A Synchronous/Asynchronous Multi-Master Replication Method

by

Amer Mohammad Kheir Chahine

B.S., Computer Science, Lebanese American University, 2004

Thesis submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science

Division of Computer Science and Mathematics

LEBANESE AMERICAN UNIVERSITY

August 2007

LEBANESE AMERICAN UNIVERSITY

Thesis Approval Form

Student Name:	Amer Chahine	I.D.: 200100610
Thesis Title :	A Synchronous/Asynchronous Multi-Master Replication Method	
Program :	M.S. in Computer Science	
Division/Dept :	Computer Science and Mathematics	
School :	Arts and Sciences - Beirut	

Approved by:

[redacted]
Ramzi Haraty, Ph.D. (Advisor)
Associate Professor of Computer Science

[redacted]
Issam Kouatly, Ph.D.
Associate Professor of Management Information System

[redacted]
Samer Habre, Ph.D.
Associate Professor of Mathematics

Date: August 24, 2007

Plagiarism Policy Compliance Statement

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Amer Chahine

Signature: [redacted]

Date: August 24, 2007

I grant to the LEBANESE AMERCIAN UNIVERSITY the right to use this work, irrespective of any copyright, for the University's own purpose without cost to the University or its students and employees. I further agree that the University may reproduce and provide single copies of the work to the public for the cost of reproduction.

To my Parents Najwa & Mohammad Kheir

To my sisters Alya & Ola

To my brothers Eid & Abdullah

Acknowledgment

First and foremost I am grateful to my advisor Dr. Ramzi Haraty. You have been an exemplar instructor not only in academic terms, but also at the personal level. Thank you for your guidance and continuous support from the moment I joined LAU. I am deeply grateful for that.

A very special thanks to Dr. Samer Habre and Dr. Issam Kouatly for being on my thesis committee.

I extend my warm regards and thanks to Dr. May Hamdan. I would like to thank you for all your valuable advice and encouragement- you have always had the right word at the right time.

I would like to express my sincere gratitude to my parents who taught me diligence, patience and hard work. They gave me the motivation from which I studied steadily throughout my childhood and adolescence. It is my great pleasure to dedicate this dissertation to my parents.

Last but not least, I would like to reveal all the gratitude and appreciation to my friends for their constant encouragement and tireless support: Abbass Sharif, Abed Hoteit, Asaad Akoum, Basma Almasri, Christine Kehyayan, Fadi Naime, Hadi Almasri, Hani Itani, Kamlish Khoudary, Katia Hallak, Mohammad Badran, Rim Lakkis, Rita Maktaby, Rola Khoury, and Saeed Haidar.

Abstract

This thesis is centered around the field of database replication. In this thesis, we present synchronous/asynchronous multi-master replication architecture and a replica control algorithm, which guarantees and provides a flexible and transparent solution. This architecture holds the advantages of previously applied methods and techniques. System performance is increased by lowering the load or usage of network resources. High availability is guaranteed by keeping synchronized nodes online or active at all time. High reliability is maintained by ensuring that the available nodes are synchronized and up-to-date. Finally, flexibility is provided in handling new nodes without the need of taking the full system offline.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Information availability becomes critical nowadays mostly due to the data-intensive applications, which populate the Internet. These throughput requirements, drive research towards distributed and replicated architectures, providing solutions that enable high availability through network shared contents. Such systems require efficient storage capability as well as easy retrieval and outstanding processing capabilities. Combining networked storage and processing entities, enables data distribution and/or replication, reduces wasted storage capacity and backup inconveniences as well as increases data availability.

Computer systems are considered to be more distributed, widespread and complex; thus, they are becoming more fragile. In terms of consequences, failures are becoming more serious and frequent. Fault tolerance and security issues are specialized topics in computer systems. Lately, societies are becoming more dependent on computers. Computer systems became an essential need in people daily life. Therefore, lives are affected by bugs, crashes and other serious problems (Neumann, 2002).

A lot of work and research has been done on security and fault tolerance, even software sellers are considering stability and security as an official priority (Maney,

2002). Different techniques and approaches have been proposed promising stable, secure, and fault-tolerant systems. Approaches that offer fault-tolerance include specialized computer languages, software engineering, and mainly replication. In this thesis we concentrate on replication.

One of the natural ways for dealing with failures is replication; that is if a replica becomes unavailable or fails, another replica takes over. Similarly, most living organisms establish fault tolerance the same way; when a cell dies, other cells recover up and contribute in surviving the whole organism.

While replication is considered an optimal solution for fault tolerance, it is not easy to implement a consistent replicated system. A minimal error in a critical system that manages an industrial unit results with crucial consequences. For example, if a server crashes, a slightly wrong account in a banking system is unacceptable. To enforce strict consistency and stability in case of a system crashes, replication is considered a complicated problem.

Customer information, inventories, payrolls are other important data, which belong to a banking database application are stored and saved in a secured databases that represent the key element of the IT infrastructure of different companies. Therefore, it is a good idea to replicate such databases in order to increase fault tolerance. However, database replication is considered to be complex and complicated when strict consistency is enforced (Cap, 1990).

In this thesis we mainly discuss the recently proposed database replication techniques. Furthermore, we propose combined replication architecture and a replica control algorithm, which guarantees and provides a flexible and transparent solution. This

architecture holds the advantages of previously applied methods and techniques such as increasing the performance of the system, maintaining high reliability, and finally being able to handle additional replicas by maintaining the availability status of the whole system.

1.1 Background

Database replication is considered one of the oldest topics in the field of computer science. It has been in research for almost twenty years (Thomas, 1979), (Gifford, 1979) and (Stonebraker, 1979). Many papers had proposed correct techniques and solutions for database replication, yet these solutions performed badly when there is an increase in the number of databases (Gray et. al, 1996). This is because such replication protocols introduce a slight difference and changes in the database engine, which leads to high number of deadlocks and high synchronization costs. Migrating data and changing the architecture is not a simple work to do because a large amount of information is used every day and stored in databases.

Databases have become significant for enterprises; thus, high performance resembles an absolute need for replicated databases (Jajodia, 1999). Achieving high performance for replicated database is established by following the consistency rules. This has been proposed as the main avenue of research (Bernstein et. al, 1998) and commercial database systems (Oracle, 1998) and (Informix, 1998) are currently following this approach, which is also known as lazy replication. This replication technique has a major advantage that it follows the black box approach. However, this type of replication does

not solve the problem of consistent fault tolerant replication. Lately, many proposals were submitted alternative to lazy replication such as group communication based replication (Schiper and Raynal, 1996), (Alonso, 1997) and (Pedone et. al, 1998). These proposals follow the white box approach; they offer better performance by providing fault tolerant consistent database replication using high level primitives.

Transactions are considered the element units of databases. They are group of commands which are performed as a logical unit on the database. ACID rules define the processing of a transaction (Gray and Reuter, 1993). Atomicity checks if transactions are executed to completion, or not at all. Consistency checks whether transactions lead the database to a legal state. Isolation makes sure that executing more than one transaction in parallel does not affect any of the transactions. Finally, Durability checks that when transactions commit, their effects are stable forever even if some failure occur. ACID properties are essential for building a consistent fault tolerant database replication.

1.2 Database Replication

A database application system provides information storage facilities to end user applications. An advantage of a database system is that it guarantees data integrity when data is being accessed concurrently. A replicated database system is composed of many redundant copies of a database distributed across many sites. Each database, being called a replica, can work individually to accept client's requests. When a replica fails, the system remains fully available due to redundancy. When a replica fails, client's requests

are routed to other replicas. In case of catastrophic disaster such as fire, data will survive as long as one replica, probably in a different geographical site remains accessible.

With database replication, data can be replicated to remote sites so that clients in the remote sites can access the data just locally. The purpose of each replica is to handle client's requests; therefore, by adding replicas and increasing the work capacity of a system, the throughput is improved. And because replicas can be located closer to applications then the response times are shorter. Therefore, by database replication both availability and performance are improved.

The main challenge of database replication is to keep the data copies consistent in the presence of updates. That is any update has to be propagated properly to guarantee that the data remain consistent in the different replicas available.

1.3 Motivation

Communication technology has been under remarkable advance over the past years, which led to great developments in the area of distributed information systems. The widespread usage of mobile computing and cluster technology in the ongoing growth of the Internet, brought new opportunities in advanced database systems. In the past few years, challenging applications such as e-commerce, tele-banking, or remote airline check-in had arisen. Data replication in such environments is the major key for efficiency as well as for fault tolerance. For instance, fault tolerance is improved when data is replicated across different sites because when servers fail, other existing sites will take over. Moreover, reducing response time and increasing the throughput of the system is

also achieved because reading is done from the local sites instead of the remote ones. However, replication has some overhead because replicas should always be in a consistent state and should show the final updated state of the data. So, it is important to employ the best replication technique which overcomes such overhead.

Currently, replication techniques are defined by synchronous and asynchronous; they are also known as eager and lazy approaches respectively (Gray et. al, 1996). Before the commitment of a transaction, the synchronous approach propagates the updates and coordinates with the other replicas. Therefore, obviously this technique provides transaction isolation, fault tolerance, and data consistency. Moreover, such protocols show flexibility because they permit available replicas to be updated. As there is huge number of synchronous techniques, only few have been used in commercial systems because database developers believe that solutions that already exists lack feasibility due to their poor performance, lack of scalability, and high complexity (Gray et. al, 1996). Lately, ongoing products consider the asynchronous approach. In the asynchronous approach propagating the updates to other available replicas is done only after the commitment of the transaction (Stacey, 1994). Therefore, fault tolerance is limited and replicas lose their consistent state as a result. However, a lot of research has been done for developing lazy asynchronous approach that solves the problem of inconsistency (Anderson et. al, 1998), (Breitbart and Korth, 1997), (Chundi et. al, 1996), and (Pacitti et. al, 1999). Some solutions suggest that updates should be propagated to a primary copy which makes replication lose some of its advantages. Upon updating a primary copy, the single point of failure problem takes place; also the primary becomes a bottleneck.

Upon comparing both synchronous and asynchronous protocols, complementary behavior is noticed. Synchronous replication protocol ensures fault tolerance and consistency, however, it lacks performance. On the other hand, asynchronous replication protocol emphasizes on efficiency, but it shows poor degree of consistency. When it is not easy to synchronize copies, asynchronous replication would be the best protocol to use. However, in terms of performance, synchronous protocol would be the best replication approach to follow. In computer clusters, which are used to build up large databases such as the web farms and other internet sites (Buyya, 1999), synchronous replication is highly recommended (Rys et. al, 1996). The main advantage of synchronous replication is that it facilitates the work in complex environment by ensuring high availability and supporting load balancing. These observations had motivated us to further research the most known replication techniques, synchronous and asynchronous approaches, designing and proposing an architecture that provides data replication efficiently.

1.4 Structure of the Thesis

The thesis is structured as follows: in Chapter 2 we first introduce replica control. Then we present background information about database replication techniques and protocols. We highlight in this section on the major drawbacks and restrictions of the current replication approaches. Chapter 3 presents Multi-replication technique. It describes both asynchronous and synchronous multi-master replication methods, it presents the challenges with multi-master replication, and finally it introduces a

preventive multi-master replication technique applied in a cluster of autonomous databases. Chapter 4 presents the proposed architecture. It provides and discusses the algorithms for the proposed implementation design and finally we illustrate an example to show how the architecture works. Chapter 5 validates the proposed architecture. It compares the presented architecture with previous work and discusses how previous drawbacks are tackled in this proposed work. Finally, in Chapter 6 we summarize the conclusions in this work and we suggest further areas of research.

Chapter 2

Literature Review

Database replication is a wide area of research. Mainly, replication is one of the most active topics in databases that is under study and research by scientists and researchers, who are aiming to find better protocols, enhanced algorithms, and more reliable replication models.

As database replication was in papers for more than twenty years, many protocols were suggested and studied fully. Many studies and analyses have been done on eager or synchronous algorithms, voting method protocols, lazy or asynchronous algorithms, optimistic and partial replications.

In this chapter we provide the literature review of database replication and we present the cons and pros of some replication techniques and protocols. Mainly, we focus on the traditional solutions and their restrictions in order to differentiate between the avoidable and the inherent drawbacks, which support the main goal of our proposed architecture that overcomes the limitations of such solutions.

2.1 Basics of Replica Control

In a distributed database system, full replication demands every site to hold the exact or full amount of data in each site, whereas partial replication does not force databases to hold the full data in each site. Accessing data is handled by transactions, which characterize the logical units of the two basic read and write operations. Replica control and concurrency control are the two major elements of a replicated database system. Coordinating the access of each replica is handled by replica control, whereas isolating concurrent transactions with conflicting operations is handled by concurrency control.

The *1-copy serializability* is the best and strongest criteria for replication in terms of correctness (Bernstein et. al, 1987). As there are many replicas available, each object exists logically once and executing concurrent transactions is harmonized to ensure serializability. Moreover, atomicity of the transactions ensure that they all execute and commit successfully on all or none of the contributing replicas, in spite of the possibility of failure. Nevertheless, some replica control protocols offer undetermined levels of accuracy so as to ensure and provide high performance. In (Gray et. al, 1996), a classification of the mechanisms of replica control is presented: *when* transactions are propagated among replicas and *where* these transactions can be executed (Table 2.1).

