically easier in design and management than distributed systems. In some cases it is necessary to distribute data over a computer network or even move some data closer to the end user to achieve better performance and scalability of the system.

In a distributed database system there will be the need to replicate some data in order to reduce network traffic and have a better response time of the system.

In this paper I will present a distributed database management system that can be used in a distributed database system based on the replication technique dependent on volume transaction.

1.1. Advantages of a Distributed Data Base System

Most companies have their branches located in different places. In this case a Distributed Data Base System is the one we will use and partially a centralized Data Base System. [6] What we can achieve in this case is for each location to have its local data and the ability to get needed data from other locations via a communication network. Moreover, if one of the servers in one node fails, the distributed database system will still be accessible.

If any data is required from a site which has been in a failure, these data may be retrieved from other locations containing the replicated data.

The expansion of the distributed system is easy, since adding a new location doesn't affect the rest nodes.

1.2. Disadvantages of a Distributed Data Base System

A Distributed Data Base System has several disadvantages. A distributed system presents more complexity and is more expensive than a centralized system. This is true because the hardware and software involved need to maintain a reliable and an efficient system. All replication and data retrieval from all nodes should be transparent to the user. Another main disadvantage of distributed database systems is the issue of security. Handling security across several locations is more complicated.

1.3. Failures in a Distributed Data Base System

In a distributed database system several failures may occur.

1.3.1. Communication Failures

Communication failures are failures in the communication system between two or more nodes. In this case several nodes or a group of nodes will be operating independently and some transactions sent from one node to another will be lost.

Reliability protocols can utilize a timeout mechanism, in order to detect undelivered transactions. A transaction is undelivered if the sender doesn't receive an acknowledgment.

1.3.2. Node Failures

Node failures usually happen because of software or hardware failures. These failures result in the loss of the main memory contents.

1.3.3. Transaction Failures

When a transaction fails, it aborts. In this case, the database has to be restored to the state it was before the transaction started. Transactions may fail for several reasons. Some failures may be due to deadlock situations or concurrency control algorithms.

1.3.4. Media Failures

This kind of failures refers 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.

2. FRAGMENTATION

The fragmentation technique for distributed databases involves splitting the centralized database into portions and moving them to different locations. This distribution is accomplished by horizontal and/or vertical partitioning. No data is stored redundantly with the exception of primary keys in the case of vertical fragmentation. [5] Using the relational model, horizontal fragmentation is accomplished by separating rows and vertical fragmentation is accomplished by separating columns. Data is normally fragmented according to the section of the organization which uses or modifies the data most frequently. For example, a company may use a department code field to determine which department is responsible for each record and where the data should physically reside in a horizontally fragmented system.

Key principles of the fragmented distribution model are that only one copy of the data exists in the database, and that ownership and ability to update the database are shared. This model is similar to the centralized model in that the data is always consistent and current. The only redundancies exist with primary key fields when using vertical fragmentation. Fragmented database systems are more complex than centralized systems, but simpler than replicated systems. The issue of a single point of failure is reduced, but not eliminated. Network usage is generally lower in fragmented systems than in centralized systems.

3. DATA REPLICATION

In distributed database systems, replication is used to ensure access to remote data by providing their local copies. It might also accomplish better response times of access to data in case we have a slow network connection. A database management system supporting replication must guarantee that changes to one of the replicas will be propagated to all other replicas in order that all of them be consistent with each other. [1] This can be achieved synchronously (when a transaction modifies one of the replicas, modifications have to be propagated to all other replicas) or asynchronously (a transaction that will modify one of the replicas is done and the database system guarantees that modifications to the rest of the replicas will be propagated to other replicas at a later time).

Synchronous replication is difficult to be used in practice because it will not allow modifying a replicated object if any of the replicas in the system is not accessible. Asynchronous replication, if there is no access to one of the replicas might lead to temporary inconsistency.

The goal of the replication service is to maintain all replicas in a state of consistency with the master within a timeliness standard required for the replicas to be useful to the application. This is accomplished through complete or incremental refresh of the master to slaves, or through delta propagation of events to slaves. These are also commonly referred to as table-based or transaction-based replication techniques respectively. The choice of which method to use is one the designer makes, taking into account such factors as the data consistency requirements of the system, the network's capabilities, and the anticipated volume of updates to the database. In practice the designer may be constrained, however, since DDBMSs typically employ only one of these two techniques.

