Fault Tolerance by Replication of Distributed Database in P2P System using Agent Approach

May Mar Oo, The` The` Soe and Aye Thida

Abstract—The term replication in a distributed database refers to the operation of copying and maintaining database objects in more than one location. There are three types of replication in distributed system that are snapshot, transactional and merge. A partial failure may happen when one component in such system fails. This failure may affect the proper operation of other components while at the same time leaving yet other components totally unaffected. An important goal in such systems design is to construct the system in a way that it can automatically recover from partial failures without seriously affecting the overall performance and continue to operate in an acceptable way while repairs are being made. The technologies, architectures, and methodologies traditionally used to develop distributed applications exhibit a variety of limitations and drawbacks when applied to large scale distributed settings (e.g., the Internet). In particular, they fail in providing the desired degree of configurability, scalability, and customizability. To address these issues, researchers are investigating a variety of innovative approaches. The most promising and intriguing ones are those based on the ability of moving code across the nodes of a network, exploiting the notion of mobile code, the agent toolkit can be chosen as a platform for the replication. So, this paper introduced the topic of agent replication for distributed database and examined the issues associated with using agent replication in a multi-agent system as well as the main issues of agent communication, read/write consistency and state synchronization.

Keywords— Agent, Distributed Database, Fault tolerance, Latency Time, P2P System, Replication

I. Introduction

THE goal of the human-computer interaction community is to make powerful applications easy to use, while retaining their full potential. Distributed database implies that a single application should be able to operate transparently on data that is spread across a variety of different database, managed by a variety of different DBMSs, running on a variety of different machines, supported by a variety of different operating systems, and connected together by a variety of different communication networks-where the term transparently means that the application operates from a logical point of view as if the data were all managed by a single DBMS running on a single machine [7]. The essential issue for distributed database is the replication that increases the locality of reference and fault tolerance i.e., if one of the machines fails, a copy of the data is still available on another machine on the network.

Replication is a cost effective way to increase availability and used for both performance and fault tolerant purposes thereby introducing a constant trade-off between consistency and efficiency. Replication is the most providing way for travelling salespeople and roaming disconnected users and enables mobile users with laptops to be updated with current database information when they connect and to upload data to a server. Data is generated and then replicated.

Building software system composed of mobile agent introduces interesting new concern for software engineering research. An agent is usually defined as an independent software program that runs on behalf of a network user. It can be characterized as having more or less intelligence and it has the ability to learn. Mobile agents add to regular agents the capability of travelling to multiple locations in the network, by saving their state and restoring it in the new host. As they travel, they work on behalf of the user, such as collecting information or delivering requests. This mobility greatly enhances the productivity of each computing element in the network and creates a powerful computing environment [5].

System availability is improved by the replication of data objects in a distributed database system. However, during updates, the complexity of keeping replicas identical arises due to failures of sites and race conditions among conflicting transactions. Fault tolerance and reliability are key issues to be addressed in the design and architecture of these systems. So, we proposed fault tolerance replication system for distributed database in P2P network by using mobile agent approach.

II. Related Work

In previous work we proposed the replication system to be used in P2P environment and we have to add the fault tolerance mechanism. To implement such mechanism, the agent is very flexible approach. Moreover there are many articles that tried to implement the agent in distributed environment and fault tolerance mechanisms. The author presents the performance evaluation of the management agent (MA) containing the adaptive tabu search (ATS) as its search

Manuscript received December 6, 2009: Revised version received February 27, 2010. This work was supported in part by the University of Computer Studies, Mandalay, Myanmar.

May Mar Oo is with the University of Computer Studies, Mandalay, Ph.D Candidate, (phone: + 095-09-2059795; e-mail: maymar80@gmail.com).

The' The' Soe was with University of Computer Studies, Mandalay, Ph.D Candidate, (C/o phone: + 095-02-88723; e-mail: myintmo08@gmail.com).

Aye Thida is with the Research and Development Department, University of Computer Studies, Mandalay, Myanmar. She is an Associate Professor (phone: +095-02-88724; e-mail: ayethidar.royal@gmail.com).

core in [1]. In article [2] they present a comparative analysis for several multi-agents participating in Trading Agents based Competition, Classic [3] is the work proposes a Metascheduler for GRID platforms based on the Interaction Protocols of the Multiagent Systems. These protocols use the paradigm of economic models to define the coordination mechanisms in agent communities. Let look at the article[4],in this work the existing relations between the imprecise computation and the fault tolerant control (FTC) are analyzed.

III. Distributed Database

A distributed database system consists of a collection of sites, connected together via some kind of communication network, in which, each site is a full database system site in its own right, but the sites have agreed to work together so that a user at any site can access data from anywhere in the network exactly as if the data were all stored at the user's can site. Each site is a database system site in its own right. The distributed database system can thus be regarded as a kind of partnership among the individual local DBMS at the individual local sites; a new software component at each site-logically an extension functionality, and it is the combination of this new component together distributed database management system.