Table 2.1: Categorization of Replica Control (Gray et. al, 1996)

where \ when	Eager	Lazy
Primary Copy	Early Solutions in Ingres	Sybase/IBM/Oracle Placement Strat.
	Serialization–Graph based	
Update Everywhere	Quorum based ROWA/ROWAA Oracle Synchr. Repl.	Oracle/Sybase/IBM Weak Consistency Strat.

There are mainly two cases for update to take place. Either it occurs within the transaction boundaries, in which replication is known as eager or synchronous, or it occurs after the transaction commitment, in which replication is known as lazy or asynchronous. Following the synchronous approach, conflicts could be detected before the commitment of the transaction. This technique ensures data consistency; however, it leads to a significant increase of response times due to communication overhead. To overcome this issue, the asynchronous approach carry out the propagated updates as a background process by delaying the transfer of updates till the transaction commits. Yet, as replicas diverge, the problem of inconsistency arises.

There are two cases for considering which replica to update, either a central update which occurs on a primary copy, or a distributed update which occurs everywhere. Following the centralized update approach, specific data item updates are executed first at the primary replica of this data item and then these updates are propagated to the other secondary replicas.

This approach simplifies concurrency control by avoiding concurrent updates to other replicas; however, it has two major drawbacks. It leads to single point of failure and it introduces a potential bottleneck. Using update everywhere approach requires the coordination of the updates when any copy of data item is updated. In synchronous or eager approach, this opens up the problem of high communication during the execution time of the transaction. In asynchronous or lazy approach, inconsistency problem arises when the same data item of different copies is updated by two different transactions and both commit locally before the update is propagated. Therefore, such conflict which leads to inconsistency should be identified and rectified.

2.2 Eager Replication

There exist many different replication methods which are used for various goals. Eager replication is one of the traditional techniques, which ensures update's propagation to happen before the transaction commits. It enforces each copy to be exactly synchronized in each and every site through propagating the updates as one big transaction. Serializability is enforced in executing transactions; therefore, concurrency control is supported (Gray et. al, 1996). It provides strong consistency because a transaction will not commit until it is certain that it will commit in all other available sites. However, it delays transaction execution.

Atomicity and 1-copy serializability can be obtained upon following the synchronous replication protocol. Timestamp based algorithms or the 2-phase locking algorithms are some of the replica control solutions which are combined with concurrency control to

provide serializability. Moreover, 2-phase commit, which is an atomic commitment protocol, runs at the end of the transaction to provide atomicity.

In table 2.1, we present a classification of some traditional protocols. Old protocols, such as the distributed INGRES, are known to follow the synchronous primary copy approach (Alsberg, 1976) and (Stonebraker, 1979). On the other hand, many architectures do not follow the centralized approach; instead they consider the update everywhere approach ensuring 1-copy equivalence. Another simple replication solution is read one write all *(ROWA)* (Bernstein et. al, 1987), in which update transactions are required to propagate to all replicas whereas the read transactions are performed locally, once. One major disadvantage that results from ROWA approach is that it does not provide fault tolerance. An enhanced approach is presented which overcome the replica failures. *Read one write all available* propagates updates to all the available replicas only (Bernstein and Goodman, 1984) and (Goodman et. al, 1983).

In spite of the advantages that eager replication techniques provides, there exist a small number of commercial systems which implement such eager techniques. An eager technique is provided by Oracle Advanced Replication (Oracle, 1997) which is performed through processes which are executed by triggers. First, an update is locally activated and then after the corresponding row triggers are fired to synchronously transfer the updates and to lock the other existing replicas. However, Oracle recommends using eager protocols only when applications require replicas to be constantly synchronized. It also enlightens the limitations of this protocol. Propagating data synchronously depends on the availability of the network and the entire system.

The major benefit of the eager replication model is that it guarantees consistency and one-copy serializability, which in turn facilitates database crash recovery. This comes out of the fact that any conflicts or failures of replica nodes can be discovered at commit time, and reported to the client who issued the commit. A major drawback is the fact that the commit of the updating transaction is delayed until all nodes which are up and running have acknowledged that they have received the updates. Thus, eager replication may potentially severely limit scalability.

In mobile applications systems, where nodes are more often disconnected, eager replication is not a good method to consider (Gray et. al, 1996). Such applications need to follow the asynchronous replication approach, where updates are propagated to all other sites only after the updating transaction commits.

2.3 Lazy Replication

As eager replication produces low level of performance and provides some complexity, lazy solutions have been developed to overcome such limitations (Chundi et. al, 1996; Pacitti et. al, 1999; Breitbart et. al, 1999). In lazy protocols, transactions are propagated to all replicas after they execute and commit locally; thus, response time is reduced. In *very stricted primary copy configuration*, 1-copy serializability is guaranteed. However, when a transaction commits and the site fails to propagate it to the other nodes, this transaction will be lost. Therefore, in this case there is no guarantee for atomicity.

Unlike eager replication, lazy replication propagates updates to replicated items asynchronously, outside of the main update transaction. Upon receiving of the

transaction's commit statement, the database implementing lazy replication commits the transaction locally and replies without delay. After some predefined period of time, the replication module refreshes transactions on each of the replica nodes in order to propagate the changes. This replication model is very scalable, because the commit of the main transaction is not delayed as it is in the eager case. It also facilitates disconnected operation and is thus suitable for mobile devices. However, this flexibility incurs windows of vulnerability where committed updates could potentially be lost if the node that executed the update transaction crashes before it had the chance to update the remaining replicas.

2.4 Quorum Consensus Approach

Quorum consensus protocol, which is a voting technique, is one of the database replication methods. (Thomas, 1979) presented the first database replication technique that follows the voting approach. The proposed consensus algorithm assumes that all replicas are exactly synchronized and replicated. The timestamp of the last transaction which updates the database is attached to this data item. The read operation returns its target along with the timestamp associated with it, whereas the write operation is performed in several steps to determine the updates that should be written and then after synchronization updates will take place based on the votes. Write or update transactions can not commit before the majority replicas accept, whereas read transactions can be performed locally. The ordering method that is performed through the timestamp is followed in both the voting process and the application of the consent update operations.

In the step that performs synchronization, a node checks the timestamp of every data item with its local copy timestamp. The requested update is rejected if it is old; otherwise, it is accepted and it gets a pass vote according to its priority among other requested updates. In order to prevent conflicts, the rule of voting does not allow a node to vote OK to more than one update request. As there is one node at least in common among the majority of the consenting nodes for any two requests with accepted updates, the database consistency is guaranteed. In this approach, messages are guaranteed to be delivered even when the site that should receive it is not accessible; hence, it ensures reliability in sending messages (Son, 1988). Moreover, mutual consistency is also guaranteed by the updates.

2.5 Partial Replication

Partial replication is another replication technique considered in the distributed database systems. This method is used when the application does not need or even can not replicate all the data on each site. Some nodes can hold certain data which is not replicated or available on other nodes. Partial replication allows a data copy to exist in some sites instead of every location site. Therefore, this technique considers data allocation, relation partitioning, distributed query processing, and load balancing. The need of such technique is to handle big databases and to ensure a balanced way of accessing data (Mukkamala et. al, 1988). Therefore, the replication technique provides high flexibility and guarantees high system performance. The main draw back of such technique is that it lacks data consistency among all the available nodes.

2.6 Optimistic and Pessimistic Replication

Designers classify replication methods into optimistic and pessimistic. Unlike the synchronous approach, optimistic algorithms propagate updates in the background and detect conflicts only after they occur (Shapiro and Saito, 2005). Optimistic approaches use different algorithms than those that are used in the traditional pessimistic approaches. Replication techniques used in the pessimistic approach try to conserve one copy consistent (Herlihy and Wing, 1990), as if one single highly available site is being accessed. This technique blocks any access to a site if it is not fully synchronized. Once the primary becomes unavailable, the other replicas choose a new primary to take place.

This approach performs well in a small network where delay is short. Unlike the pessimistic approach, optimistic replication methods perform highly in mobile environments or wide area networks. The concurrency control mechanism is the main difference between the optimistic and the pessimistic replication techniques. Optimistic replication technique propagates the updates in the background, and then they fix conflicts after they take place.

Chapter 3

Multi-Master Replication

Multi-master replication technique is one of the latest techniques in the field of database replication. The environment of multi-master replication is as follows: each site is considered to be a primary; none of the available sites is considered secondary. All the sites contain the same and complete amount of data and all databases are fully synchronized at all times. Writing to site and deleting from site is done to all sites at the same time. Simultaneously, when a modification is done, it is propagated to all other available sites in the system, whereas reading is done from the first available site. This keeps the replica in a full synchronization state; however, some problems are introduced which will be discussed shortly in the coming sections.

3.1 General Overview

Multi-master replication technique enforces data items to be always synchronized. When any update or delete transaction is executed in one database, this transaction will be immediately propagated in the same way to all other replicas in the system.

The multi-master replication architecture consists of a group of databases forming a replication group. One of these databases is considered as the master definition site; where as all the other databases are defined as master sites (Keating, 2001). The job of the master definition is site is to invoke most of the replication administration commands.

3.2 Multi-master Replication Methods

There are two methods with multi-master replication. The first method is the synchronous multi-master replication, which is also known as eager replication. This method causes every transaction to propagate to the other available sites directly. If any database crashes or looses connection and cannot accept the propagated transaction, the transaction will not be accepted by any of the available replicas within the replication group. The synchronous multi-master replication method requires each and every replica to be online or available so as for the transaction to be accepted, else propagation will not occur to any of the replicas; hence, synchronization will not take place. Therefore, the only case where synchronization of a replicated group can happen is when all of the replication group members are available at all time and ready to receive transactions.

The second multi-master replication method is asynchronous multi-master replication. This method, which is also known as the lazy multi-master replication, temporary stores the transactions that should be executed on a replica in the deferred transaction queue. In this approach, a push job is periodically performed which sends all the available transactions in the buffer to all the available replicas. After a specific period of time, all the transactions that have been sent to all the replicas will be purged out from

the queue so as to stop the buffer from the incremental growth (Keating, 2001). Therefore, in asynchronous multi-master replication all the replicas that are available all the time will be immediately updated and synchronized when they receive the transactions from the buffer, whereas other replicas which are unavailable due to a failure or disconnection, are considered asynchronous and transactions will be propagated to them when they are back online and connected to the replication group.

3.3 Synchronous versus Asynchronous Replication

In spite of being a new technique that is developed and proposed to overcome the weaknesses of traditional replication methods, multi-master replication technique faces some challenges. Asynchronous multi-master replication has been around longer than synchronous multi-master replication (Keating, 2001). In this section we will point out the advantages that the asynchronous multi-master replication has over the synchronous multi-master replication.

Asynchronous replication provides higher performance and less network communication than the synchronous replication. Moreover, asynchronous replication provides higher efficiency as it stores a bulk of transactions in a buffer and then it propagates them all as one big transaction instead of propagating each transaction individually. This approach is highly desirable when replicating remote nodes which are distributed in different regions.

The synchronous replication approach results in a high overhead as it demands each transaction to propagate individually to other replicas; hence, establishing a separate

connection. Fewer connections are established while propagating transactions in the asynchronous multi-master replication; propagation is done to a set of transactions instead of individual ones.

Furthermore, asynchronous multi-master replication offers higher availability. When any site becomes offline in the asynchronous multi-master replication environment, all other replicas which are available will receive the updates and will be synchronized. Reading and writing to replicas will still be possible. Those crashed nodes which did not receive the transactions will be informed about the updates as soon as they become online and transactions will be propagated to them from the deferred transaction queue.

In synchronous multi-master replication environment, if there exist a single crashed site, which has been disconnected due to a certain failure, propagation of transaction will not take place. Synchronous multi-master replication provides lower availability because in order for propagation and synchronization to take place, it forces all the nodes in the replication group to be available. This method follows the all-or-none approach. A single disconnected site will prevent propagation of transaction to occur.

3.4 Problems with Multi-master Replication

Although multi-master replication shows flexibility and powerful characteristics, some complexity exists. After mentioning all the beneficial characteristics that exists in the asynchronous multi-master replication, there exists a single major weakness that results from this replication technique. Multi-master replication leads to divergence in

data. This section discusses the various data conflicts that results from asynchronous replication.

3.4.1 Possibility of conflicts

As asynchronous replication approach stores all transactions then forwards them after a specific period of time, some operation conflicts exist.

One of the operation conflicts that results from asynchronous multi-master replication is the update conflict. This conflict occurs whenever a certain data item is propagated and modified on more than one database before each of these databases propagate this update to the other sites. Here is an example showing the update conflict. Assume that the coordinator propagates transactions after a period of 5 units. At time=1, one of the clients logs into a site X and modifies a certain data item to "London". Later, at time =4, another client logs into a site Y and modifies the same data item to "New York", keeping in mind that updates occurred on site X is not yet propagated to other sites. At time=5, which is time for the coordinator to broadcast the updates, a conflict error (update conflict) will be encountered since the same data item has been altered more than once, which will confuse the coordinator about the value of this data item, "London" or "New York".