We could also design our system in a way to be able to use master/slave model to change from one node to another node containing a copy of all data files (for every group). This will allow some flexibility of the system to respond to changing load requirements. For example, the system could move the master designation to a node preparing to perform a large number of batch updates or to an alternate node in the event the master experiences a failure. However, the basic principle of the master/slave model is always in effect. Only one node can be the master at any one time [10].

The update anywhere model, as its name suggests, allows updates to be performed on any copy of the data in the distributed database - there is no designated master for any data element. Read and update operations may be performed at any location. Updates are propagated to all copies within the database. The update-anywhere model complies with C.J. Date's second rule for distributed databases, "No reliance on a single site".

Replication in the update-anywhere model can be performed either synchronously or asynchronously. Synchronous replication requires the use of the two-phase commit protocol, and ensures all atomicity, consistency, isolation, and durability (ACID) properties of transactions. Asynchronous replication may be used in some cases, with the loss of isolation between transactions operating on identical data at different sites at the same time. The choice between the two methods is one the system designer must make based upon the requirements of the system, the capabilities of the network, and the anticipated volume of updates to the database. The costs of implementing synchronous replication are higher relative to asynchronous replication due to the increased requirements on the network and the database system hardware. [1]

Asynchronous replication under the update-anywhere model can cause a great deal of complexity when data consistency must be maintained. No design using this technique can completely prevent data conflicts from occurring. Conflicts are only detected after they occur, and must be repaired using manual operations or system-generated compensating transactions. Because the original transactions are sometimes undone, this violates the concept of durability of transactions. Undoing transactions can have a ripple effect as undoing one transaction creates referential integrity or other problems for later transactions. Operating update-anywhere databases with asynchronous replication can be compared to running a centralized database with all locking mechanisms off - performance is greatly increased, but transactional integrity is often lost [1].

4. CONCURRENCY CONTROL

In replicated databases, concurrency control is implemented only in systems using synchronous replication under the update-anywhere model. Distributed databases that use fragmentation techniques or master/slave replication do not require concurrency control except of what is required in traditional centralized database systems. Distributed databases using asynchronous replication don’t use distributed concurrency control. They rely on conflict detection and compensating transactions. [7]

The two-phase commit protocol is a required component of distributed databases that use synchronous replication. The two-phase commit protocol consists of the coordinating node issuing a transaction to all affected nodes, waiting for each node to acknowledge that it is prepared to commit, and then issuing a commit order to the affected nodes.[13] If not all affected nodes acknowledge that they are prepared to commit after the first phase, then the coordinating node issues a rollback instruction to all nodes. With the two-phase commit protocol, there is a short period of vulnerability between commit times at different nodes during the second phase when data consistency could possibly be lost due to a node or network failure.

5. FAILURE RECOVERY

Replicated database systems can provide a level of fault-tolerance failure recovery beyond of what can be achieved through traditional means as it is the use of redundant array of inexpensive disks (RAID). By replicating the database so that it is on two separate computers in different physical locations on the network, the probability that failures will cause a loss of data is significantly reduced. There are two options available for implementing failure recovery: warm standby and hot standby.

Warm standby uses asynchronous replication to maintain the standby server in a state nearly consistent with that of the primary server. [5] Due to the lag between transactions being committed on the primary server and replication to the standby server, a small number of transactions are normally lost during a primary server failure and switchover to the standby server.

Hot standby uses synchronous replication to maintain the standby server in a state always consistent with the primary server. From an availability perspective this is the preferred solution, but the higher costs and potential lower performance of synchronous replication databases cause many companies to select a warm standby solution. Buretta recommends a combination of local hot standby, normally RAID, and offsite warm standby server [1]

6. PROBLEMS CONNECTED WITH DATA DISTRIBUTION AND REPLICATION

Distribution and replication of data in a business application system may lead to a complicated implementation of the system and in case of failure to some dangerous and unpredictable behavior. The first problem is to assure that all nodes of the network system use the same catalog data (customers, suppliers, storehouses, e.t.c.) This can be done by using one master copy and replication of this to every server in each location. The first solution leads to remote updates in case of modifying a file from many sites of the business network system. The second requires more expensive versions of database management system.

7. SIMILAR WORK

It is necessary to study performance improvement technics in distributed database system processing. Most object database systems are using data migration method. There are some studies dealing with overload control for replicated database systems. There are no studies on performance improvement by means of partially delayed replication of specific data files. I will explain in this paper a useful performance improvement by partially separating files on multiple servers and multiple users in groups in a distributed database system.