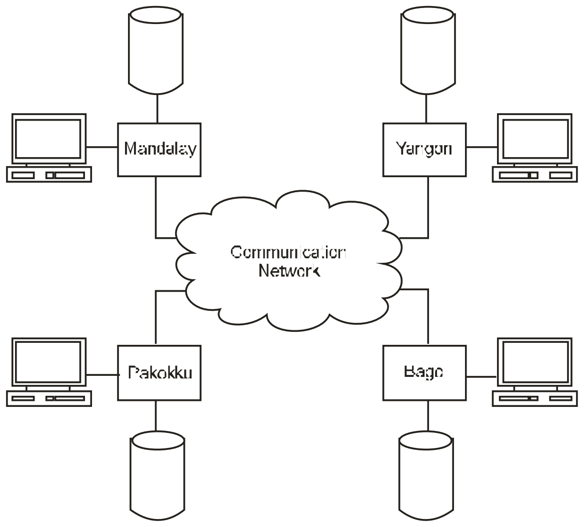


Fig. 1 A Typical Distributed Database System

It has sometimes been assumed that the physical distribution of data is not the most significant issue. The proponents of this view would therefore feel comfortable in labelling as a distributed database two (related) databases that reside in the same computer system. However, the physical distribution of data is very important. It creates problems that are not encountered when the databases reside in the same computer. It simply implies that the communication between them is done over a network instead of through shared memory, with the network as the only shared resource. This fundamental rule lead to 12 objectives, among these objectives data replication supports a system a given account record, may be represented by many distinct copies of the same record object, at many different sites. It can near better performance and better availability. A given replicated object is updated [6].

A. Problems of Distributed Systems

The main problems in distributed database system can be categorized as;

1. Query processing
2. Catalog management
3. Update propagation
4. Recovery control
5. Concurrency control

(1) Query processing

The objective of minimizing network utilization implies that the query optimization process itself need to be distributed, as well as the query execution process.

(2) Catalog management

In a distributed system, the system catalog will include not only the usual catalog data regarding base relvars, views, authorizations, etc; but also all the necessary control information to enable the system to provide the desired location, fragmentation, and replication independence.

(3) Update Propagation

The basic problem with data replication is that an update to any given logical object must be propagated to all stored copies of that object. A difficulty that arises immediately is that some site holding a copy of the object might be unavailable (because of a site or network failure) at the time of the update. The obvious strategy of propagating updates immediately to all copies is thus probably unacceptable because it implies that the update-and therefore the transaction-will fail if any one of those copies is currently unavailable.

(4) Recovery control

Recovery control in distributed systems is typically based on the two-phase commit protocol. It is required in any environment in which a single transaction can interact with several autonomous resource managers.

(5) Concurrency control

Concurrency control in most distributed systems is based on locking, just as it is in most non distributed systems. In a distributed system, however, requests to test, set, and release locks become messages (assuming that the object under consideration is at a remote site) and message mean overhead.

B. Transparent Management of Distributed and Replicated Data

Transparency refers to separation of the higher-level semantics of a system from lower-level implementation issues.

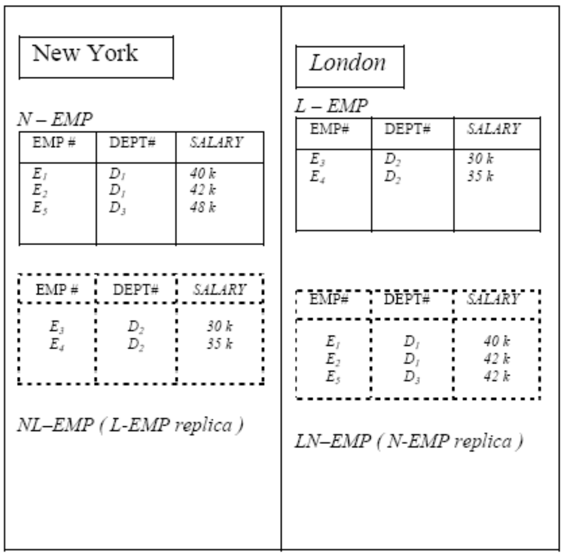
A transparent system hides the implementation details from users. The advantage of a fully transparent DBMS is the high level of support that it provides for the development of complex application. A system supports data replication if a given stored relvar or, more generally, a given fragment of a given stored relvar can be represented by many distinct copies or replicas, stored at many distinct sites. Fully transparent access means that the users can still pose the query as specified in local, without paying attention to the fragmentation, location or replication of data and let the system worry about resolving these issues.

For performance, reliability and availability reasons, it is usually desirable to be able to distribute data in a replicated fashion across the machines on network. Such replication helps performance since diverse and conflicting user requirements can be more easily accommodate. Assuming that data is replicated, the issue related to transparency that needs to be addressed is whether the users should be aware of the existence of copies or whether the system should handle the management of copies and the user should act as if there is a single copy of the data. On the other hand, doing so inevitably results in the loss of some flexibility. It is not the system that decides whether or not to have copies and how many copies to have, but the user application. Any change in these decisions because of various considerations definitely affects the user application and therefore reduces data independence considerably. Given these considerations, it is desirable that replication transparency be provided as a standard feature of DBMS. It is important that replication transparency refers only to the existence of replicas, not to their actual location.

IV. Replication

Replication is the process of copying data from a data store or file system to multiple computers to synchronize the data. Database replication is becoming more important role for database applications. Replicated data are becoming more and more of interest lately. Replication is a cost effective way to increase availability and used for both performance and fault tolerant purposes thereby introducing a constant tradeoff between consistency and efficiency [12]. Active directory provides multi-master replications of the directory between domain controllers within a given domain. The replica of the directory on cash domain controller is writable. This allows updates to be applied to any replica of a given domain. The replication service automatically copies the changes from a given replica to all other replicas.