Another conflict which takes place in multi-master replication is the uniqueness conflict. This conflict occurs when propagating records to replicas results in having the same value for the same data item record to more than one row for the same column, with a restriction that no duplicate values are allowed in that column. Here is a scenario which shows such conflict. Suppose that User_ID is one of the columns in a database table that

has to be replicated taking into consideration that it is a primary key; hence, it should have unique values. A user logs in to site X at time=1 with User_ID= "Michael Jordan" and updates the data item row User_ID to "Michael J". Another user logs in to site Y at time=4 with User_ID= "Michael Jackson" and updates the data item row User_ID column to "Michael J" also. Assume at time=5 the coordinator wants to propagate the changes. Unique conflict error will be generated as it detects that more than one row of the same primary filed column has a duplicate value.

One last conflict that occurs in the asynchronous multi-master replication is the delete conflict. This type of conflict happens when some data item rows are taken off totally from one site and when the coordinator propagates this delete transactions, it detects that it cannot find the target data items in other replicas. In other words, the coordinator is trying to delete a record which has been modified or already deleted, generating a delete conflict.

Resolving such types of conflicts that occur in the asynchronous multi-master replication is handled by conflict resolution handlers. Mainly, the conflict resolution handlers for update conflicts perform better than those for the uniqueness conflicts; where as delete conflicts have no resolution handlers.

One of the update resolution handlers is site priority, which gives one replica higher priority over all the other replicas. Therefore, when there is an update conflict among several replicas, the updates of the replica with the highest priority will be considered to be the final value for all replicas. Average and timestamp handlers are two other update conflict handlers. The average resolution handler considers the average value of the

conflicting data items, whereas the timestamp resolution handler considers the last or the recent update to be propagated to replicas.

3.4.2 Extremely Short Push Intervals

The asynchronous multi-master replication forces the transactions in the deferred queue to be pushed out from the buffer to all other replicas every period of time. A lot of system recourses will be consumed if the period is scheduled to be relatively short. Starting and ending a push job require high CPU resources; therefore as much as these jobs are active, high CPU usage will be consumed. Executing push processes more often causes the CPU to fall behind; hence, propagation of transactions to all replicas will be delayed.

3.4.3 Monitoring Utility for Synchronous Replication

In synchronous multi-master replication, updating replicas require each and every site in the replication group to be accessible, running, and active. Therefore, transactions will be immediately propagated to every site for execution; else if there is a single site that is not active or disconnected, transactions will be ignored.

In order to overcome such replication issue, running a utility that would always check the status of every replica would be a good solution. This utility will ensure whether the replicas can be accessed regularly. It would try to recover the failed replica by reconnecting it to the replication group. This utility application tries to restart a node

from failure and if the node does not respond, it would dynamically remove it from the replication group.

3.4.4 Adding Replicas

With synchronous multi-master replication, adding a replica to a replication group is not simple. A simple solution for adding new node is to consider an asynchronous replication system in which one site can be brought offline so as it propagates all its content to the new added node. Asynchronous replication system saves all transactions in a buffer. Therefore, when all replicas become online they will synchronize all together.

Two approaches are proposed for adding new replicas. The first technique suggests replicating with a stand alone program the contents of any synchronized node to the new site. This technique is not feasible due to the big work needed to be done which may affect the whole system performance.

The second proposed technique for adding a new replica node is to make the whole system unavailable for some period of time, until the newly added node is fully synchronized with the other nodes in the replication group. This technique has also one main disadvantage that is it affects the system availability because at some time the system is not available to perform any request or client operation.

3.4.5 Creating Many Replication Groups

When some data items in a database holds time sensitive data such as stock quotes, synchronous replication becomes a need. On the other hand, when data items in a database holds less time sensitive data such as salary records, asynchronous replication would be the best approach to follow for replication.

The benefit of having multiple replication groups results in the low network traffic. The time sensitive data propagates more frequently unlike the less time sensitive data that propagates much less often.

Creating different replication groups is something doable and feasible within a single database. It is a good solution for having within the same database both synchronous and asynchronous replication groups. However, one restriction that should be considered in using multiple replication groups is that data item should belong only to one replication group; either to the synchronous or to the asynchronous replication group.

Having different replication groups with a replicated system seems to be a great idea (Keating, 2001); however, it is a complicated and complex technique to follow as it is not easy to manage and configure such system.

3.4.6 The Quick "Purge" Method

As we mentioned earlier, asynchronous multi-master replication places transactions in a buffer called the "deferred transaction queue". Periodically, these transactions are propagated to all the available nodes in the replication group. The transactions that have been sent to all the replicas successfully should be purged out from the buffer so as to avoid the buffer from the incremental increase in size. Improper scheduling for the purging process will result the system to do extra work that is undesired, affecting the system performance.

3.5 Preventive Multi-master Replication

Clustering sites is a good solution for managing replication. As some system applications has their databases stored at different remote locations, it is more efficient to cluster such databases and manage their replication in parallel which would also enhance the load balancing of the whole system. The main advantage for configuring such architecture is replicating data across remote nodes which will increase the response time for requests and will ensure the system availability in case of any replica failure. However, on the presence of updates, managing data consistency of the remote databases in such system architecture is not an easy task (Gray et al, 1996).

In (Pacitti, 2003), a multi-master preventive technique is proposed which tries to solve the problem of data consistency for a group of clustered databases.

As we mentioned previously regarding the asynchronous multi-master replication, transactions would commit only when they update a database copy totally at some node. Therefore, when transactions finish from updating one database, they are propagated to the other sites immediately; hence, maintaining mutual consistency (Pacitti et al,. 2003). In this proposed lazy replication environment there are two types of nodes. There is one master node called the multi owner node, and all the other nodes in the replication group are considered slave nodes.

3.5.1 Multi-master Cluster Architecture

The multi-master cluster architecture consists of a group of replicated databases/replicas. Each replica consists of the following layers: request router layer, application manager layer, replication manager layer, and transaction load balancer.

The general processing of a user request starts upon its arrival to the replication group. Propagation of the user request is delivered randomly to a node in the clustered group. Using the global user directory services, authentication and authorization is handled by the request router of the current node. User requests are managed completely asynchronously. After authentication, the requests are routed to the proper node which have the corresponding application and have the lightest load. The current load monitor utility, which exists on each node, is responsible for determining the load weight on each replica. It periodically monitors the load generated by the requested transactions and the running application on each node using traditional methods.

Each load monitor utility then broadcasts the determined weight of corresponding replica to all other replicas within the cluster. This weight will be used to choose the best replica which is the one with lowest load.

Managing application instantiation and execution is done by the application manager layer. Whenever there is an execution for a transaction via the proper application service provider, the transaction load balancer selects the node with the lightest load on which the transaction will take place.

Managing access to replicated data is done by the replication manager which enforces transactions to execute in a serial order in every site. This will guarantee and ensure strong consistency in the replication system. Whenever a transaction commits or aborts, the replication manager informs the transaction load balancer which in turn communicates with the application manager layer.

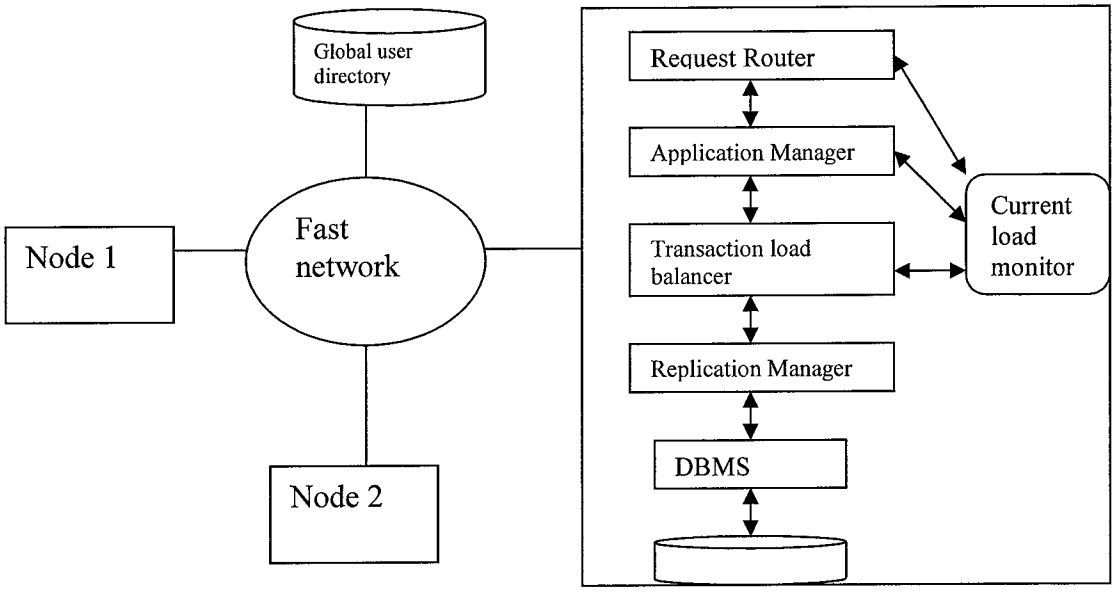

Figure 3.1: Cluster Architecture

3.5.2 Multi-master Refreshment

Consider a multi-master replication environment that provides a reliable FIFO multicasting communication, in which messages are delivered in the same serial order they have been sent. Consider the parameter MAX which is the maximum time needed for a message to be delivered to the target node. Assume a variable x to be the maximum difference between any two correct local clocks which are presented on each node.

Following the FIFO order approach does not guarantee strong consistency in multi-master replication environment. Sometimes there exists a group of nodes which have at least two unordered transactions. Therefore, inconsistency may result.

To overcome inconsistency, the refresher algorithm considers a parameter C which corresponds to the transaction's timestamp. The chronological order of these values will enhance the serial propagation and execution of transactions to guarantee consistency. Therefore, the goal of the refreshment procedure is to present a sequence of transactions, which are considered to have the same chronological order at each node. The procedure starts by checking if there is an old transaction in a node before it submits a new transaction to that node. When a new transaction is submitted at time Ct, it will be delayed by MAX + x. Therefore, a new transaction will be submitted at C + Max + x; thus, chronological and total orderings are assured.

When a transaction is propagated to a certain node, the node in turn transmits that transaction to all the available nodes in the replication group including itself. The transaction is then placed in FIFO order queue with respect to the triggering site. Each node in the multi-master replication group has a number of pending queues

corresponding to the number of sites. Every pending queue, which is used to establish the chronological ordering, corresponds to a multi-master node.

Let us illustrate the algorithm by an example. Suppose we have two nodes i and j, masters of the copy R. So at node i, there are two pending queues: q(i) and q(j) corresponding to multi-master nodes i and j. T1 and T2 are two transactions which update R, respectively on node i and on node j. Let us suppose that Max is equal to 10 and ε is equal to 1. So, on node i, we have the following sequence of execution:

- At time 10: T2 arrives at node i with a timestamp C2 = 5

• q(i) = [T2 (5)], q(j) = [ ]

• T2 is chosen by the Refresher to be the next transaction to perform at delivery_time 16 (5 + 10 + 1), and the time is set to expire at time 16.

- At time 12: T1 arrives from node j with a timestamp C1 = 3

• q(i) = [T2 (5)], q(j) = [T1 (3)]

• T1 is chosen by the Refresher to be the next transaction to perform at delivery_time 14 (3 + 10 + 1), and the time is re-set to expire at time 14.

- At time 14: the timeout expires and the Refresher writes T1 into the running queue.

• q(i) = [T2 (5)], q(j) = [ ]

• T2 is selected to be the next transaction to perform at delivery_time 16 (5 + 10 + 1)

- At time 16: the timeout expires. The Refresher writes T2 into the running queue.

• q(i) = [ ], q(j) = [ ]

Although the transactions are received in wrong order with respect to their timestamps (T2 then T1) they are written into the running queue in chronological order according to their timestamps (T1 then T2). Thus, the total order is enforced even if messages are not sent in total order.