8. SUGGESTED MANAGEMENT SYSTEM WITH PARTIALLY DELAYED REPLICATION

In most computer applications, we usually meet two kinds of files, master files (customer, supplier, storehouse, e.t.c.) and transaction files (customer transaction file, supplier's transaction file, storehouse transaction file, e.t.c.)

Master files are usually small in size and it is easy to copy or update each of them to every server (replicas). Transaction files are larger in size files and if we try to copy them in every server, it will take valuable time and the response of our network system will not be efficient.

We will assume that we will be able to provide connectivity between different database servers which do not add another database platform to existing architectures.

Our distributed system is composed of:

- A master server on which each global transaction is submitted, and
- Local servers that have replicas of parts of the distributed database.

All transactions are mandatory, which means that they are executed within their local server.

Each transaction is sent to the master server and from the master server to only those local servers who have replicas of parts of the distributed database, whenever there is no traffic from the local server to the master server.

All servers including tha master server are partially connected which means that any user that is connected to his local server can access another local server directly or indirectly.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 4, Issue 2, pp: (328-337), Month: April - June 2016, Available at: www.researchpublish.com

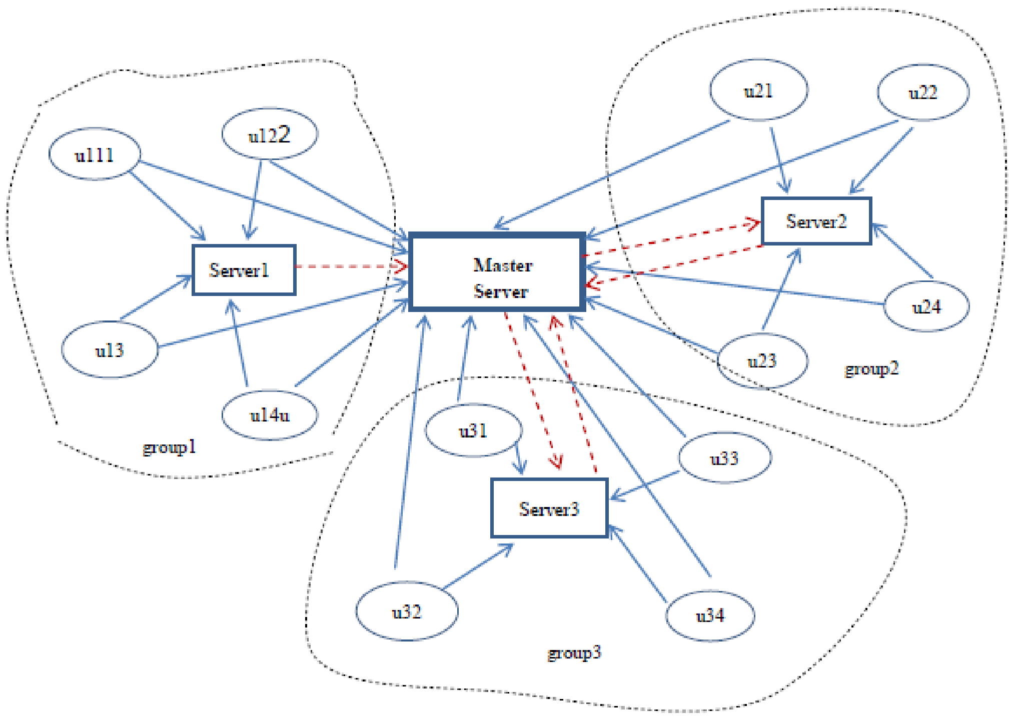

Fig. 1: Partially delayed update of transaction files

In our database management system, many servers all over the world will exist. We can group the servers naming each group as group1, group2, group3, e.t.c. For every group, we will have one server (with it's backup) and each server of each group will be connected with the master server. On the servers of each group we will keep all master files and replicas of transaction files. On the master server (and it's backup server) we will also keep all files (master files and transaction files). If we need to submit a transaction from a site of some group, this transaction will be created on a log transaction file. With this transaction, the master file (holding main and *necessary information* like the product's name, details concerning the specific product, *the available quantity*, e.t.c.) that belongs to the group will be updated and the master file on the master server will be also updated (synchronously). This way all required information will be updated on all servers and on the master server. The transaction file will be updated on the group's server and at a delayed time (when there is no traffic on the network – dashed lines in fig. 1) it will be updated on the master file and from the master file the system will update the transaction files on local servers where there is a replica (asynchronously). In other words, propagated updates of every transaction file will happen from each site of our centralized system in every group where replicas exist through the master server, whenever there is no traffic on the network for that group.

