Improving the Fault Tolerance for Distributed Database Using Check Pointing and Agent System

May Mar Oo
University of Computer Studies, Mandalay
maymar80@gmail.com

Abstract

Fault tolerance is very essential for distributed database in a client server environment. In that distributed database system, a partial failure may happen when one component fail in the system. This failure may affect the proper operation of other components totally. The major purpose of the system is to continue for operating in an acceptable way while repairs are being made and to automatically recover from partial failures without seriously affecting the overall performance. Check pointing is a technique for inserting fault tolerance into computing systems. It basically consists of storing a snapshot of the current application state, and later on, uses it for restarting the execution in case of failure. Data replication is a distributed database solution used to improve performance and availability for several types of distributed applications such as data warehousing and on-line financial transactions. Replication techniques also address fault-tolerance activities of masking failures, and reconfiguring the system in response. So, this system improves the fault tolerance for distributed database with minimal recovery time and we analyze the latency time of insert, delete and update operations on the distributed database.

1. Introduction

Fault tolerance can allow processes executing in a computer system to survive failures within the system. Data replication is often used in distributed database applications to improve data availability and performance [1]. Replicated data must be periodically refreshed using update propagation strategies. Replication techniques also address fault-tolerance activities of masking failures, and reconfiguring the system in response. Databases are fully replicated in order to get two complementary features: performance improvement and high availability. Performance can be improved when a database is replicated since each replica can serve read-only access without requiring any coordination with the rest of replicas. Thus, when most of the application accesses to the data are read-only, they can be served locally and without preventing accesses in the same or other replicas. Moreover, with a careful management, the failure of one or more replicas does not compromise the availability of the database. Database replication management was decomposed into two tasks: concurrency control and replica control, usually solved by different protocols. The solutions of the no replicated domain were evolved into distributed concurrency control protocols [2].

A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network. The database is physically distributed across the data sites by fragmenting and replicating the data. In distributed systems, information processing is distributed over several computers rather than confined to a single machine. Distributed system characteristics are resource sharing, openess, concurrency, scability and fault tolerance. A difficulty that arises immediately is that some

site holding a copy of the object might be unavailable 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.

Redundancy is used to mask the fault of the proposed system. So, we can replicate the process and organize them into a group to replace a single process with a group. In effect, when the primary crashes, the backups execute some election algorithm to choose a new primary. 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. Fault tolerance and reliability are key issues to be addressed in the design and architecture of these systems. Coordinated check pointing has been introduced by Candy and Lamport. This technique requires that at least one process send a marker to notify the other ones to take a snapshot of their local states and then form a global checkpoint. The global state obtained from a coordinated checkpoint is coherent, allowing the system to recover from the last full completed checkpoint wave. It does not generate any orphan processes, or domino effects, but all the compute nodes must rollback to a previous state in case of any failure. The recovery process is straightforward, and simple garbage collection reduces the size needed to store the checkpoints.

2. Related Works

A.Fedoruk et.al presented the improving fault tolerance by replicating agents. They examined the use of transparent agent replication as a 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 [3]. S.Hagg presented a sentinel approach to fault handling in multi-agent systems. They presented the sentinels to guard certain functionality and to protect from undesired states. They introduced the sentinels form a control structure to the MAS, and through the semantic addressing scheme they can monitor communication, build models of other agents, and intervene according to given guidelines [4]. O.Marin et.al presented how to bring flexibility to fault-tolerant systems. They proposed a framework for building applications that provide adaptive fault tolerance, and put forward the promising results obtained when testing the implementation of this framework. In this paper, we analyze the fault tolerance for distributed database using check pointing and agent approach.

3. Distributed Database

The database is physically distributed across the data sites by fragmenting and replicating the data. In a distributed DBMS, data and the applications that access that data can be localized at the same site, eliminating the need for remote data access that is typical of teleprocessing-based timesharing systems. Furthermore, since each site handles fewer applications and a smaller portion of the database, contention for resources and for data access can be reduced. Finally, the inherent parallelism of distributed systems provides the possibility of inter-query parallelism and intra query parallelism. If the user access to the distributed database consists only of querying then provision of inter-query and intra-query parallelism would imply that as much of the database as possible should be replicated [5]. However, since most database accesses are not read-only, the mixing of read and update operations requires support for distributed transactions.

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[6].