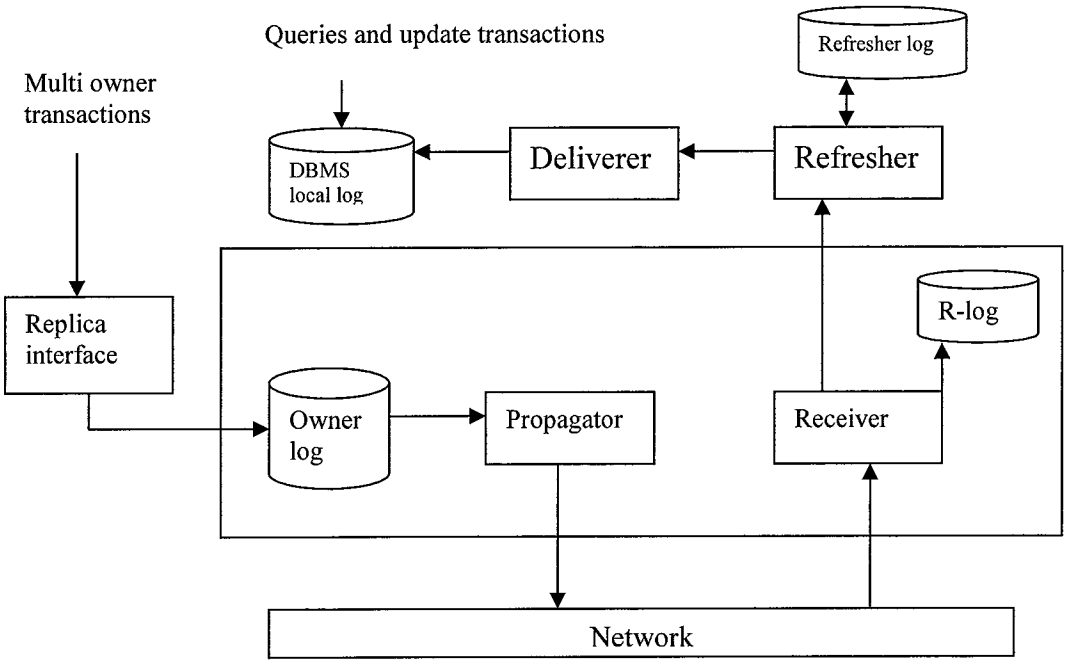

Figure 3.2: Multi Owner Node Architecture

The main elements for running the refreshment procedure are presented in figure 3.2. Managing the incoming requests is done through the Replica interface. Executing the received requests and propagating them is done by the Receiver and the Propagator elements respectively. Reading transaction from the top queues and performing the chronological ordering is done by the Refresher. Once a transaction is ordered, then the Refresher writes it to the running queue in FIFO order, one after the other. Finally

Deliver keeps checking top of the running queue to start transaction execution, one after the other, in the local DBMS.

3.5.3 Refresher Algorithm

Figure 3.3 depicts the refresher algorithm. The three steps of the algorithm used in the Refresher module appear below. Step 1, at the reception of a new message, shows the selection of the message that is supposed to execute first. This is done by picking up the latest transaction according to its timestamp from the pending queue. Step 2, calculates the delivery time according to the time stamp of the message and the Max + x, and then a local reverse timer is set which will expire at the delivery_time. Finally, in step 3, when the timer is over, executing the transaction is done by submitting it to the running queue.

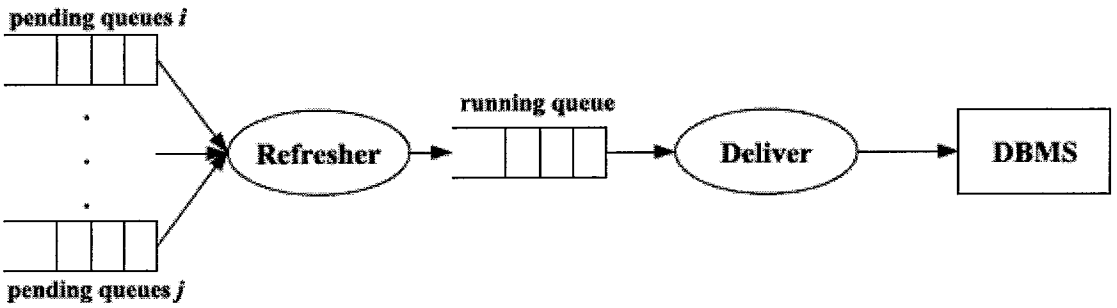

Figure 3.3: Refreshment Architecture

Multi-master Refresher

Input:
pending queues *q1,...qn*

Output:
running queue

Variables:
curr_T: currently selected transaction to be executed;
first_T: transaction with the lowest timestamp in the pending queue
timer: local reverse timer whose state is either active or inactive

Begin
timer.state = inactive;
curr_T = *first_T* = 0;
Repeat
On arrival of a new message
or when (*timer*.state = active **and** timer.value = 0) **do**

Step1:
first_T <- message with min *C* among top messages of the pending queues;

Step2:
If *first_T* <> *curr_T* **then**
    curr_M <- *first_T*;
    calculate *delivery_time*(*curr_T*);
    timer.value <- *delivery_time*(*curr_T*) – local_time
    timer.state <- active;
end if

Step 3:
If timer.state = active **and** timer.value = 0 **then**
    append *curr_T* to the running queue;
    dequeue *curr_T* from its pending queue;
    timer.state <- inactive;
end if
for ever
end

Chapter 4

Proposed Architecture

As multi-master replication considers both the synchronous and asynchronous approaches, this chapter describes the proposed database replication architecture which combines both synchronous and asynchronous multi-master approaches.

4.1 Overview

The synchronous multi-master replication technique requires all sites in the replication group to be available and connected all the time in order for transactions to be propagated to all replicas so as to get updated. Unfortunately, when one site fails to respond and gets disconnected from the replication group of the system, the whole cluster should be brought offline in order to recover the failed node and to bring all the nodes back to the synchronized stated. The same scenario is applied when adding additional replica to the replication group.

Unlike the synchronous multi-master replication, asynchronous multi-master replication technique considers each replica in the replication group as a master node and

allows some replicas to be offline at any time, which will not affect the replication group. In this technique, every replica is responsible about its state; hence, replicas do not rely on other replicas to receive transactions and updates. This replication technique requires transactions to be logged, therefore, recovered replicas which connect back to the replication group after a certain failure, can share this log and become resynchronized. This multi-master replication technique seems to be better than the synchronous multi-master replication. It shows powerfulness and flexibility in dealing with recovered nodes and adding replicas to the replication group. However, the only drawback that this replication technique presents is the usage of relatively high amount of network resources which this technique consumes upon communicating and synchronizing.

After describing the advantages and disadvantages of both the synchronous and asynchronous multi-master replication techniques, we found out that each technique lacks the reliability that makes it the optimal multi mater replication method for an entire replication system. This explains why the proposed multi-master replication techniques have not been deployed. Therefore, in this thesis we aim to find a better system architecture which holds the advantages of both the asynchronous and synchronous multi-master replication.

4.2 General Architecture

The general architecture of the proposed replication combines the advantages of both the synchronous and the asynchronous multi-master methods. The synchronous approach provides reliability and availability and the asynchronous approach provides flexibility and performance.

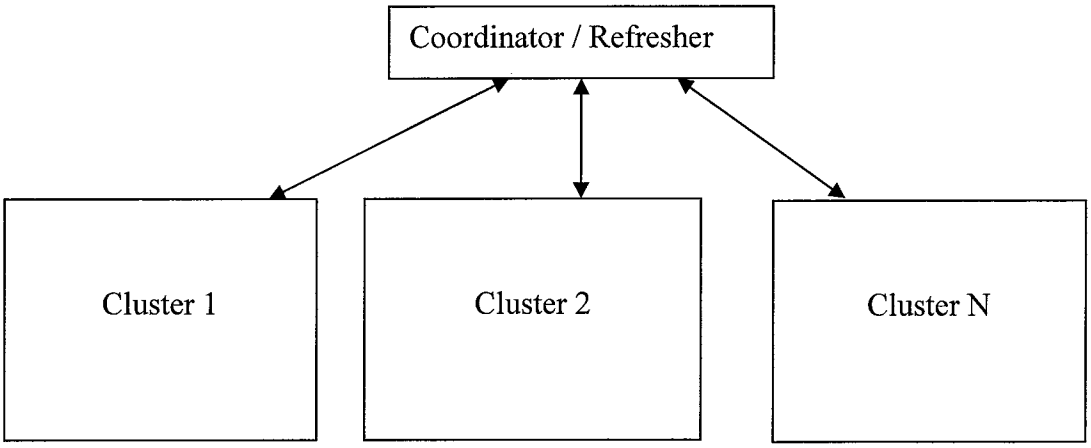

Figure 4.1: Overview of the Proposed Architecture

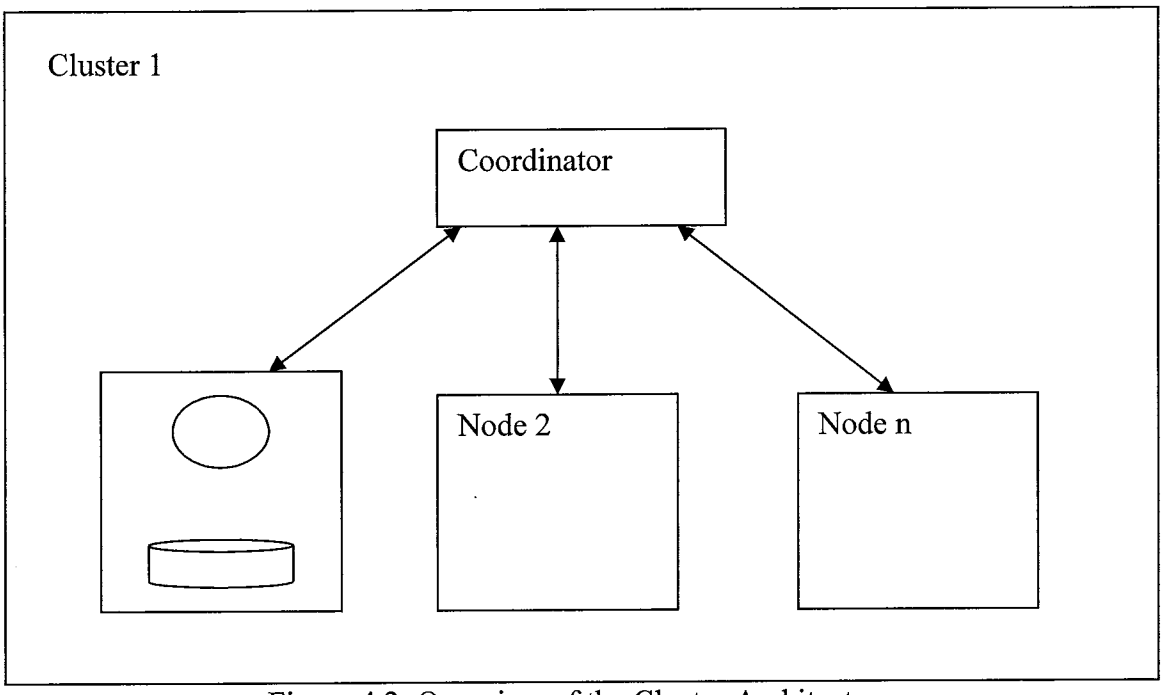

Figure 4.2: Overview of the Cluster Architecture

As figure 4.1 and 4.2 depict, the architecture is made up of two layers. The upper layer behaves asynchronously while dealing with clusters, whereas the lower layer behaves synchronously while dealing with the inner clustered nodes. Therefore, our proposed architecture behaves in a synchronous/asynchronous multi-master replication approach.

The top layer contains a group of clusters; every cluster consists of some nodes. The nodes are grouped or added to each cluster according to a certain criterion set by the system, such as distributing the nodes equally into clusters or distributing them according to the nearest site or region. For example, when adding new node to the system it may be inserted into the corresponding cluster with the least number of nodes in order to maintain equal number of nodes in each cluster. Say there are four clusters and a total of 20 nodes, so each cluster would contain five nodes. This criterion will maintain the

system performance by dealing with one cluster of nodes rather than dealing with the entire nodes that might be distributed to one cluster, and will ensure availability by bringing some clusters offline sometimes and maintaining other clusters online at all times.

Following the asynchronous multi-master approach, each client request is delivered to the coordinator of the first layer following the same way as it is in (Pacitti 2003). A measurement for all clusters is calculated to find the best cluster with the lightest transaction load to send the transactions to. Asynchronously, transactions propagate to the coordinator of the chosen cluster. Then synchronous multi-master approach will take place in propagating every transaction to each node within the cluster immediately. After updating the corresponding cluster, these transactions will be propagated again from the coordinator/refresher of the top layer to the second layer coordinator of the next best cluster following the same procedure. Eventually, all the nodes in each cluster will receive the transactions.

When any cluster gets disconnected from the system due to a certain system error or malfunction, transactions which are aware of the failed cluster, are saved in the logs of the main coordinator and then they will be forwarded to the best available cluster in the system. However, if one node fails in a certain cluster, it will be taken off totally from that cluster. Any incoming transaction to this cluster will be saved in the cluster's coordinator logs and will be propagated directly to all the active nodes within this cluster. When there is a recovered node or in case of adding additional node to a cluster, this node will synchronize with all the available nodes to become totally synchronized.

To resynchronize old existing nodes coming from failure or disconnection, the system puts the node back in the same cluster that it was under, and then it disconnects this cluster from the whole system in order for the internal nodes to be updated and synchronized internally as the case in synchronous replication method. In this case only one cluster is brought offline and the other clusters are kept online to maintain availability. The same procedure takes place when a newly available node is to be added to the replication system.

The advantage of this technique in handling failed nodes is to preserve the system performance. This technique removes the failed node in order to maintain system availability also.

4.3 Algorithms

In this section we present two algorithms. The first one corresponds to the architecture of the upper layer, mainly for the coordinator/refresher of this layer. The second one corresponds to the architecture or behavior of the lower layer coordinator.

4.3.1 Algorithm for Coordinator/Refresher

```
Do while system is up
  Wait for new transactions
  If new transactions Then
      Calculate cluster weights
      Find winner (least weight)
      If there exist any failure in a cluster then
          Forward transaction to refresher and next winner
      Else
          Forward to winner
      End If
  End If
End While
```

This algorithm waits for an incoming request transaction. Calculation of all active clusters is done for choosing the best cluster with lightest weight in order to forward transactions to. If the cluster contains no failures then transactions are propagated to it directly, else they will be forwarded back to the coordinator and to the cluster with the next lightest weight.