Master files on every group will be updated synchronously. [2] All replication will take place asynchronously to the updating of the master files. This means that all of the data replicas of master files are made consistent with the master server.

You can see on fig.1 by continued lines the update of each transaction from each user in every group, in group's server and with dashed lines the update of transaction files at a delayed time on every other group's server and on the master server.

In fig. 2 we can see the normal update of each transaction from each user in every group with only continued lines.

In my example, I suppose that in every group there are 4 users and all together there are 3 groups, group1, group2 and group3, then making the following assumptions for both cases:

For group1: The time needed for each transaction when executed to update the transaction file on server1, is 2 sec and the time needed for each transaction when executed to update the transaction file on master server is 2.5 sec.

For group2: The time needed for each transaction when executed to update the transaction file on server2, is 2 sec and the time needed for each transaction when executed to update the transaction file on master server is 2.6 sec.

For group1: The time needed for each transaction when executed to update the transaction file on server3, is 2 sec and the time needed for each transaction when executed to update the transaction file on master server is 2.7 sec.

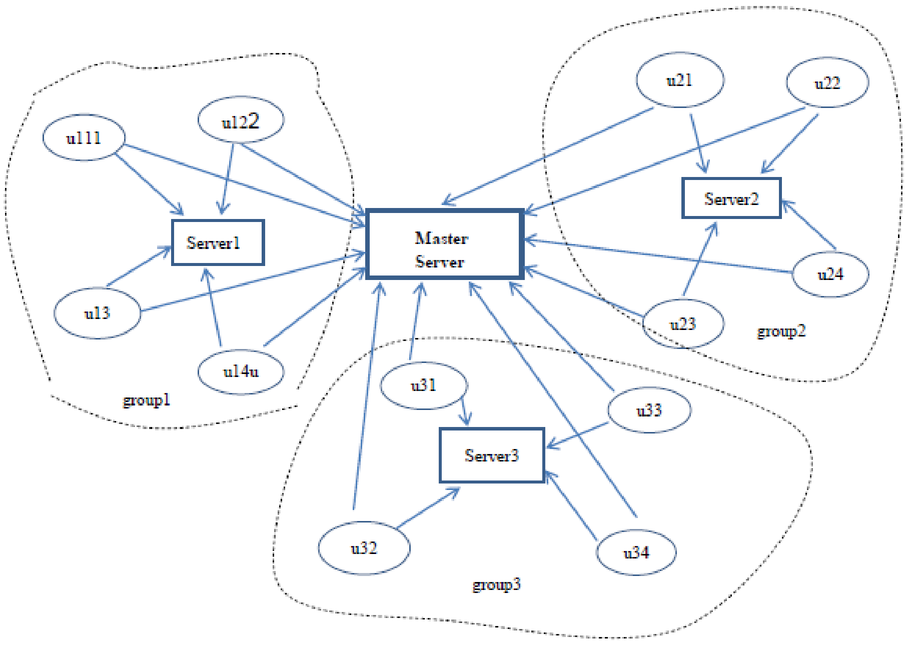

Fig. 2: Normal update of transaction files

We can calculate the total execution time of updating the transaction files on each server and on master server without the partially delayed replication, as we see it in fig. 2

For group1: The time will be : 4 (users) * 2.0 = 8.0 sec for updating the transaction files on server1
4 (users) * 2.5 = 10.0 sec for updating the transaction files on master server

For group2: The time will be : 4 (users) * 2.0 = 8.0 sec for updating the transaction files on server2
4 (users) * 2.6 = 10.4 sec for updating the transaction files on master server

For group3: The time will be : 4 (users) * 2.0 = 8.0 sec for updating the transaction files on server3
4 (users) * 2.7 = 10.8 sec for updating the transaction files on master server

The total time required for the execution of all transactions is: 55.20 sec

We will calculate now the total execution time of updating the transaction files on each server and on master server with the partially delayed replication, as we see it in fig. 1

For group1: The time will be: 4 (users) * 2.0 = 8.0 sec for updating the transaction files on server1
For group2: The time will be: 4 (users) * 2.0 = 8.0 sec for updating the transaction files on server2
For group3: The time will be: 4 (users) * 2.0 = 8.0 sec for updating the transaction files on server3

The total time required for the execution of all updates on transaction files is: 24.0 sec

We don't take into account here the required time for the execution of transactions to update the transaction files on every other server and on the master server, because the execution of these transactions will happen in delayed time, when there is no traffic on the network.