Replication is desirable for at least, two reasons: first, it can mean better performance and second, it can also mean better availability. When object is updated, all copies of that object must be updated which called the update propagation problem. The problem with data replication is that an update to any given logical object must be propagated to all stored copies of that object. Some site holding a copy of the object might be unavailable at the time of update. All copies are probably unacceptable. This problem works as primary copy scheme. It does represent a violation of local autonomy objective because a transaction might now fail. Primary copy of some object is unavailable even if a local copy is available. All update propagation must be completed [7].


New York

N – EMP

EMP #	DEPT#	SALARY
E_1	D_1	40 k
E_2	D_1	42 k
E_5	D_3	48 k

EMP #	DEPT#	SALARY
E_3	D_2	30 k
E_4	D_2	35 k

NL–EMP (L-EMP replica)

London

L – EMP

EMP#	DEPT#	SALARY
E_3	D_2	30 k
E_4	D_2	35 k

EMP#	DEPT#	SALARY
E_1	D_1	40 k
E_2	D_1	42 k
E_5	D_3	42 k

LN–EMP (N-EMP replica)

Fig.2 An example of replication

In a totally isolated database system, individual systems are DBMS or the way to communicate with them. In each systems, the processing of user transactions that access multiple databases is especially difficult since there is no control over the execution individual DBMS and the limitations of client server systems become evident in an Internet scale distributed environment. Distributed Systems in replication offer an alternative to traditional client/server systems. To provide that, our proposed system is implemented as a distributed system in replication for the performance and reliability of the system, we use the nature of distributed and parallel DBMS. Our purposed aims at allowing transparent access to distributed database on P2P network based or replica. Replication is one of the oldest and most important topics in the overall area of distributed systems. It is a process of copying/moving data between databases on the same or different servers.

A. Basic Concepts of Replication

Replication uses the following three servers [5].

(1) Publisher
(2) Distributor
(3) Subscriber

The basic problem with data replication is that an update to any given logical object must be propagated to all stored copies of that object. A difficulty that arises immediately is that some sites holding a copy of the object might be unavailable (because of a site or network failure) at the time of the update. The obvious strategy of propagating updates immediately to all copies is thus probably unacceptable, because it implies that the update and therefore the transaction will fail if any one of those copies is currently unavailable. In a sense, in fact, data is less available under this strategy than it would be in the non-replicated case.

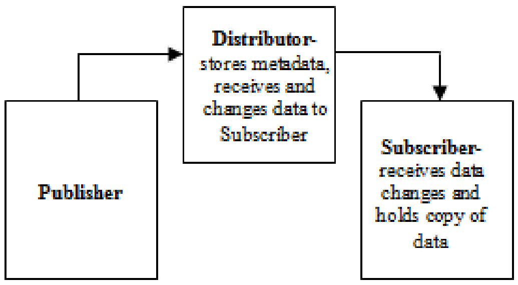

Distributor-
stores metadata, receives and changes data to Subscriber
Publisher
Subscriber-
receives data changes and holds copy of data

Fig.3 Replication process

B. Three Types of Replication

There are three types of replication in distributed database as follows:

(1) Snapshot Replication
(2) Transactional Replication
(3) Merge Replication

(1) Snapshot Replication

Snapshot replication [8] is the simplest form of replication and it simply takes a "snapshot" of the data on one server and moves that data to another server (or another database on the same server). It distributes data exactly as it appears at a specific moment in time and does not monitor for updates to the data. It is the best used method of replicating data that change infrequently or where the most up-to-date values are not requirement.

When synchronization occurs, the entire snapshot is generated and sent to the subscribers. After the initial synchronization snapshot, replication can refresh data in published tables periodically based on the schedule you specify. Although snapshot replication is the easiest type to set up and maintain, it requires copying all data each time a table is refreshed.

(2) Transactional Replication

Transactional replication [8] involves copying data from the publisher to the subscriber(s) once and then delivering transactions to the subscriber(s) as they occur on the publisher. It is also known as dynamic replication and typically used in server-to-server environments. The application requires low latency between the time changes are made at the publisher and the changes arrive at the subscriber. An initial snapshot of data is applied to subscriber, and when data modifications are made at the publisher, the individual transactions are captured and propagated to subscriber. By default, subscribers to transactional publication should be treated as read only, because changes are not propagated back to the publisher. Transactional replication performance tuning and optimization examined performance in transactional replications and based on the results of tests conducted using a variety of hardware configurations and replication environments [13].

(3) Merge Replication

Merge replication [8] is the process of distributing data from publisher to subscriber, allowing the publisher and subscriber to make update data while they are connected or disconnected, and then merging the updates between sites when they are connected. It is typically used in server-to-client environments and appropriate where multiple subscribers might update the same data at various times and propagate those changes to the publisher and to other subscribers that need to receive data, make changes offline, and later synchronize changes with the publisher and other subscribers, each subscriber requires a different partition of data, conflicts might occur and, when they do, the users need the ability to detect and resolve them.