4.3.2 Algorithm for the Coordinator within the Cluster

```
Do while the cluster is up
  If new transactions then
        Propagate transactions synchronously
  Else if failure exists in a node then
        Remove node totally
  Else if old node is back online then
        Bring current cluster offline
        Resynchronize this node by requesting transactions from refresher
        Resynchronize the cluster by requesting transactions from coordinator/refresher
        Bring cluster online
   End if
End while
```

In this algorithm we first check if new transactions arrived in order to propagate them directly within the cluster. Then the algorithm checks if a failure exists in a node so as to remove it totally. Finally, it checks if an old node is back online so as to resynchronize it by bringing the cluster offline and requesting the transactions saved in the local refresher to be propagated again, and then to ensure that the cluster is already synchronized it request any new transactions available in the main coordinator/refresher.

Let us illustrate the algorithms by an example. Suppose we have four nodes distributed into two clusters, cluster A and cluster B respectively. Assume three transactions to be propagated or updated to all nodes. (Figure 4.3)

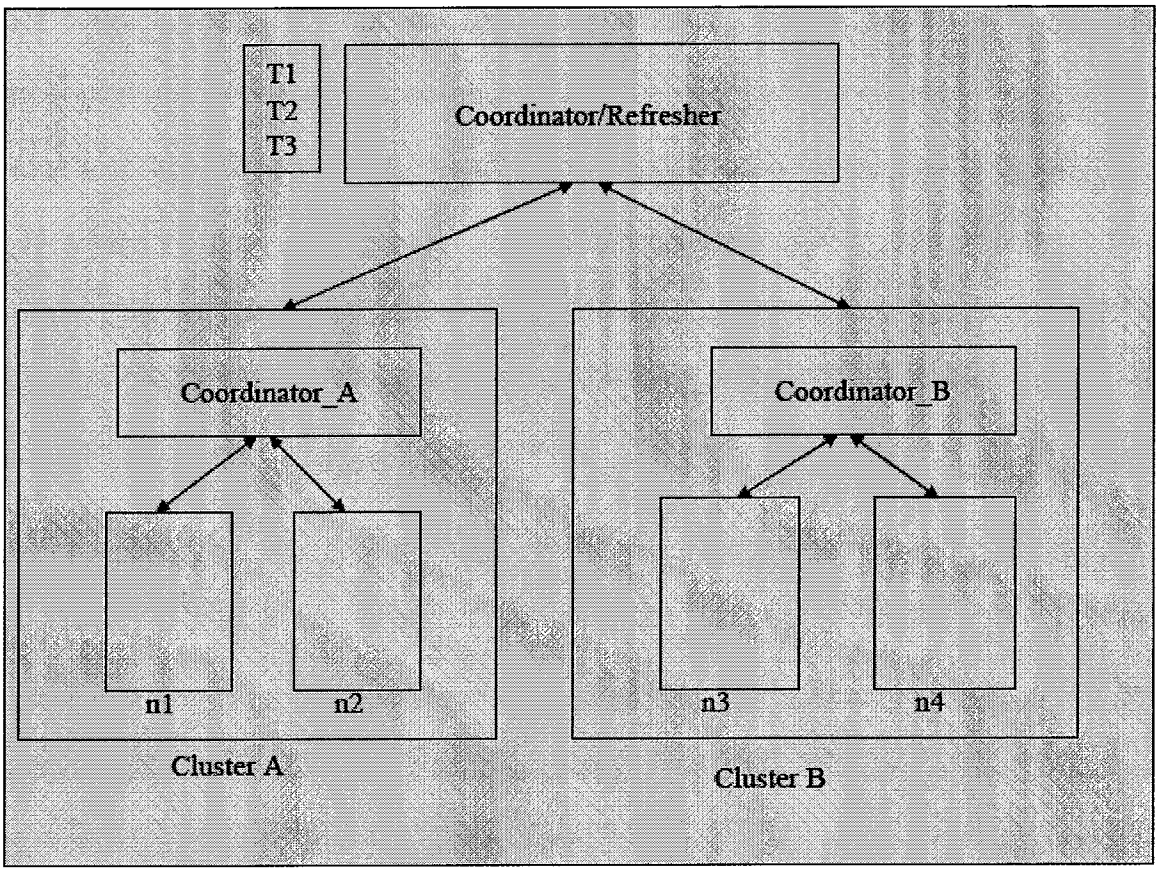

Figure 4.3: Initial State of the System

Following the proposed architecture, assume that cluster A has the lightest transaction load, therefore cluster A is the winner and transactions are to be propagated asynchronously to from the coordinator/refresher to Coordinator_A first. During this process, and until the first cluster (cluster A) to be updated and transactions to be propagated to all its internal nodes, any client request to the system will be delayed shortly so as data to maintain system reliability and to ensure data integrity. Therefore, during the process of cluster synchronization, only one cluster at a time will be locked or brought offline and when it finish data synchronization it will be brought online, and client request can access its nodes. (Figure 4.4)

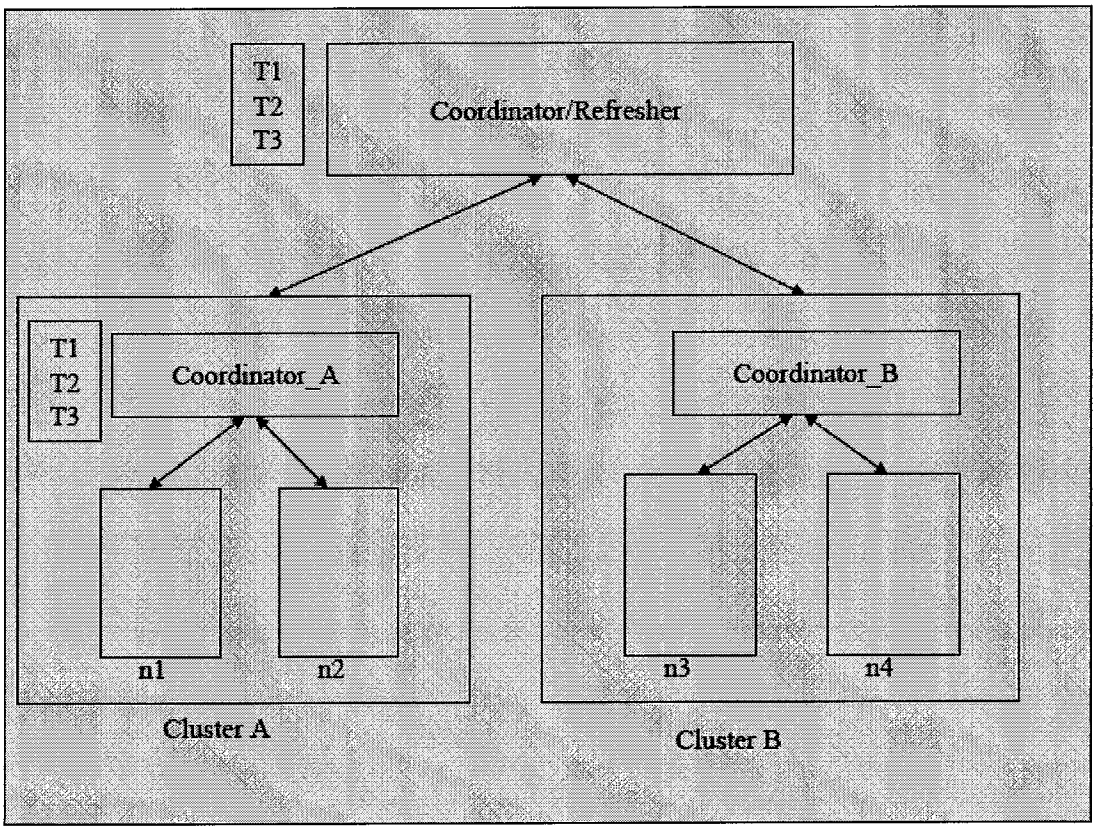

Figure 4.4: Asynchronous Propagation of T1, T2 and T3 to Coordinator_A.

In figure 4.5, Coordinator_A starts propagating T1 synchronously to n1 and n2,

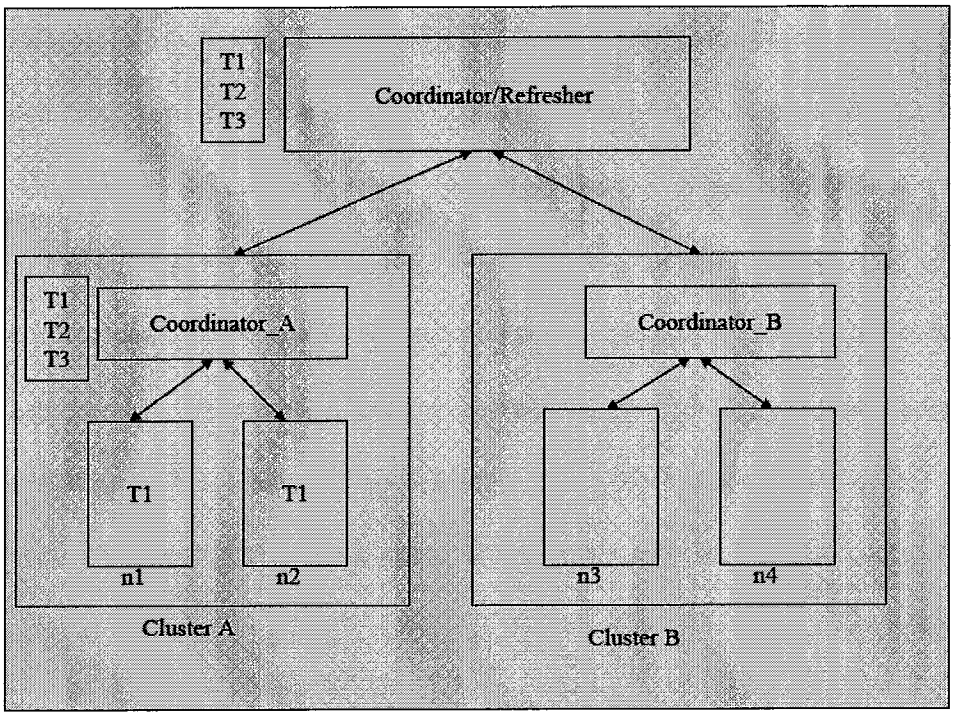

Figure 4.5: Synchronous Propagation of T1 to n1 and n2.

In figure 4.6, Coordinator_A propagates transaction T2 synchronously to n1 and n2.

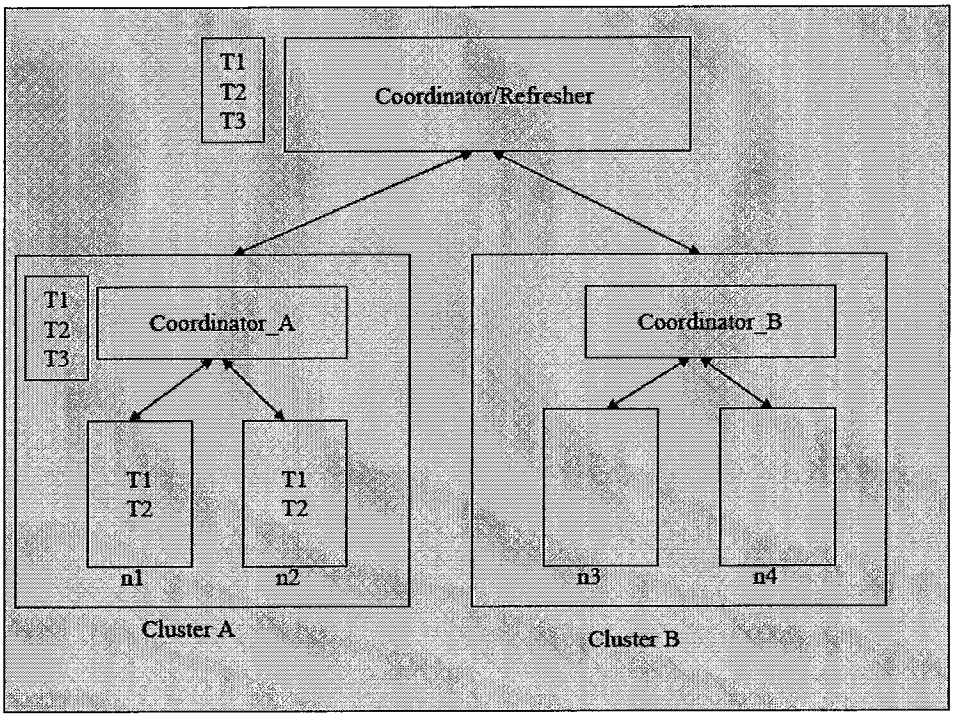

Figure 4.6: Synchronous Propagation of T2 to n1 and n2.

In figure 4.7, Coordinator_A propagates transaction T3 synchronously to n1 and n2. Now after Cluster A has been fully synchronized, it can be accessed by client's request, whereas Cluster B is chosen to be the next winner. Transactions are propagated from the coordinator/refresher to Coordinator_B asynchronously. Cluster B will be locked so as to propagate the transactions synchronously to n3 and n4.