8.1. Suggested protocol

The protocol to manage execution of transactions concerning transaction files, is based on the implementation strategy for centralized concurrency control algorithms.

During the execution of each transaction on the local server, the system acquires locks to access data objects. When the execution of the transaction is completed, then this transaction is placed in a transaction queue. Whenever there is no traffic between the local server and the master server, the system sends a request to the master server to execute the transactions that are held into the transaction queue.

When the master server receive this request, then it executes the transaction locally and right after it's execution it looks to find the local servers who have replicas of parts of the distributed database and sends a request to execute the transaction to each of those servers.

When the local server commits the received request from the master server, it requests a write lock to update the data object in its local work-area.

When the transaction completes its execution successfully, the local server sends a SUCCESSFUL message to the master server. If for some reason the transaction fails to execute, the local server will send an ABORTED message to the master server.

When all transactions from each local server are executed successfully on the master server and on each local server who has replicas of parts of the distributed database, the master server will send an UPDATE-COMPLETED to every local server with replicas. This means that whenever a user requests a transaction analysis for any data object, if the local server is informed by the master server with the UPDATE-COMPLETED message, the transaction analysis of this data object will be submitted to the user. If the local server hasn't received an UPDATE-COMPLETED message from the master server, this means that the transaction file is not updated and consequently the request for providing the transaction analysis to the user will not take place.

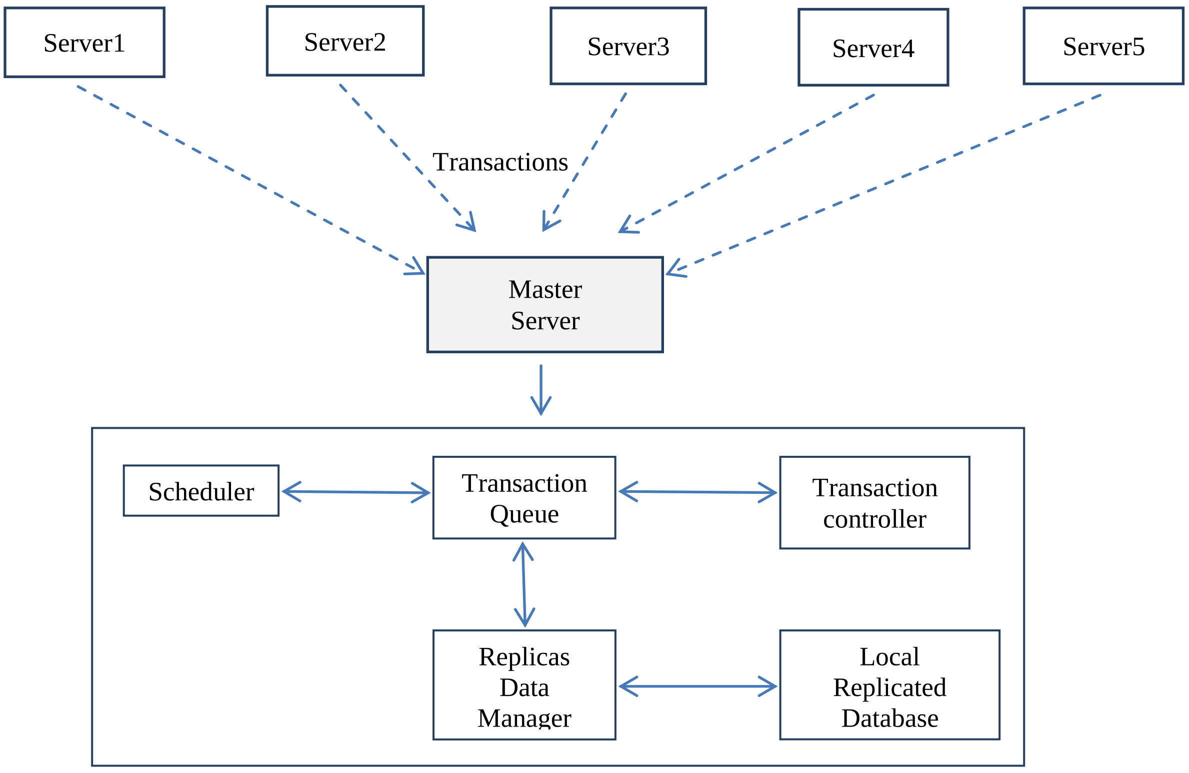

Fig. 3: The different modules of the system

8.2. Algorithm

In the algorithm I use some variables and data structures as I describe them below.

