Journal of Theoretical and Applied Information Technology

www.jatit.org

USING REPLICATED DATA TO REDUCE BACKUP COST IN DISTRIBUTED DATABASES

[1]ALIREZA POORDAVOODI, [2]MOHAMMADREZA KHAYYAMBASHI, [3]JAFAR HAMIN

[1, 3] M.Sc. Student, Computer Department, University of Sheikhbahaee, Isfahan, Iran

[2] Asstt. Prof., Computer Department, University of Isfahan, Isfahan, Iran

ABSTRACT

To increase fault tolerance in distributed database, it is better to add a backup server for each primary server in the system. It is clear that the primary server and backup server need to be connected to each other. To connect these computers to each other when they are in a long distance, it is necessary to use a lease line which needs to be charged as data is transferred. As more packets are transferred between primary and backup server, more money need to be paid for charging this line. So if number of transferred packets between these computers reduces, the company can economize in its expenditures. On the other side, when number of updating information from the primary server to backup server reduces, the number of transaction which should be performed in the backup server reduces.

To achieve this goal, we introduce a new method which reduces the number of transferred packet between primary and backup server. In this method, the replicated data of primary server is used to backup mechanism. In our method the primary server sends transactions in data which are not replicated in other computers. So the transactions on the replicated data are not transferred to backup server, and as a result the numbers of transferred packet get reduce.

Keywords: *Distributed database, Fault tolerance, Remote backup, Data replication, Load balancing, Update filtering*

1. INTRODUCTION

One of the non-functional requirements in distributed databases is performance [1]. A technique to improve this requirement is data replication [2-7]. Data replication maintains multiple copies of data, called replicas, on separate servers. So the requests to these servers are answered locally [8, 9]. Replication improves performance by the following: *i)* reducing latency, since users can access replicated data, so it avoids remote network access; and *ii)* increasing throughput, since multiple computers can serve data simultaneously. When data is replicated in more than one computer, it is necessary to keep it consistence. There are some protocols such as single lock, distributed lock, primary copy, majority protocol, biased protocol, and quorum consensus protocol [10] which are responsible to keep data consistence.

Another non-functional requirement in distributed system is fault tolerance. It constructs the system in such a way that it can automatically recover from partial failures without seriously affecting the overall performance. So it ensures that the system can work accurately even in case of occurrence of faults [11, 12]. We can achieve this ability by performing transaction processing at one server, called the primary server, and having a remote backup server where all data in the primary server are replicated. The remote server must be kept synchronized with the primary server, as updates are performed at the primary. This synchronization is achieved by sending all log records from primary server to the remote backup server. The remote backup server must be physically separated from the primary, so a disaster at the primary does not damage the remote backup server. When a primary server fails, its remote backup server is responsible to answer the requests until the primary server come back to stable state [10, 13]. Figure 1 shows the architecture of a remote backup system.

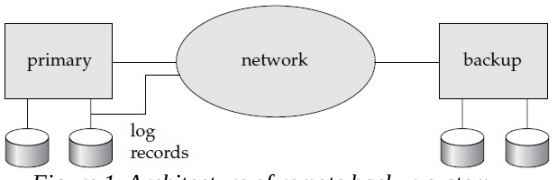

Figure 1. Architecture of remote backup system

In some situations data in the primary server is replicated in some guest computers. In these situations replicated data is used to reduce necessary time to access data. This data can be used for an extra function; recovering from failure. In this paper we introduce a method which uses this data to recover the primary server from failure. This method causes decreasing traffic between primary server and backup server, and as a result remote backup server is updated by a better performance. On the other side, the numbers of transferred packet gets reduce and less money needs to be paid for charging the leased line between primary and backup servers.

The remainder of paper includes following sections: Section 2 expresses related work. Section 3 describes how to use replicated data in backup process. Section 4 includes implementation remarks. In section 5, we evaluate our method and in section 6, some constrains in our method is presented. After that in section 7, we present a conclusion.

2. RELATED WORKS

One of the goals in using distributed databases is high availability; that is, the database must function almost all the time. In particular, since failures are more likely in large distributed systems, a distributed database must continue functioning even when there are various types of failures. The ability to continue functioning even during failures is referred to as robustness. For a distributed system to be robust, it must detect failures, reconfigure the system so that computation may continue, and recover when a processor or a link is repaired.

Remote backup systems and replication are two alternative approaches to provide high availability in distributed database. Remote backup systems offer a lower-cost approach to high availability than replication. On the other hand, replication can provide greater availability by having multiple replicas available, and using the majority protocol.

In the majority-based approach, each data object stores with it a version number to detect when it was last written to. Whenever a transaction writes an object it also updates the version number in this way:

• If data object *a* is replicated in *n* different sites, then a lock-request message must be sent to more than one-half of the *n* sites in which *a* is stored. The transaction does not operate on *a* until it has successfully obtained a lock on a majority of the replicas of *a*.

• Read operations look at all replicas on which a lock has been obtained, and read the value from the replica that has the highest version number. (Optionally, they may also write this value back to replicas with lower version numbers.)