V. Agents

An agent is a computation that may hop from site to site over the network [9]. We review the concepts of agents, agent servers, suitcases, and briefings. A suitcase is a piece of data that an agent carries with it as it moves from site to site. It contains the long-term memory of the agent. It may include a list of sites to visit, the tasks to perform at each site, and the results of performing those tasks. A briefing is data that an agent receives at each site, as it enters the site. It may include advice for the agent (e.g. “too busy now, try this other site”), and any site dependent data such as local file systems and databases. An agent server, for a given site, is a program that accepts code over the network, executes the code, and provides it with a local briefing.

A hop instruction is used by agents to move from one site to the next. This instruction has as parameters an agent server, the code of an agent, and a suitcase. The agent and the suitcase are sent to the agent server for execution. Finally, an agent is a user-defined piece of code parameterized by a suitcase and a briefing. All the data needs of the agents should be satisfied by what it finds in either the suitcase or the briefing parameters. At each site, the agent inspects the briefing and the suitcase to decide what to do. After performing some tasks, it typically executes a hop instruction to move to the next site. If an agent has a user interface, it takes a snapshot of the interface, stores it in the suitcase during the hop, and rebuilds the interface from the snapshot at the destination.

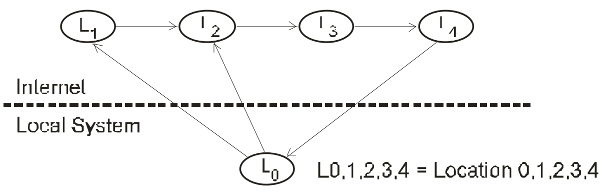


Fig.4 The Scenario for Agent transaction

Agents in computer systems are often mobile. The ability of migration provides mobile agents a means to overcome the high latency or limited bandwidth problem of traditional client-server interactions by moving their computations to required resources or services. Migration also provides a means to protect data. Agents may need to migrate to given locations to view and process specific data. A distinction can be made between migrations in which the execution state is migrated along with the unit of computation or not. Systems providing the former option are said to support strong mobility, as opposed to systems that discard the execution state across migration, and are hence said to provide weak mobility. In systems supporting strong mobility, migration is completely transparent to the migrated program, whereas with weak mobility, extra programming is required in order to save part of the execution state. Strong mobility requires that the entire state of the agent, including its execution stack and program counter, is saved before the agent is migrated to its new location. Strong mobility is a complicated task to realize, and typical implementations of this functionality in multi-agent platforms provide platform specific solutions. As a consequence, interoperability between heterogeneous multi-agent systems is difficult, if not impossible, to realize. Many of the multi-agent platforms support weak mobility.

Most of the agent systems are implemented on top of the Java Virtual Machine (JVM), which provides with object serialization basic mechanisms to implement weak mobility. The JVM does not provide mechanisms to deal with the execution state.

A. Java Aglets

The Java Aglets API (J-A API) [9], developed by IBM Tokyo Research Laboratory in Japan, extends Java with support for weak mobility. The context of an aglet provides a set of basic services, e.g., retrieval of the list of aglets currently contained in that context or creation of new aglets within the context. Java Aglets provides two migration primitives: dispatch is the primitive that performs code shipping of stand-alone code (the code segment of the aglet) to the context specified as parameter. The mechanism is asynchronous and immediate. The symmetrical primitive retract performs code fetching of standalone code, and is used to force an aglet to come back to the context where retract is executed, with a synchronous, immediate mechanism. In both cases, the aglet is re-executed from scratch after migration, although it retains the value of its object attributes which are used to provide an initial state for its computation. The attribute values may contain references to resources, which are always managed by copy. Finally, being based on Java, the Aglets API supports Java mechanisms as well.

B. Advantages of Mobile Agents

Mobile agents introduce some advantages to the usual programming models:

• the construction of economic high quality and high performance applications. Applications using mobile agents use the network transparently, taking full advantage of the local resources. The data processing is done at the source rather than remotely fetched.

• the efficient and economic use of low-bandwidth, high-latency, error prone communications. The agent network uses a store and forward mechanism to transfer agents between nodes. This mechanism is enhanced with queuing and persistent checkpoints allowing mobile agents to use communications of mobile devices without degradation in reliability and response.

• the use of low-cost portable personal communication devices. Mobile devices are one of the areas of growth in the computer industry. Everything from laptops to palmtops, cellular phones, and electronic books will access the Internet services to deliver user tasks. Nonetheless these devices will have unreliable low-bandwidth, high-latency network connections provided by telephone lines or even wireless. Mobile agents will be the ideal to accomplish the tasks performed by these devices.

• the enhance of secure communications on public networks. As they travel, mobile agents carry with them user credentials. These are authenticated during execution in every point in the network. To enhance security, agents and their data travel along the network fully encrypted.

• the alternative to the client/server paradigm. With mobile agents the request/response architecture of the client/server paradigm is avoided, because the flow of control moves across the network. Every node becomes a server and the agent moves from location to location to find the services it needs at each point of its execution. The complicated issue of exponential scaling of servers and connections required between multiple servers becomes a simple capability issue. The relationship between servers and users is now coded in each agent, instead of being scattered along clients and servers. There is also the advantage of the hold time for connections being reduced to the time required to move the agent in or out of the host. As the agent carries its own credentials, the connection is only a move operation, abolishing authentication and spoofing. The agent moves only once, no requests flow across the connection. This allows optimization at several levels.

• the software distribution on demand. The wide use of code on demand systems such as Java Applets, Java servlets and Active X has opened the door to an installation alternative: software distribution on demand, which can transport code and install packages automatically. This kind of distribution is a potential advantage for every mobile code system, therefore to mobile agents.