- T_i denotes a global transaction submitted to the master server
- TransactionReady is the structure of each element of transaction queue

Within the MASTER SERVER:

```
When a transaction $T_i$ is submitted to the master server
DO
   $\forall i \in \{1,k\}$:
   BEGIN
       UPDATE($T_i$);execute transaction $T_i$
       Send REQUEST($T_i$) ;send REQUEST messages to all replica servers
   END
END DO
```

Within Any Local Server:

```
When a local server receives a REQUEST($T_i$) from the master server
Index=0;
DO
     Index=index+1;
     LOCKREQUEST($T_i$); request for a lock to execute this transaction locally
    Do until lock is provided
          EXECUTE($T_i$)
    END DO
    TRANSACTION_QUEUE(Index)=$T_i$
    Index=0;
    If processor laxity>0 THEN
        Index=Index+1;
        Ti=TRANSACTION_QUEUE(index);
        Send REQUEST(Ti);  send a request to the master server to execute $T_i$
        WHILE ($s_r$ is a replica server)
            DO
                Send REQUEST($T_i$); send a request to replica server sr for updating
                UPDATE($T_i$);
                If WRITE_SECCESSFUL($T_i$) THEN
                      Send SUCCESSFUL($T_i$); send successful message to the master server
                ELSE
                      Send ABORTED($T_i$); send message to the master server that $T_i$ was aborted
                END IF
          END DO
   END IF
END DO
END
```

Our conclusion is that when we apply the suggested partially delayed replication method, the total execution time is much less than the total execution time that was needed when we applied the replication method. We also observe that this difference gets greater when the number of users and groups increases. This difference will increase considerably when there is a large number of groups, giving us the opportunity to seize in the best way using the system when there is no network traffic (having no time cost) to update the transaction files on every other server except the one we are working and on the master server and always keep updated all files on every replica server that belongs to each different group and to master server.

We summarize the above results in the following two tables.

Table 1: Execution time applying normal replication method

Transactions	**Execution time**
Group1 updating server1	8.0 sec
Group1 updating the rest group servers and master server	10.0 sec
Group2 updating server2	8.0 sec
Group2 updating the rest group servers and master server	10.4 sec
Group3 updating server3	8.0 sec
Group3 updating the rest group servers and master server	10.8 sec

Table 2: Execution time applying partially delayed replication method

Transactions	Execution time
Group1 updating server1	8.0 sec
Group1 updating the rest group servers and master server	0.0 sec
Group2 updating server2	8.0 sec
Group2 updating the rest group servers and master server	0.0 sec
Group3 updating server3	8.0 sec
Group3 updating the rest group servers and master server	0.0 sec

8.3. Replication method suggested

Master files: We have replicas of all master files in every group’s server.

For our master files the “Primary-copy” approach or “master-slave” approach will be used. The basic idea on this approach of replicating files is that for each data item or file a primary copy exists and all other replicas are secondary copies. The updates can only be done by the primary copy which is the owner of the file. If a write request is sent to a secondary copy, the request is passed on to the primary copy which does the updates and propagates the changes to all secondary copies.

Transaction files: All transaction files reside on each group’s server with replicas and on the master server.

8.4. Data consistency

For all master files on every group updates are possible on replicas and the degree of data consistency is decreased. In this case, consistency also depends on the frequency of updates and the data items that will be covered by the update and by the expected response time. I have to mention here that the highest degree of data consistency that we can get on replicated files can only be established by having fully synchronous replicas which the suggested database management system does not support. Using the “primary-copy” approach as I mentioned above, we get a high degree of data consistency and this way write performance is improved.

8.5. Concurrency control

In our database management system we will use conflict detection and compensating transactions.

8.6. Failure recovery

Warm standby will be used to maintain any standby server consistent with the master server.