4. Failure Model of the System

There are many different kinds of failure models in distributed database systems. For example, Crash failure, Omission failure, Timing failure, Response failure, Arbitrary failure. In this system, we assume that there are the omission failure in the proposed system.

An omission failure occurs when a server fails to respond to a request [7].Several things might go wrong. In the case of a receive omission failure, the server perhaps never got the request in the first place. Note that it may well be the case that the connection between a client and a server has been correctly established, but that there was no thread listening to incoming request. Also, a receive omission failure will generally not affect the current state of the server, as the server is unaware of any message sent to it.

Likewise, a send omission failure happens when the server has done its work, but somehow fails in sending a response. Such a failure may happen, for example, when a send buffer overflows while the server was not prepared for such a situation. In contrast to a receive omission failure, the server may now be in a state reflecting that it has just completed a service for the client. As a consequence, if the sending of its response fails, the server may need to be prepared that the client will reissue its previous request.

Other types of omission failures not related to communication may be caused by software errors such as infinite loops or improper memory management by which the server is said to hang.

5. Failure masking by Redundancy and replication

If a system is to be fault tolerant, the best it can do is to try to hide the occurrence of failures from other processes. The key technique for masking faults is to use redundancy. Three kinds are possible: information redundancy, time redundancy, and physical redundancy. With information redundancy, extra bits are added to allow recovery from garbled bits.

With time redundancy, an action is performed, and then, if need be, it is performed again. If a transaction aborts, it can be redone with no harm. Time redundancy is especially helpful when the faults are transient or intermittent. With physical redundancy, extra equipment or processes are added to make it possible for the system as a whole to tolerate the loss or malfunctioning of some components. Physical redundancy can thus be done either in hardware or in software. Process groups are part of the solution for building fault-tolerant systems. In particular, having a group of identical processes allows us to mask one or more faulty processes in that group. We can replicate processes and organize them into a group to replace a single process with a group. There are two ways to approach such replication: by means of primary-based protocols, or through replicated-write protocols.

Primary-based replication in the case of fault tolerance generally appears in the form of a primary-backup protocol. In this case, a group of processes is organized in a hierarchical fashion in which a primary coordinates all write operations. In practice, the primary is fixed, although its role can be taken over by one of the backups, if need be. In effect, when the primary

crashes, the backups execute some election algorithm to choose a new primary. Replicated-write protocols are used in the form of active replication, as well as by means of quorum-based protocols. These solutions correspond to organizing a collection of identical processes into a flat group. The main advantage is that such groups have no single point of failure, at the cost of distributed coordination.

6.Fault-tolerance replicating by agents

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. Multi agent system is appropriate solution for development difficult, distributed systems, but they are still vulnerable to any kinds of faults that any distributed system may have. These faults can come from program defects, sudden changes, processor and communication fault, as well as emerging actions, which are not predicted. Agent replication is the act of creating one or more duplicates of one or more agents in a multi-agent system [8]. Each of these agents may perform the same task as the original 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. In details, proxy is providing communication between replicates and other agents in the MAS. 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. 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. 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. It is shown in Figure “1”.

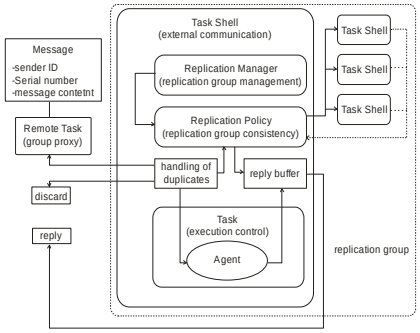

Figure 1. Replication management using Agent

7. Check pointing for fault tolerance of distributed database

In a fault-tolerant distributed system, backward error recovery requires that the system regularly saves its state onto stable storage. There are two different kinds of check pointing such as independent check pointing and coordinated check pointing. This system use coordinated check pointing.

In coordinated check pointing, all processes synchronize to jointly write their state to local stable storage. The main advantage of coordinate check pointing is that the saved state is automatically globally consistent, so that cascaded rollbacks leading to the domino effect are avoided.

A coordinator first multicasts a checkpoint request message to all processes. When a process receives such a message, it takes a local checkpoint, queues any subsequent message handed to it by the application it is executing, queues any subsequent message handed to it by the application it is executing , and acknowledges to the coordinator that it is has taken a checkpoint. When the coordinator has received an acknowledgement from all processes, it multicasts a checkpoint done message to allow the blocked processes to continue.