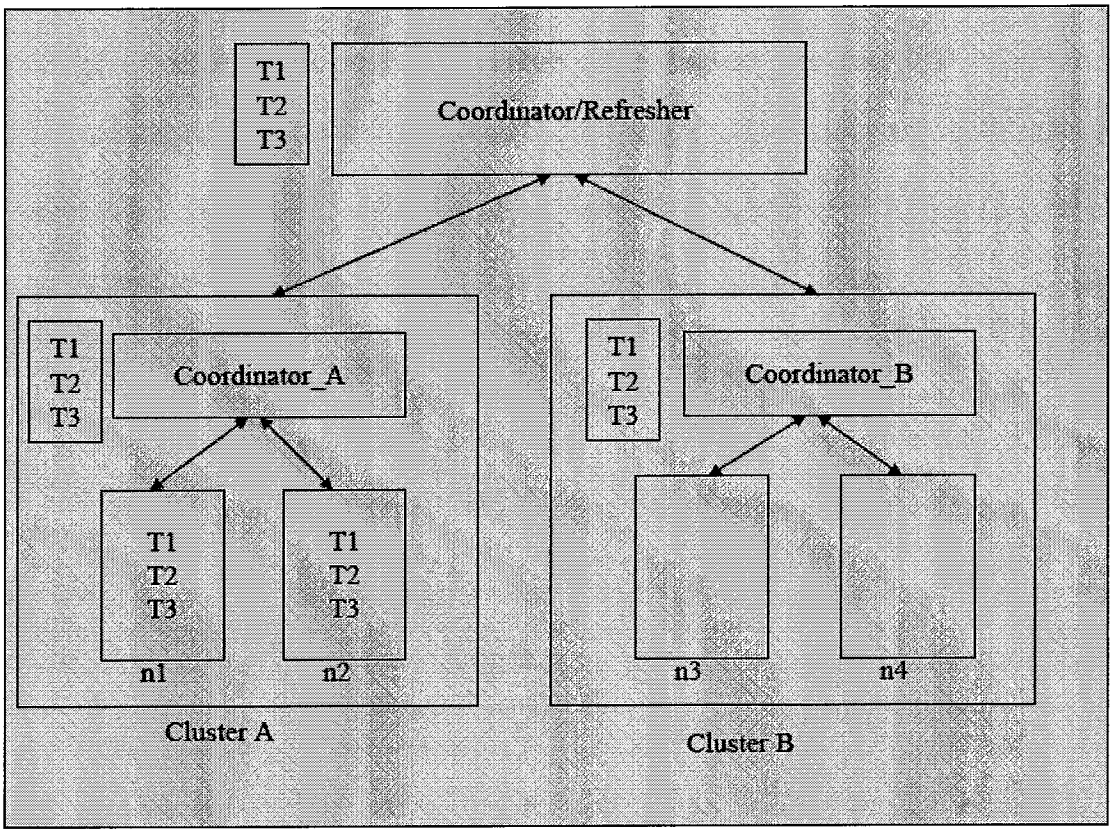

Figure 4.7: Cluster A has been Fully Synchronized.

In figure 4.8, Coordinator_B propagates transaction T1 synchronously to n3 and n4.

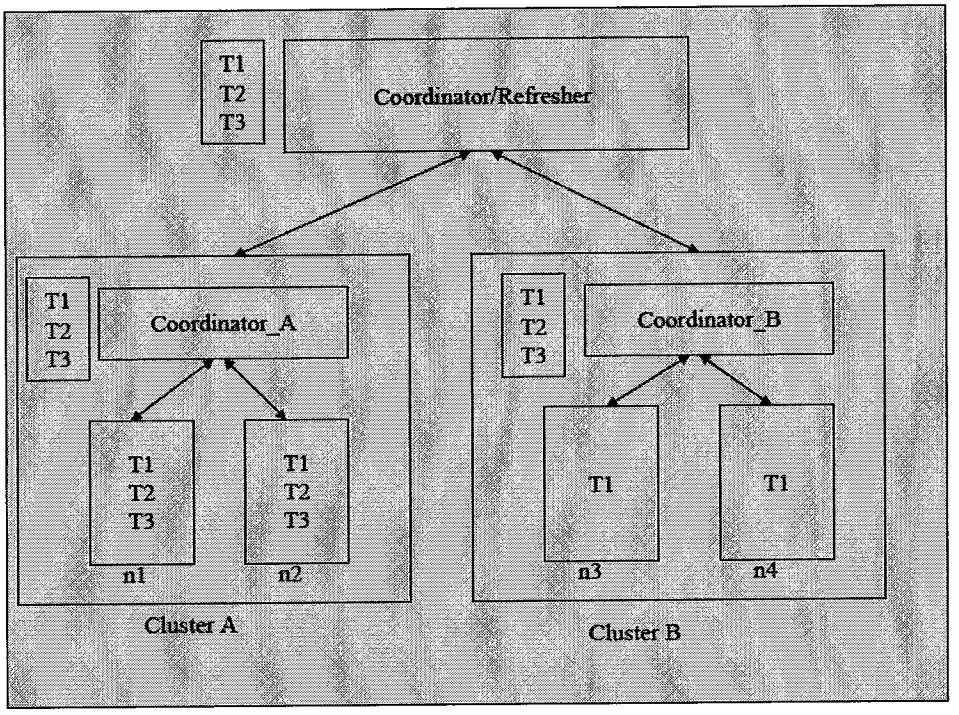

Figure 4.8: Synchronous Propagation of T1 to n3 and n4.

Figure 4.9 depicts the propagation of T2 synchronously to n3 and n4.

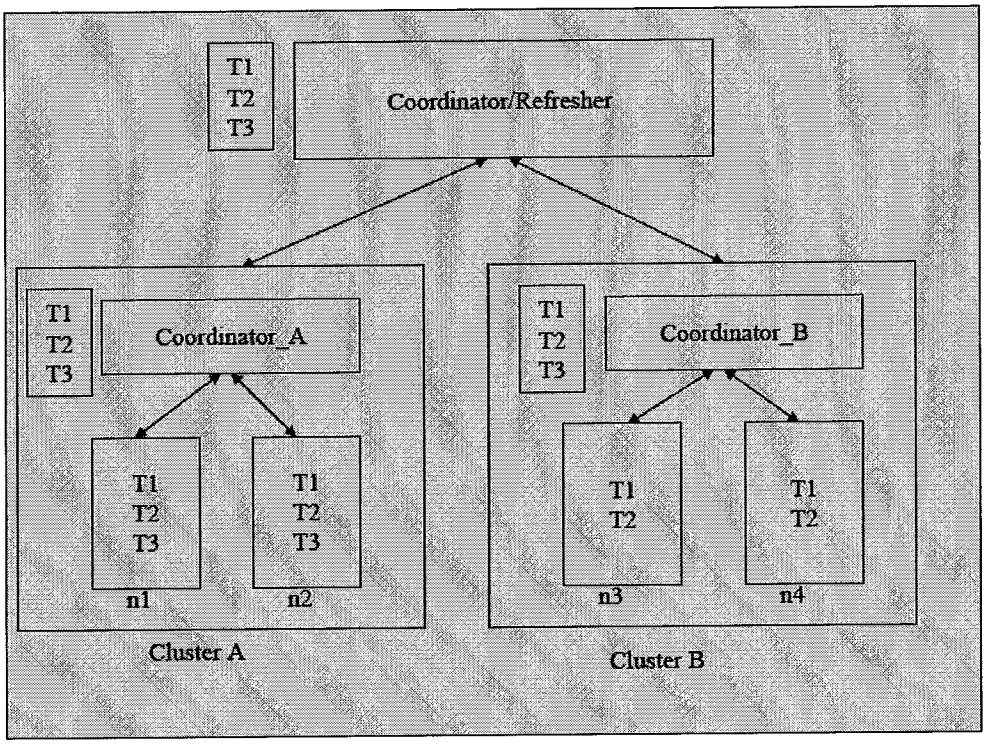

Figure 4.9: Synchronous Propagation of T2 to n3 and n4.

In figure 4.10, Coordinator_B propagates transaction T3 synchronously to n3 and n4. Now after Cluster B has been fully synchronized, it can be accessed by client's request. As there is no more clusters to be synchronized the system is set to be fully synchronized waiting for client's request.

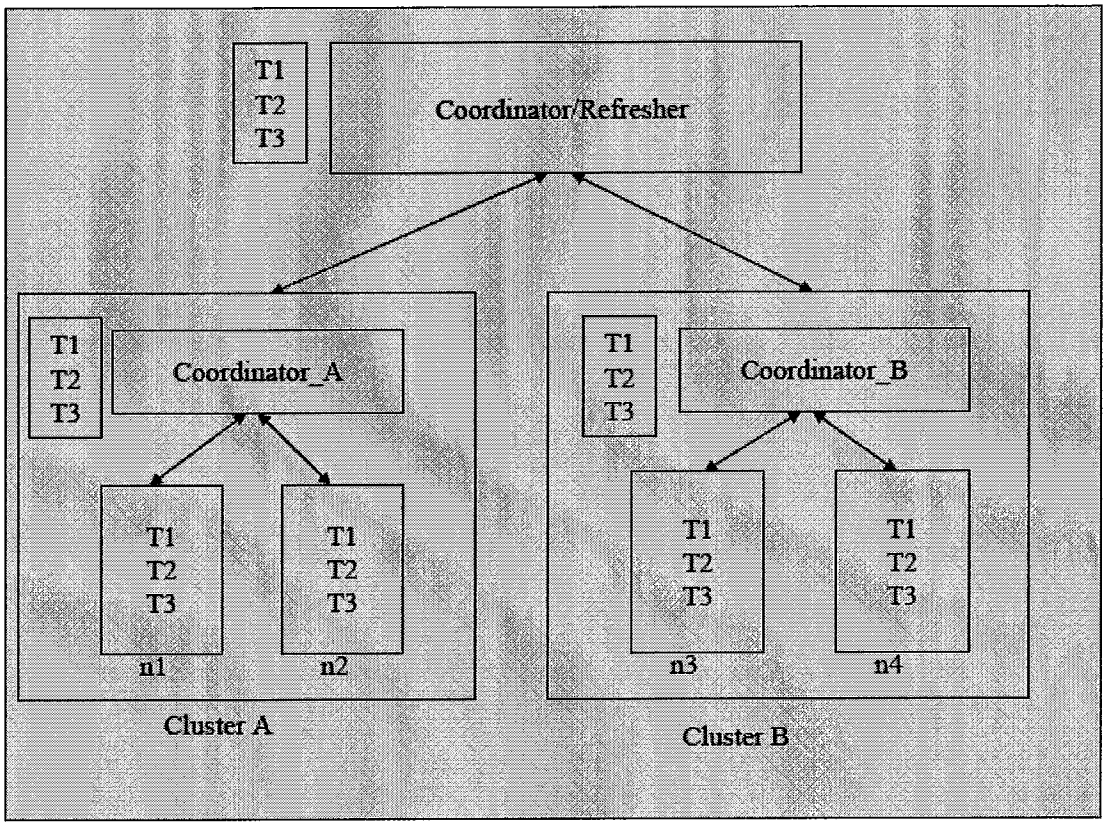

Figure 4.10: Final System State, Fully Synchronized.

Chapter 5

Validation

This chapter describes the advantages of our proposed synchronous/asynchronous multi-master replication architecture. We compare this architecture with the existing synchronous and asynchronous multi-master replication architectures. Mainly, we present a description about the main drawbacks that the existing solutions encounter and how our proposed solution overcomes these limitations. We focus on the main characteristics that build up a better replication system: efficiency, availability, performance, consistency and flexibility.

5.1 Efficiency

In our proposed synchronous/asynchronous multi-master replication architecture, we suggested grouping nodes into system defined clusters. Client requests are handled asynchronously which avoids the problem of bottleneck. When a set of transactions is to be propagated to all available replicas, one cluster of nodes (one with the lightest load) will be updated first. This will enhance the efficiency in retrieving an up-to-date data item from a certain replica. In the synchronous multi replication architecture, the set of

transactions will be propagated as a group to each replica node separately; hence, retrieving data items from the system would occur only when every replica is fully synchronized separately. In the asynchronous multi replication architecture, each transaction will be propagated separately to each replica node in the replication group. Therefore, retrieving a data item can be done only when all the transactions are propagated to all replicas successfully in the replication group.

In this thesis, the proposed architecture allows retrieving data faster than any other proposed architecture, because retrieving data can occurs when at least one cluster of nodes is fully synchronized.

5.2 Availability

Availability is one of the important aspects of good database replication architecture. Some known eager replication solutions demand bringing all the system down while propagating transactions. This is to ensure serializability and consistency in data propagation. The limitation of such solution is data availability. In other words, all nodes will be brought offline until transactions are propagated and nodes are synchronized.

In our system architecture, propagating transactions require bringing one cluster of nodes down at a time, until the process of synchronization is done. In this case, a delay for accessing a certain data can exist for a very short period of time until the first winner cluster finishes its process of internal data synchronization. Therefore, while other clusters are in the process of synchronization, reading can be done from any previous synchronized cluster.