8.7. Security

The application program will be responsible for governing user access to the data.

9. CONCLUSIONS

Distributed database systems are suitable for large companies that consist of many branches located in different places all over the world. Replication of some data (all master files in our designed system) is necessary to provide the necessary information and to guarantee access to data from remote servers and improve performance.

Replication is necessary in case of data generated in a branch and only read by other branches. Asynchronous replication (for all transaction files located on the servers of each group and their mirrors) is considering the best solution because it easily can hide a server or network failure.

Basically this distributed database management system is based on centralized system for every group of the network and a distributed database system as a whole system.

The suggested partially delayed replication is an efficient technique in the distributed databases, providing the following:

- The manageability of our distributed data base system is proportional to the number of files that are defined and maintained.

- When a node fails, all data remains accessible to the other nodes since master files exist to each node and transaction files exist to every local server. After detecting node failure the system is automatically reconfigured. When the node is on again, the system will copy all master files from the closer node. All transactions that were made before the node failure are not lost because they are written in a log file.

REFERENCES

[1] Buretta, Marie. (1997) Data Replication, New York: John Wiley & Sons

[2] Burleson, Donald K. (1994) Managing Distributed Databases. New York: John Wiley & Sons

[3] M. T. Φzsu, P. Valduriez, Principles of Distributed Database Systems, Prentice-Hall, 1991.

[4] Oracle8 Server Documentation, Oracle Corporation, 1997.

[5] D. Bell and J. Grimson. Distributed Database Systems, Reading, MA: Addison-Wesley, 1993.

[6] M.T. ¨ Ozsu and P. Valduriez. Principles of Distributed Database Systems, Englewood Cliffs, NJ: Prentice-Hall, 1991.

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

[8] Anraham Silberschartz, Henry F. Korth (2006) Database SystemConcepts, Mc Graw Hill International edition.

[9] Paolo Atzeni, Stefano Ceri, Stefano Paraboschi amd Riccardo Torlone (1999), Database Systems Concepts, Languages and Architectures, Mc Graw Hill.

[10] Ramakrishnan, Gehrke Database Management Systems, Mc Graw Hill International edition (2003)

[11] Bhargava Bharat: Concurrency Control in Database systems: IEEE Transactions on knowledge and data engineering, VOL.11, No1,1999 (IEEE)

[12] Ali R. Abdou, Hany M. Harb: Two phase locking concurrency control in distributed databases with N-Tier architecture; IEEE 2004.

[13] Bhargava Bharat: Concurrency Control in Database systems: IEEE Transactions on knowledge and data engineering, VOL.11, No1,1999 (IEEE)

[14] Y. Yoon, "Transaction Scheduling and Commit Processing for Real-Time Distributed Database Systems", Ph. D. Thesis, Korea Adv. Inst. of Science and Technology, May 1994.

[15] A. Bestavros, "Multi-version Speculative Concurrency Control with Delayed Commit", P m . of Zntl. Conf. on Computers and their Applications, March 1994.

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

[17] R. Gupta, J. Haritsa, K. Ramamritham and S. Seshadri, "Commit Processing in Distributed RTDBS", TR-96-01, DSL/SERC, Indian Institute of Science

[18] J. Haritsa, M. Carey, and M. Livny, "Data Access Scheduling in Firm Real-Time Database Systems", Real-Time Systems Journal, 4 (3), 1992.

[19] O. Ulusoy and G. Belford, "Real-Time Lock Based Concurrency Control in a Distributed Database System", Proc. Of 12th Zntl. Conf. on Distributed Computing Systems, 1992.

[20] A. Mohan, B. Lindsay and R. Obermarck, "Transaction Management in the R* Distributed Database Management System", ACM TODS, 11(4), 1986.

[21] R. McFadden, Jeffrey A. Hoffer, "Modern Database Management", fourth edition, The Benjamin/Cummings publishing company Inc, 1991

[22] Son, S. and Zhang, F. (1995). Real-time replication control for distributed database systems: "Algorithms and their

[23] Performances". In the 4th International Conference on Database Systems for Advanced Applications, Singapore.

[24] Ulusoy, O. (1994). "Processing real-time transactions in a replicated database system. In Distributed and Parallel

[25] Databases", volume 2, pages 405–436.

[26] Son, S. (1987). "Using replication for high performance database support in distributed real-time systems". In the 8th IEEE Real-Time Systems Symposium, pages 79–86.