• Asynchronous tasks. Agents can be used to perform a set of asynchronous tasks in the network. The client part of the application can be transferred from the device to the network. Once the transfer is done, the device can be disconnected and reconnected when the result is available. This is very useful in mobile devices where it is almost impossible to maintain a connection for days, weeks or longer. It is important that the agent's system is fault-tolerant, carrying his work independently of communication and host failures. It must also be aware of the exactly-once semantic, not performing a task more than once, even if it was interrupted by a failure. Some kind of check-pointing must be maintained.

C.A Problem - Specific Fault - Tolerance Mechanism for Asynchronous, Distributed Systems

The theory on fault-tolerant mechanism for distributed systems is based on different kind of networks, such as LAN, WAN. Since these networks are providing various services over the network, it makes communications between the users entrusted. So, the algorithm, which provides reliable services and scalability of the resources, is needed to be designed. In fact, scalability is achieved by a fully decentralized algorithm, in which the dynamically available resources are managed through a membership protocol [10]. On other side, fault tolerance is assured in meaning of that the loss of up to all but one resource will not affect the quality of the solution. There are some approaches, such as synchronous or asynchronous, work sharing, and information sharing mechanisms. Synchronous design underlying that all processes will wait each other to complete, and not to wait in the asynchronous algorithm. Work sharing technique allows monitoring all processes and assigning any available slots, so these processes may work in parallel. Information sharing mechanism based on storing all information about all known sharing solutions and its decision is the best known solution for this kind of problem.

The internet connected computers can implement such algorithm to give all resources for distributed systems, such as [10]: scalability, dynamic availability, unreliability, communication characteristics, heterogeneity, and lack of centralized control [10]. They describe dynamic design which was chosen by algorithm designers. As usual dynamic methods provide better performance and consistency, and all processes update data with better solutions for future reference and faster decision making result. However, to make more reliable some extensions were invented, such as a group membership protocol to allow dynamic variation in the number and components of resources and a fault tolerance mechanism. The group membership protocol is for collecting and updating information about which resources participate in the computation at any given time [10].

This works really simple. When new process or computer comes to the assigned group, it sends request to join the group to known server, which controls this group and knows if anyone from this group is present. If not, the group is initialized, and on other hand if group is active, the server just simply joins this member to this group. On other side when process lives the group by any causes, such as self termination or even failure membership server is aware of these kind processes. As a result, the system always knows who is where and what status of the systems' components are. Moreover, the server is able to log all activities of all members, and make sure that process is not in timeout, due to some inactivity period of it. The fault-tolerant mechanism does not capture failures of computers and restore their data, but rather focuses on detecting missing results [10]. This can be obtained by getting placed problem on the particular place as a node in the tree, and by calculating code for each location in the tree. As a result, each node can be identified by unique code.

D.Fault-tolerance replicating by agents

Replication of services leads to increase level of complication of the system. The transparent agent replication is the technique in which the replicates of agents appear and act as one entity thus avoiding an increase in system complexity and minimizing additional system loads. Another point is inter-agent communication, read/write consistency, resource locking, resource synthesis and state synchronization [10]. Multi-agent systems (MASs) appropriate solution for development difficult, distributed systems, but they are still vulnerable to any kinds of faults that any distributed system may have. The modular nature of a MAS gives it a certain level of inherent fault tolerance, however the non-deterministic nature of the agents, the dynamic environment, the inter connectedness of the agents and the lack of any central control point make it impossible to foresee possible fault states and make fault handling behaviour unpredictable[10]. Since that, entire system may crash just because only one agent had a small error. The authors made their assumptions based on the open environment system, where agents are not reliable and might not be fault-tolerant. There are four essential characteristics of a multi-agent system: a MAS is composed of autonomous software agents, a MAS has no single point of control, a MAS interacts with a dynamic environment, and the agents within a MAS are social (agents communicate and interact with each other and may form relationships) [10]. Moreover, there exist various fault types, which can bring wide system crash. These faults can come from program defects, sudden changes, processor and communication fault, as well as emerging actions, which are not predicted. In addition, there are many techniques to have replication agents, and they either agent centric or system-centric. The major difference between them is that system-centric is more complex structure, but it can catch system-wide faults. The mixture of these two types of agents is classified as a Dependable Mobile Agent System. However, both of the techniques uses agent replication aspects, and the redundant replicates and any proxies are external to original agent, using a system centric approach, but individual agents must have the capability to utilize replication [10].

Agent replication is the act of creating one or more duplicates of one or more agents in a multi-agent system [10]. Each of these agents may perform the same task as the original agent. Duplicate agents form replicate group and each member is named as replicates. As soon as a replicate group is formed and it is viewable by all members of the MAS, there are several ways that agents can interact with it:

• An agent can send requests to each replicate in turn until it receives an appropriate reply.

• An agent can send requests to all replicates and select one, or synthesize the replies that it receives.

• An agent can pick one of the replicates based on particular criteria (speed, reliability, etc.) and interact only with that agent [10].