Moreover, in eager replication solutions, when one node is disconnected from the replication group, propagation of transactions will not take place. Whereas in the proposed architecture, as it represents a synchronous replication approach, it does not demand all nodes to be available so as for propagation to take place. When a node becomes unavailable, only this node will be taken off from the cluster and the propagation of transaction will continue for the other available nodes in the cluster. When that node is back online, it will be placed in its original cluster and that cluster will start synchronizing within its internal nodes to assure that all nodes have the same data.

5.3 Performance

Synchronous replication demands all transactions to establish an individual connection to each replica in the group, which results in high overhead and low system performance. The proposed system architecture represents the asynchronous replication technique. Only few connections are needed as the propagation of transactions is done as a group asynchronously from the top layer coordinator to the bottom layer coordinator; hence, less system overhead. Although our system propagates the transactions within a cluster synchronously, demanding each node to establish a connection with the cluster coordinator, the load will be distributed to this group of nodes only; therefore, the overall system performance will not be affected.

Solutions based on total order broadcast are not well suited for large scale replication. Exchanging messages in order to provide total ordering would increase dramatically in a large scale replication system. The Database State Machine, (Sousa, 2001) and (Sousa, 2002), supports partial replication for heterogeneous databases and thus does not violate

autonomy. However, its synchronous protocol uses two-phase locking that is known for its poor scalability, thus making it inappropriate for database clusters. In our proposed system architecture, message exchange is done between a particular number of clusters instead of the all the nodes that exist in the replication group.

5.4 Consistency

Asynchronous or lazy replication typically trades consistency for performance. It propagates each transaction separately to each replica alone. With lazy replication, propagating updates is not effected by the status of other replicas. Therefore, system performance is persevered. However, strong consistency is not guaranteed due to some failure in a replica which might occur, causing the replica to resynchronize again with other replicas.

The property of mutual consistency in the synchronous or eager replication is guaranteed. Eager replication ensures to deliver the client requests or the transactions to each replica in the same order as they have been received. However, with eager replication the system keeps track of the status of the whole replicas available in the replication group, which affects the performance of the system.

The algorithm presented in (Jiménez et. al, 2002) provides strong consistency for multi-master and partial replication while preserving DBMS autonomy. However, it requires that transactions update a fixed primary copy: each type of transaction is associated with one node so a transaction of that type can only be performed at that node. This is a problem for update intensive applications. Furthermore, the algorithm uses two messages to multicast the transaction reliably and to ensure total ordered multicast. The

cost of these messages is higher than the single FIFO multicast message we use. Moreover, using a logical total order message increases the overhead of physical messages exchanged when increasing the number of nodes.

In our algorithm, all the nodes that hold the resources necessary for the transaction perform it entirely. Any failure in a cluster or even in a node is treated in a way that will not affect system consistency. A failed node is treated as a newly added replica to the replication group, where as a failed cluster is brought offline until its recovered and all nodes within this cluster are totally synchronized through the transaction log. Meanwhile, any client request will be performed from any other available cluster in the system.

5.5 Flexibility

In synchronous replication solutions, adding new node to the system require synchronization with all the nodes in the system so as to propagate all transactions to this newly added node which leads the full system to be brought offline. Therefore, dealing with newly added node to the system in such solution is not flexible because it affects data availability and system performance.

In the proposed architecture, dealing with newly added nodes to the system is as if dealing with a returning node from failure. The architecture inserts this new node to a certain cluster according to a certain criterion as we discussed early. Then this cluster only has to be brought offline while resynchronizing all the nodes together.

5.6 Experimental Study

In this section, we will present a theoretical experimental study about how these transactions are going to propagate following the synchronous, asynchronous and finally the synchronous/asynchronous multi-master replication methods. We will focus on the time needed for full synchronization to take place, the number of connections established during synchronization process, how message exchange takes place, and finally the total amount of the replication system usage resources.

Assume the following three transactions tA, tB, and tC to be propagated to the following six replicas R1, R2, R3, R4, R5, and R6 in the same replication group. We also assume that each transaction needs certain period of time to be successfully propagated to each replica taking into consideration that each replica is always available and having stable network resources and characteristics. Finally, we assume 2, 1, and 3 to be the corresponding values for the time needed for tA, tB, and tC respectively to propagate to each replica.

Following the synchronous approach:

For synchronous replication, the system first checks whether all replicas are online in order for the transactions to be propagated. Then, each transaction *t* is propagated separately to each replica establishing T*R connections for x_t period of time; where T is the total number of transactions needed to be propagated, R is the number of all replicas in the replication group, and x is the period needed for each transaction to be fully propagated on each replica. Referring to our example, there are 3 transactions and 6 replicas, establishing a total of 18 connections.

The total period of time needed for all connections is calculated by $\Sigma^{T}_{t=1}\ x_t$. Therefore, applying the formula we get a total period of: 2 + 1 + 3= 6.

Finally, as it is depicted in figure 5.1, the system will be always busy establishing full connection with each replica until every transaction is fully propagated.

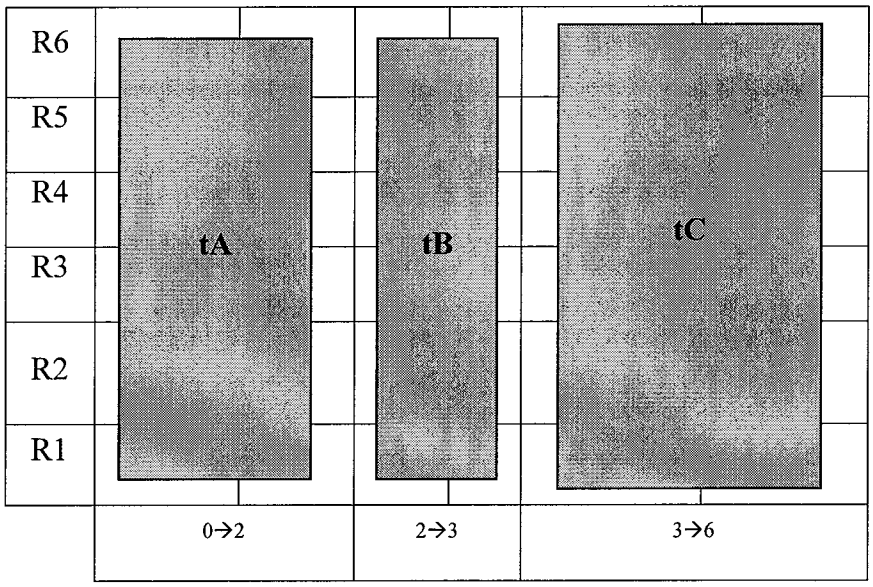

Figure 5.1: System during Synchronous Approach

Following the asynchronous approach:

First, the system checks for the best replica among all the available ones. Then, all transactions are propagated as one big transaction asynchronously to each replica establishing R connections for x_t period of time (starting with the *best node*). Therefore, as there are 6 replicas in our example, the total number of connections to be established is 6.

The total period of time needed for all connections is calculated by $R*\Sigma^{T}_{t=1}\ x_t$. Therefore, applying the formula we get a total period of: 6 *(2 +1 +3) = 36.

Finally, as it is depicted in figure 5.2, the system will be dealing with one replica at a certain period of time; thus, the system will consume less resources as few connections are established at a certain period of time through out the process of data synchronization.

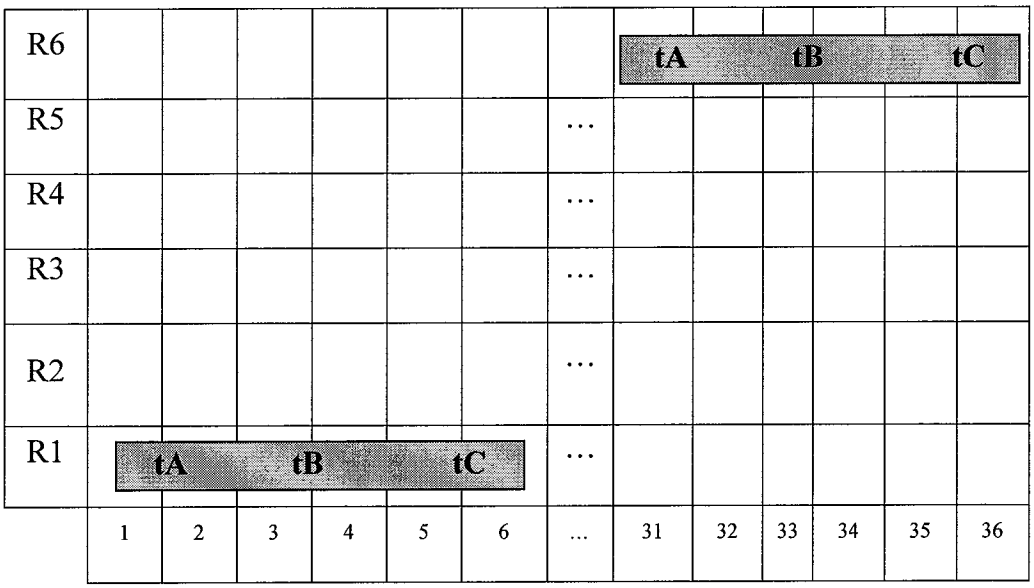

Figure 5.2: System during Asynchronous Approach

Following the synchronous/asynchronous multi-master approach:

For this approach, we assume that the replicas are clustered equally into two clusters C1 and C2. First, the system checks for the best cluster among all available clusters in the replication system. Then, all transactions are propagated asynchronously to each cluster establishing C connections. Each transaction t is then propagated separately to each cluster establishing T*CR connections; where T is the total number of transactions needed to be propagated and CR is the number of clustered replicas in a certain cluster. Therefore, the total number of connections established during this replication method is T*CR+C. Referring to our example, there are 3 transactions and 6 replicas which are clustered into two separate clusters, establishing a total of 3*6+2=20 connections. The total period of time needed for all connections is calculated by

$$\Sigma^{C}_{t=1} \Sigma^{T}_{t=1} x_t.$$

Therefore, in our example, a total of: (2 + 1 + 3) + (2 + 1 + 3) = 12 units of time is required by the system to establish a full synchronization. However, applying the formula for C=1, i.e. for only one cluster (the *best cluster*), will produce a synchronized set of replicas. In other words, only a period of 6 is needed to produce a fully synchronized set or replicas under the same cluster.

Finally, as it is depicted in figure 5.3, the system will be dealing with only one cluster of replicas at a certain period of time; thus, the system will consume less resources as there is a partial number of connections established at a time to synchronize one cluster at a certain period of time. (Total Amount of Resources/Number of Clusters)

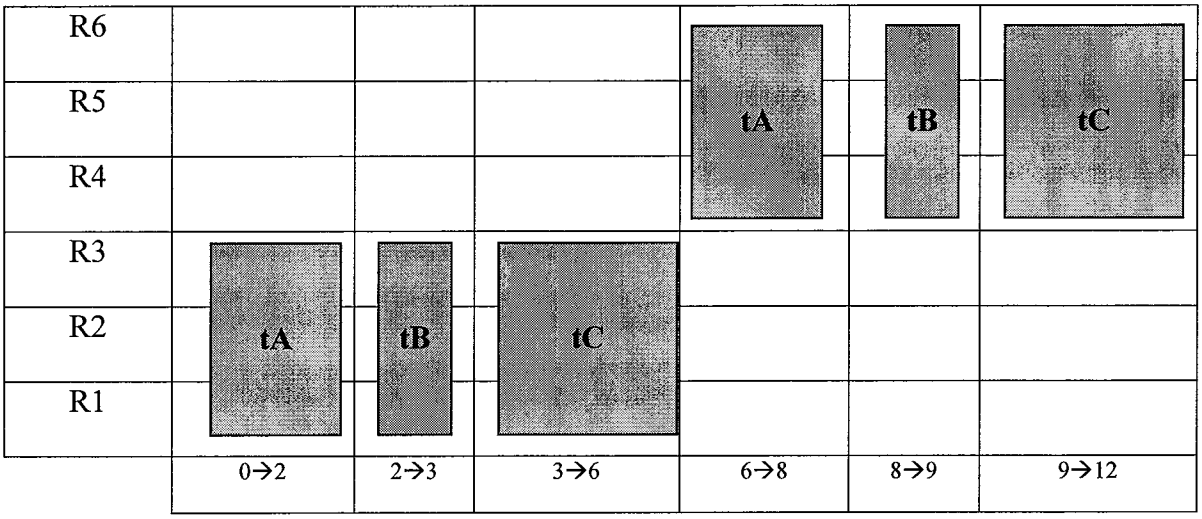

Figure 5.3: System during Synchronous/Asynchronous Approach

Synchronous and Asynchronous VS Synchronous/Asynchronous

Although the synchronous/asynchronous multi-master establishes more connections than the synchronous replication method and although the total period of time for the all connections required to be established in this approach is more than that established in the synchronous approach, the synchronous/asynchronous multi-master replication has fewer message exchange and consumes less network and system resources than the synchronous method. The synchronous replication locks all the available replicas in the system while synchronization, where as in the synchronous/asynchronous replication few replicas are being handled at a time, providing a faster way of replicating data. Unlike the synchronous/asynchronous multi-master approach, the synchronous replication method establishes full connection to all replicas at every time a propagation of an update is needed to be done.