It is easy to see that this approach will also lead to a globally consistent state, because no incoming message will ever be registered as part of a checkpoint. At the same time, outgoing messages are queued locally until the checkpoint done message is received.

8. Overview of proposed system

In this section, we present the overview of the proposed system. 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 client-server system. Replication in this 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. The system uses the transactional replication because it can transfer only the changed data to the subscriber with minimal latency time than others. In this transactional replication, data alteration and custom processing can be easily implemented using custom replication stored procedures, while transferring the data. The proposed system sets checkpoint for the performance of client and sever with optimal recovery time. In addition, this system uses the agent replication for recovery of the accessing the distributed database system with minimal recovery time.

9. Experimental results

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

In figure 2, 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 this system, it also tried to cover the fault tolerance purpose, latency is a little greater.

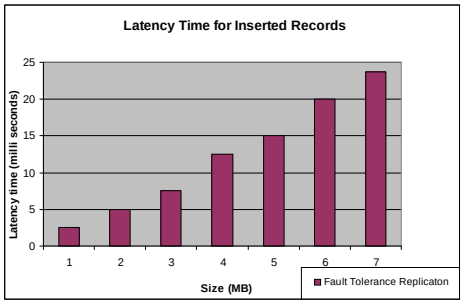

Figure2. Latency time for inserted records

The evaluation performance of updated records depending to their file size and the latency time is described in figure 3. According to the above experimental results, the latency time is gradually increased when 1MB of file size is reached 2,000 milliseconds to 7 MB of file size is reached 23,500 milliseconds. The bigger the file size, the higher the latency time is.

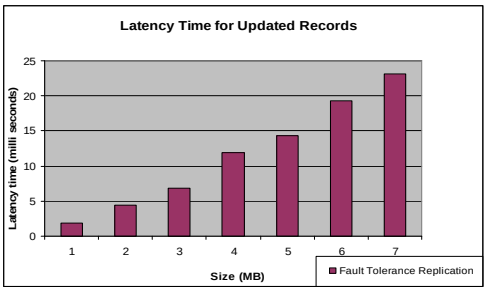

Figure 3. Latency time for updated records

Similarly, the horizontal axis represented measurement of file size in megabytes as shown in figure 4. 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.

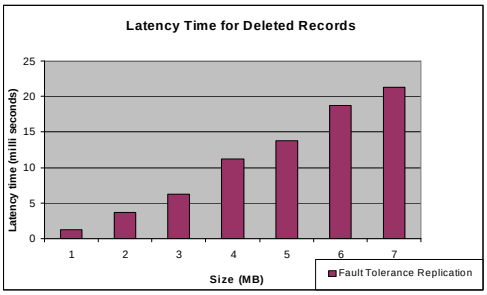

Figure4. Latency time for deleted records

As mentioned above, the latency time of three operations of the system is slightly difference. The experimental results show that by using the mobile agent system, the latency time for various operations on different files are not too different. So it can be used efficiently.

10. Conclusion

We use the system with some mechanism that support for fault tolerance replication. At that point, we use the agent replication to replicate the database. Check pointing is also use for recovery of the accessing for the databases. We compute the latency time of insert, deleted, update operations depend on the file size. The bigger the file sizes, the higher the latency time. According to the expected 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. 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 with safely.

References

[1] Esther PACITTI , Improving Data Freshness in Replicated Databases, Institut National De Recherche En Infromatique Et En Automatioque, No 3617, February 1999.

[2]F.D.Munoz et.al, Database Replication Approaches, Instituto Tecnol´ogico de Inform´atica, Technical Report TR-ITI-ITE-07/19, October 5, 2007.

[3] A.Fedoruk, R.Deters, Improving Fault tolerance by Replicating Agents, AAMAS'02, Bologna, Italy2002.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[4] S.Hagg, A Sentinel Approach to FaultHandling in Multi-Agent Systems, Department of omputer Science, University of Karlskorna Ronneby,Sweden.

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

[6] C.J.Date, An Introduction to Database Systems 7th edition, May 2000.

[7]A.S.Tanenbaum et.al, Distributed Systems, Principles and Paradigms, Interantional Edition.

[8]Y. Gershteyn, Fault Tolerance in Distributed Systems, Department of Computer Science, Rochester Institute of Technology, Rochester, NY, USA, February 5, 2003.