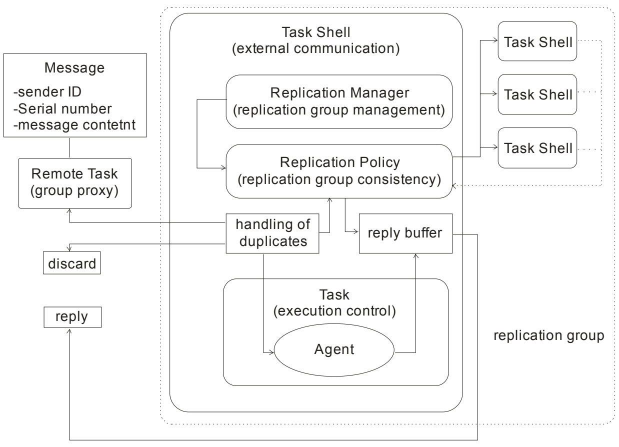


Fig.5 Replication management using Agent

The transparent replication uses proxy to communicate between replicas and clients. So, proxies provide two important functions: they make the replicate group appear to be a single entity and they control execution and state management of a replicate group [10]. In details, proxy is providing communication between replicates and other agents in the MAS [10]. In this case, proxy becomes single point of failure, which requires backups of itself and if one proxy goes down another and another can replace main proxy's functionalities. This brings another overhead for this solution, but it pay off by improving reliability of the system. Another proxy to exchange data between replicas, and this proxy make sure that all members receive equal data. There are some downsides of this part such as it becomes single point of failure solution. Also, it may cause time delay between data received and data backed up to the replicates. The other role that proxies can fill is management of the replicate group. Through various replicate group management policies, some of the replication issues can be dealt with. Three replication management policies are identified: hot-standby, cold-standby, and active [10]. Depending on policy system besides which proxy will start after current working proxy goes down. As well as proxies may improve performance management of the replicate group [10].

VI. Latency Time

This section describes latency time in replication. In a computer system, latency is often used to mean any delay or waiting that increases real or perceived response time beyond the response time desired. Latency is the delay that occurs after a send operation is executed before data starts to arrive at the destination computer [11]. Specific contributors to computer latency include mismatches in data speed between the microprocessor and input/output devices and inadequate data buffers. Within a computer, latency can be removed or "hidden" by such techniques as perfecting (anticipating the need for data input requests) and multithreading, or using parallelism across multiple execution threads.

The latency assumption seems to be that data should be transmitted instantly between one point and another (that is, with no delay at all). The contributors to network latency include:

Propagation: This is simply the time it takes for a packet to travel between one place and another at the speed of light.

Transmission: The medium itself (whether optical fiber, wireless, or some other) introduces some delay. The size of the packet introduces delay in a round trip since a larger packet will take longer to receive and return than a short one.

Router and other processing: Each gateway node takes time to examine and possibly change the header in a packet.

Other computer and storage delays: Within networks at each end of the journey, a packet may be subject to storage and hard disk access delays at intermediate devices such as switches and bridges.

In our proposed system, we can compute the latency time by the following equation:

$$LT = R_{bt} - S_{at} \qquad (1)$$

where,

LT = Latency time between different machines

R_{bt} = Receiving time at destination point

$S_{a}t$ = Sending time at source point

As mention, we compute and compare the latency time for the three types of operations on transactional replication.

VII. Overview of Proposed System

In this section, we present the overview of the proposed system. Before describing the proposed system, there are some assumptions. This system is only for homogeneous databases. Different types of representative environments for replication are LAN, P2P systems, Grids and so on. Among them, this proposed system uses P2P system. P2P system provides a decentralized approach to solve classical problems in replication and caching. They offer many advantages over traditional centralized approaches: e.g., organic scaling, no costly infrastructure, and fault tolerance. P2P systems need to deal with data location, data integration, data querying, and data consistency issues.

Replication in P2P system is a key effectiveness of distributed system in that can provide enhanced performance, high availability and fault tolerance. It can provide all transactions such as insert, update and delete with data synchronization. Synchronization refers to the propagation of data changes between publisher and subscriber. Therefore, synchronization is the crucial role for replication. There are three types of replication such as snapshot, transactional and merge.

Among them, this system proposed the transactional replication. Transactional replication allows data modifications to be propagated incrementally between different locations in a distributed environment. It is the most efficient and flexible synchronization model in terms of response time and customizability.

We choose the transactional replication because it can transfer only the changed data (instead of sending the completed data every time) to the subscriber with minimal latency time than others (such as snapshot and merge replication). In this transactional replication, data alteration and custom processing can be easily implemented using custom replication stored procedures, while transferring the data.

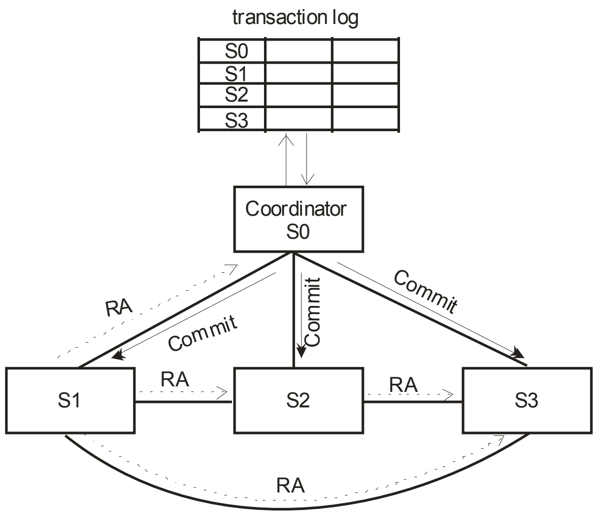


Fig.6 Overall Architecture of the system

Firstly, users need to choose whether they update the data or enquire the replica data from remote site. If they want to read the data, they can read the data that they desired via the proxy. If they want to update the data, data may be updated and then they may also be propagated by the replica agent. The agent may also be replicated for fault tolerance purpose. The commit will be performed by the selected coordinator. The coordinator also maintains the states of each replica and it can rollback to previous state if any fault occurs. Figure 6 shows the architecture of the system.