As for the asynchronous approach, although it establishes fewer connections than the synchronous/asynchronous multi-master approach, and despite its low usage to system resources, the total period of time needed for all the established connections during the process of system replication is much more than the period of time needed for connections in the proposed synchronous/asynchronous multi-master architecture. Therefore, unlike the synchronous/asynchronous multi-master replication approach, the asynchronous replication method is not a good option when time sensitive data that needs to be updated more often is being replicated.

Table 5.1 presents a comparison between the synchronous, asynchronous and synchronous/asynchronous multi-master replication methods.

Table 5.1: Comparison among the Various Replication Techniques

	Synchronous	**Asynchronous**	**Synchronous/ Asynchronous**
Total Number of connections	T*R	R	T*R + C
Period for connections	$\Sigma^{T}_{t=1} x_t$	$R*\Sigma^{T}_{t=1} x_t$	$\Sigma^{C}_{t=1} \Sigma^{T}_{t=1} x_t$
Usage of Resources	High	Low	Moderate
Message Exchange	1 Transaction-to- N Replica	N Transactions-to- 1 Replica	1 Transaction-to- N/C Replica

Chapter 6

Conclusion and Future Work

In this context, data replication is used to improve data availability and query load balancing and thus performance. In this work, we have proposed synchronous/asynchronous cluster architecture and a replica control algorithm, which guarantees and provides a flexible and transparent solution that enables users' requests to be managed asynchronously in order to avoid any type of system bottleneck.

We have presented the design and architecture of a new replication technique. The multi-master refresher algorithm prevents conflicts, by exploiting the cluster's high speed network; thus, providing strong consistency, without the constraints of eager replication. Despite that this architecture proposes synchronous approach in layer level two, all its disadvantages are handled by the asynchronous approach that is proposed and followed in the top layer of the architecture. The need for synchronous approach, which guarantees reliability, comes from the fact that there exist some time sensitive data items (i.e. stock quotes) which would be updated more frequently than less time sensitive records (i.e. salary).

In conclusion, the proposed architecture holds the advantages of previously applied methods and techniques such as increasing the performance of the system by lowering the load or usage of network resources, ensuring high availability by keeping synchronized nodes online or active at all time, maintaining high reliability by ensuring that the available nodes are synchronized and up-to-date, and finally being able to handle additional nodes.

Future work includes proposing optimization techniques which would provide the refresher procedure a better and enhanced method to manage transactions on the running queue. Further work also includes defining the best criteria for distributing nodes into clusters so as to maintain load balancing and thus performance.

References

Agrawal D. and El Abbadi A.. (1990). The tree quorum protocol: an efficient approach for managing replicated data. In Proc. of the Int. Conf. on Very Large Data Bases (VLDB), pages 243–254, Brisbane, Australia.

Agrawal D., Alonso G., El Abbadi A., and Stanoi I. (1997). Exploiting atomic broadcast in replicated databases. In Proc. of Europ. Conf. on Parallel Processing (Euro-Par), pages 496–503, Passau, Germany.

Alsberg P. A. and Day J. D.. (1976). A principle for resilient sharing of distributed resources. In Proc. of the Int. Conf. on Software Engineering, pages 562–570, San Francisco, California.

Alonso G.. (1997). Partial database replication and group communication primitives (extended abstract). In *Proceedings of the* 2nd *European Research Seminar on Advances in Distributed Systems (ERSADS' 97)*, pages 171–176, Zinal (Valais, Switzerland).

Anderson T. A., Breitbart Y., Korth H. F., and Wool A.. (1998). Replication, consistency, and practicality: Are these mutually exclusive? In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 484–495, Seattle, Washington.

Bernstein P.A. and Goodman N. (1984). An algorithm for concurrency control and recovery in replicated distributed databases. ACM Transactions on Database Systems, 9(4):596–615.

Bernstein P., Brodie M., Ceri S., DeWitt D., Franklin M., Garcia-Molina H., J. Gray, J. Held, J. Hellerstein, H. V Jagadish, M. Lesk,D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, and J. Ullman. (1998). The Asilomar report on database research. Technical Report MSRTR-98-57, Microsoft Research, One MicrosoftWay, Redmond,WA 98052.

Breitbart Y. and Korth H. F. (1997). Replication and consistency: Being lazy helps sometimes. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 173–184, Tucson, Arizona.

Breitbart, Y., Komondoor, R., Rastogi, R., And Silberschatz, A. (1999). Update propagation protocols for replicated databases. In Proceedings of the 1999 ACM

Conference on Management of Data (SIGMOD '99, Philadelphia, PA, June). ACM Press, New York, NY, 97–108.

Buyya R. (1999). High Performance Cluster Computing. Prentice Hall PTR.

Cap C. H. (1990). Distributed systems with data replication: A non-technical survey. Technical Report ifi-90.11, Department of Computer Science, University of Zürich, 190, Winterthurstraße CH-8057 Zürich, Switzerland.

Carey M. J. and Livny M. (1991). Conflict detection tradeoffs for replicated data. ACM Transactions on Database Systems, 16(4):703–746.

Chundi, P., Rosenkrantz, D. J., And Ravi, S. S. (1996). Deferred updates and data placement in distributed databases. In Proceedings of the 12th IEEE International Conference on Data Engineering (ICDE '97, New Orleans, LA, Feb.). IEEE Press, Piscataway, NJ, 469–476.

Coulon, C., Pacitti, E., Valduriez, P. (2004). Scaling up the Preventive Replication of Autonomous Databases in Cluster Systems, Int. Conf. on High Performance Computing for Computational Science (VecPar -2004), Valencia, Spain.

Eager D. L. and Sevcik K. C.. (1983). Achieving robustness in distributed database systems. ACM Transactions on Database Systems, 8(3):354–381.

El Abbadi A. and Toueg S. (1989). Maintaining availability in partitioned replicated databases. ACM Transactions on Database Systems, 14(2):264–290.

Fox, A. And Brewer, E. A. (1999). Harvest, yield, and scalable tolerant systems. In 6th Workshop on Hot Topics in Operating Systems (HOTOSVI). Rio Rico, AZ. 174–178.

Gançarski S., Naacke H., Pacitti E., Valduriez P. (2002). Load Balancing of Autonomous Applications and Databases in a Cluster System, Parallel Processing with Autonomous Databases in a Cluster System, Int. Conf. of Cooperative Information Systems (CoopIS).

Garmany, J. And Freeman, R. (2004). "Multi-Master Replication Conflict Avoidance and Resolution." Oracle Replicatio, Rampant TechPress, 4th Qtr 2004, pp. 9-15.

Gifford D. K. (1979). Weighted voting for replicated data. In *Proceedings of the Seventh Symposium on Operating System Principles SOSP 7*, pages 150–162, Asilomar Conference Grounds, Pacific Grove CA. ACM, New York.

Goodman N., Skeen D., Chan A., Dayal U., Fox S., and Ries D. R.. (1983). A recovery algorithm for a distributed database system. In Proc. of the ACM Symp. on Principles of Database Systems (PODS), pages 8–15, Atlanta, Georgia.

Gray J., Helland P.,. O'Neil P. E, and Shasha D.. (1996). The dangers of replication and a solution. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 173–182, Montreal, Canada.

Gray J. and Reuter A. (1993). Transaction Processing: Concepts and Techniques. Morgan Kaufmann.

Herlihy, M.P. And Wing, J.M. (1990). "Linearizability: A correctness condition for concurrent objects." ACM Trans Program. Lang. Syst. 12, 3, 463-492.

Holliday J., Steinke R., Agrawal D., and El Abbadi A. (2000). Epidemic quorums for managing replicated data. In Proc. of the IEEE Int. Performance, Computing and Communications Conf. (IPCCC), pages 93–100, Phoenix, Arizona.

Informix. (1998). 4100 Bohannon Drive, Menlo Park, California 94025 USA. *Informix Replication: A High-Performance Solution for Distributingand Sharing Information.*

Jajodia S. (1999). Data replication gaining popularity. *IEEE Concurency*, pages 85–86. Interview of Yuri Breitbart and Hank Korth.

Jajodia S. and Mutchler D.. (1987). Dynamic voting. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 227–238, San Francisco, California.

Jiménez-Peris, R., Patiño-Martínez, M., Kemme, B., Alonso, G. (2002). Improving the Scalability of Fault-Tolerant Database Clusters: Early Results. Int. Conf. on distributed Computing Systems (ICDCS).

Keating, B. (2001). "Challenges Involved in MultiMaster Replication", www.dbspecialists.com.

Kemme, B., Alonso, G. (2000). "A New Approach to Developing and Implementing Eager Database Replication Protocols", ACM Trans and Database Systems, Vol. 25, N# 3, pp. 333-379.

Kumar A. and Segev A. (1993). Cost and availability tradeoffs in replicated concurrency control. ACM Transactions on Database Systems, 18(1):102–131.

Schiper A. and Raynal M. (1996). From group communication to transactions in distributed systems. *Communications of the ACM*, 39(4):84–87.

Maney K. (2002). Microsoft shifts its focus to security. *USA Today.*

Mukkamala, R., Bruell, S. C., and Shultz, R. K. (1988). "Design of Partially Replicated Distributed Database Systems: An Integrated Methodology," ACM.

Neumann P. G. (2002). The risk digest: Forum on risks to the public in computers and related systems. *ACM Committee on Computers and Public Policy*, 21(87).

Oracle (1997). Oracle8(TM) Server Replication, Concepts Manual.

Oracle Corporation. (1998). 500 Oracle Parkway, Redwoord City, CA 94065. *Oracle8iTMAdvanced Replication*. Oracle Technical White Paper.

Pacitti, E., Minet, P., AND Simon, E. (1999). Fast algorithms for maintaining replica consistency in lazy master replicated databases. In Proceedings of the International Conference on Very Large Data Bases (VLDB, Edinburgh, Scotland, Sept.). 126–137.

Pacitti, E., Minet, P., AND Simon, E. (2001). "Replica Consistency in Lazy Master Replicated Databases". Distributed and Parallel Databases, Kluwer Academic, 9(3).

Pacitti, E., Ozsu, M. T., Coulon, C. (2003). "Preventive Multi-master Replication in a Cluster of Autonomous Databases," Euro-Par 2003, LNCS 2790, pp318-327.

Pacitti, E., Ozsu, M. T., Coulon, C., Valduries, P. (2005). "Preventive Replication in a Database Cluster," Distributed and Parallel Databases, 18,223-251.

Paris J. F. and Long D. E. (1988). Efficient dynamic voting algorithms. In Proc. of the Int. Conf. on Data Engineering (ICDE), pages 268–275, Los Angeles, California.

Pedone F., Guerraoui R., and Schiper A. (1998). Exploiting atomic broadcast in replicated databases. In *Proceedings of EuroPar (EuroPar' 98)*.

Rangarajan S., Setia S., and Tripathi S. K. (1995). A fault-tolerant algorithm for replicated data management. IEEE Transactions on Parallel and Distributed Systems, 6(12):1271–1282.

Rys M., Norrie M. C., and Schek H.-J. (1996). Intra-Transaction Parallelism in the Mapping of an Object Model to a Relational Multi-Processor System. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 460–471, Mumbai (Bombay), India.

Saito, Y. (2000). "Optimistic Replication Algorithms".

Saito, Y., Shapiro, M. (2005). "Optimistic Replication", ACM Computing Surveys, Vol. 37, N3, pp. 42-81.

Sivasubramania, S., Szymaniak, M., Pierre, G., and Van Steen, M. (2004). "replication for Web Hosting Systems," ACM Computing Surveys, Vol. 36, N# 3, 99. 291-334.

Son, S. H. (1988). "Replicated Data Management in Distributed Database Systems," SIGMOD Records, vol. 17, N# 4.

Sousa, F., Pedone, R. Oliveira, F Moura. (2001). Partial Replication in the Database State Machine. IEEE Int. Symposium on Network Computing and Applications (NCA).

Sousa, A., Pereira F, Moura F., Oliveira R. (2002). Optimistic total order in wide area networks. Proc. 21st IEEE symposium on reliable distributed systems, pages 190-199.

Stacey D.. (1994). Replication: DB2, Oracle, or Sybase. *Database Programming & Design*, 7(12).

Stanoi I., Agrawal D., and El Abbadi A. (1998). Using broadcast primitives in replicated databases. In Proc. of the Int. Conf. on Distributed Computing Systems (ICDCS), pages 148–155, Amsterdam, The Netherlands.

Stonebraker M. (1979). Concurrency control and consistency of multiple copies of data in distributed Ingres. IEEE Transactions on Software Engineering, 5(3):188–194.

Theel O. and Pagnia H. (1998). Optimal replica control protocols exhibit symmetric operation availabilities. In Proc. of the Int. Symp. on Fault-Tolerant Computing (FTCS), pages 252–261, Munich, Germany.

Thomas R. H. (1979). A majority consensus approach to concurrency control for multiple copy databases. ACM Transactions on Database Systems, 4(2):180–209.

Wolski, A. (2002). "Database replication for the mobile era" ICDE.