VIII. Experimental Results

In this section, we present and compare the latency time of insert, update and delete operations for previous transactional replication system and our new fault tolerance replication system.

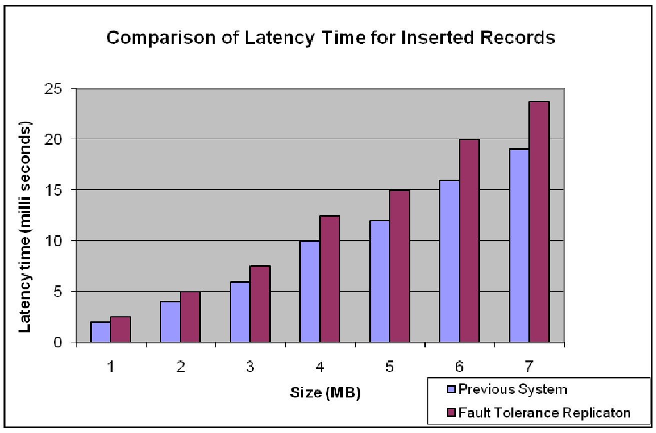


Fig.7 Latency time for inserted records

Figure 7 shows comparison of the latency time for inserted records. The horizontal axis represented file size in megabytes. The vertical axis represented latency time in milliseconds when transferring data from one machine to another. In previous system, latency time is absolutely depending on the file size. The bigger the file sizes, the higher the latency time, but in new system it also tried to cover the fault tolerance purpose, latency is a little greater.

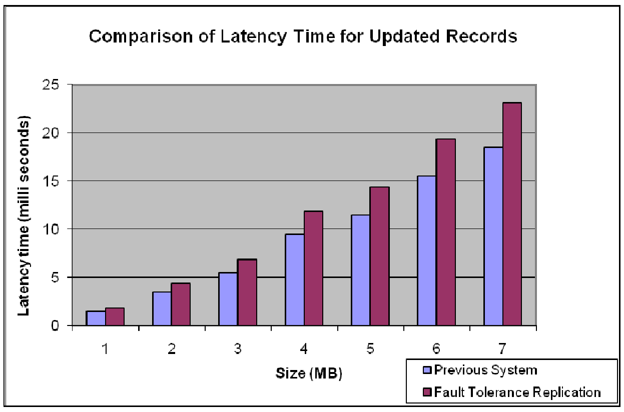


Fig.8 Latency time for updated records

The evaluation performance of updated records depending to their file size and the latency time is described in Figure 8. According to the above experimental results, the latency time is gradually increased when 1MB of file size is reached 1,500 milliseconds to 7 MB of file size is reached 18,500 milliseconds. The more the file size, the more the latency time is.

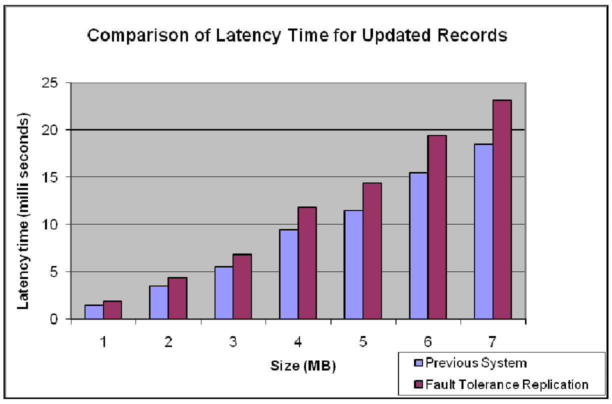


Fig.9 Latency time for deleted records

Similarly, the horizontal axis represented measurement of file size in megabytes as shown in Figure 9. The vertical axis represented latency time when transferring data from one machine to another. Latency time is absolutely depending on the file size. The bigger the file sizes, the higher the latency time.

As mentioned above, the latency time of three operations for both systems are slightly difference. In previous system when transferring file size of 1MB, Insert operation takes 2,000 milliseconds, Update operation takes 1,500 milliseconds and Delete operation takes 1,000 milliseconds but in new system they are increase proportionally. Although we can see the more file size, the more latency time, it is not clearly difference latencies among them. Therefore, we can show the overall average time is nearly the same according to our experimental results.

IX. Conclusion

In previous paper, we present the transactional replication with latency time. But P2P network is a dynamic self-organization network, in which peer can freely join or leave, so there will lost a lot of important data when some important nodes fail, and there exists load imbalance of node in the P2P network. So, we added the previous system with some mechanism that support for fault tolerance replication. At that point, we used the Mobile agent to control the transactions. The proposed system also computes latency time of update, delete and insert operations depend on the file sizes. The bigger the file sizes, the higher the latency time. According to the experimental results the average file sizes of three transactions are nearly the same. There is a little difference in latency when transferring data from one machine to another. So, this proposed system can transfer data between publisher and subscriber with minimal latency. This system is suitable for distributed system when distributing data in a replicated fashion across the machine on the network with optimal results. It also improves the performance, availability and reliability for distributed system. It can supplement disaster-recovery plans by duplicating the data from a local database server to remote database server.

Acknowledgment

First of all,we would like to thank our supervisor Associate Professor Dr.Aye Thida,University of Computer Studies Mandalay,Myanmar for accepting us into the research and guiding us along the way with patience,offering bright suggestions and sound advice.We also thankful to Assistant Lecturer U Thein Than Thwin,University of Computer Studies,Mandalay,Myanmar for overall support throughtout the period of the study.Finally,we also thank all my friends for their motivating discussion about research and help.

References

[1] J. Kluabwang, D. Puangdownreong, S. Sujitjorn, Performance Assessment of Search Management Agent under Asymmetrical Problems and Control Design Applications, School of Electrical Engineering, Institute of Engineering Suranaree University of Technology, WSEAS Transactions on Computers,ISSN: 1109-2750, Issue 4, Volume 8, April 2009.
[2] A. Rios-Bolivar,L. Parraguez, F. Hidrobo, M. Heraoui, J. Anato, F. Rivas, An Imprecise Computation Framework for Fault Tolerant Control Design,WSEAS Transactions on Computers, ISSN: 1109-2750 ,Issue 7, Volume 8, July 2009.
[3] D. Mancas, S. Udristoiu, Ecaterina - Irina Manole, B. Lapadat, A Comparison of Multi-Agents Competing for Trading Agents Competition,WSEAS Transactions on Computers, ISSN: 1109-2750, Issue 12, Volume 7, December 2008.
[4] J. Aguilar, R. umoza, Cemisid, A Multiagent Grid Metascheduler, Facultad de Ingeniería, Universidad de los Andes, Mérida, Venezuela, WSEAS Transactions on Computers,ISSN:1109-2750,Issue 8, Volume 8, August 2009.
[5] H. Paulino, A Mobile Agent Systems' Overview, Departamento de Inform´atica, Faculdade de Ciˆencias e Tecnologia, Universidade Nova de lisboa, February, 2002.
[6] C. J. Date, An Introduction to Database Systems 7th edition, May 2000.
[7] Principles of Distributed Database Systems, second edition, Prentice Hall, Upper Saddle River, New Jersey 07458.
[8] http://en.wikipedia.org/wiki/Distributed_ data- base
[9] A.I. Wang,A.A.Hanssen,B.S.Nymoen,Design Principles for a Mobile Multi-Agent Architecture for Cooperative Software Engineering , Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), N-7491 Trondheim, Norway.
[10] Y. Gershteyn,Fault Tolerance in Distributed Systems, Department of Computer Science, Rochester Institute of Technology, Rochester, NY, USA, February 5, 2003.
[11] G. Coulouris, J. Dollimore and T. Kindberg, Distributed systems concepts and design, Third Edition.
[12] V.Martins, E. Pacitti and P. Valduriez, Survey of data replication in P2P systems, 2006.
[13] B. Newman, X. Schildwachter and G. Yvkoff, Transactional replication performance tuning and optimization, 2001.

May Mar Oo,Ph.D Candidate,University Of Computer Studies,Mandalay,Myanmar.Native town is Mandalay,Myanmar and Date Of Birth is November 20,1980. B.C.Sc degree is hold in 2000,B.C.Sc(Hons:) degree is hold in 2001 and M.C.Sc degree is hold in 2002 at University Of Computer Studies,Mandalay,Myanmar.Her major field is Computer Science.

She worked as Tutor at Mandalay University,Myanmar in 2005 and Hpa-an University,Myanmar in 2009.Currently,she works as Assistant Lecture at University Of Computer Studies,Mandalay,Myanmar.She accepted the journal of NAUN Conference in December 8, 2009.

The' The' Soe,Ph.D Candiate,University Of Computer Studies,Mandalay,Myanmar.Native town is Lashio,Myanmar and Date Of Birth is December 7,1979. B.C.Sc degree is hold in 2007,B.C.Sc(Hons:)

degree is hold in 2008 and M.C.Sc degree is hold in 2009 at University Of Computer Studies,Mandalay,Myanmar.Her major field is Computer Science.

She worked as Tutor at Mandalay University in 2008 and She works as Assistant Lecture at University Of Computer Studies,Mandalay,Myanmar.She accepted the journal of NAUN Conference in December 8, 2009.

Aye Thida, Ph.D, Associate Professor, University Of Computer Studies,Mandalay,Myanmar.Native town is Monywar, Myanmar and Date Of Birth is April 7,1972. D.C.Sc degree is hold in 2000,M.I.Sc degree is hold in 2001 and Ph.D degree is hold in 2004 at University Of Computer Studies,Yangon,Myanmar.Her major field is Computer Science.

She worked as Tutor at Mandalay University,Myanmar in 2002 and she also worked as Headmistress at Loikaw University,Myanmar in 2007.Currently she works as Associate Professor at University Of Computer Studies,Mandalay,Myanmar.She accepted the journal of NAUN Conference in December 8, 2009.

Fault Tolerance by Replication of Distributed Database in p2p system using Agent Approach.Ms.Oo,Ms. Soe and Dr. Thida.They were researchers with University of Computer Studies,Mandalay,Myanmar.