Writes read all the replicas just like reads to find the highest version number (this step would normally have been performed earlier in the transaction by a read, and the result can be reused). The new version number is one more than the highest version number. The write operation writes all the replicas on which it has obtained locks, and sets the version number at all the replicas to the new version number.

Failures during a transaction (whether network partitions or site failures) can be tolerated as long as (1) the sites available at commit contain a majority of replicas of all the objects written to and (2) during reads, a majority of replicas are read to find the version numbers. If these requirements are violated, the transaction must be aborted. As long as the requirements are satisfied, the two-phase commit protocol can be used, as usual, on the sites that are available. In this scheme, reintegration is trivial; nothing needs to be done. This is because writes would have updated a majority of the replicas, while reads will read a majority of the replicas and find at least one replica that has the latest version [10]. When a transaction needs to lock data item *Q*, it simply requests a lock on *Q* from the lock manager at one site that contains a replica of *Q*. As before, the response to the request is delayed until it can be granted.

In this special case, there is no need to use version numbers; however, if even a single site containing a data item fails, no write to the item can proceed, since the write quorum will not be available. This protocol is called the read one; write all protocol since all replicas must be written.

To allow work to proceed in the event of failures, we would like to be able to use a read one, write all available protocol. In this approach, a read operation proceeds as in the read one, write all scheme; any available replica can be read, and a read lock is obtained at that replica. A write operation is shipped to all replicas; and writes locks are acquired on all the replicas. If a site is down, the transaction manager proceeds without waiting for the site to recover.

While this approach appears very attractive, there are several complications. In particular, temporary

communication failure may cause a site to appear to be unavailable, resulting in a write not being performed, but when the link is restored, the site is not aware that it has to perform some reintegration actions to catch up on writes it has lost. Further, if the network partitions, each partition may proceed to update the same data item, believing that sites in the other partitions are all dead.

The read one, write all available scheme can be used if there is never any network partitioning, but it can result in inconsistencies in the event of network partitions [10].

There are some methods to backup primary server in different database management systems. For example, Transaction Replication, Failover Clustering, Log Shipping, and Database Mirroring are some backup methods which are used in SQL Server 2005 [14]. The Database Mirroring method is one of the best methods for backup data. In the simplest deployment of database mirroring, there are two major server-side components, the *principal server instance* (principal) and the *mirror server instance* (mirror) [14]. The principal, as the name implies, contains the principal database. This is the database where you will perform your transactions.

The basic idea behind database mirroring is that synchronized versions of the database are maintained on the principal and mirror. If the principal database becomes unavailable, then the client application will smoothly switch over to the mirror database, and operation (from the user's point of view) will continue as normal. So, a client interacts with the principal and submits a transaction. The principal writes the requested change to the principal transaction log and automatically transfers the information describing the transaction over to the mirror, where it is written to the mirror transaction log. The mirror then sends an acknowledgement to the principal. The mirror continuously uses the remote transaction log to "replicate" changes made to the principal database to the mirror database. This relationship between mirroring components is shown in figure 2 [14].

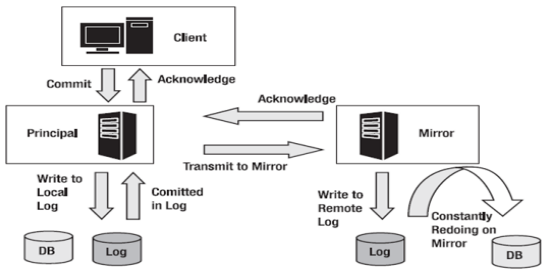

Figure 2. A database mirroring architecture

3. USING REPLICATED DATA TO REDUCE BACKUP COST

When data in primary server are replicated in some guest computers, the guest computers can be used to recover system from failure. So updating information from primary server to backup server can be reduced. As a result, processing load in the backup server and network traffic between primary server and backup server are reduced. In this situation, it is not necessary to transfer the transactions on the replicated data from the primary server to the backup server. The primary server sends to backup server just the transactions on data which are not replicated to the guest computers. So all data in the backup server is not updated, and only the data which is not replicated is updated. When a crash takes place in the primary server, the backup server requests other updated data from the guest computers which have replicated version of that data. In fact in our method, the replicated data is used to update backup server.

To apply our method in the distributed database, some operations in the primary server, backup server, and guest computer must be changed or added. In the following sections we describe these necessary changes.

3.1. Necessary Changes in the Primary Server

As we mentioned, the primary server transfer just transaction on data which is not replicated in the guest computers. So the primary server and backup server should know which data item is replicated in which guest computer. Hence when the primary server replicates a data item to a guest computer, assigns a name to that data and send this name along data item to the guest computer. After that, the primary server sends a packet to backup server which includes name of replicated data and the guest in which is replicated.

If a guest computer requests a data item which is named already, the primary server does not name it again. The primary sends data item with its name to the guest. It also resends data item name and new guest computer which requests that data to the backup server.

3.2. Necessary Changes in the Backup Server

As we mentioned, some data in backup server in not updated. Instead, it receives from the primary server name of this data and the guest computer which has this data. So when the primary server

fails, backup server has to get these data from appropriate guest and updates its old data by new data. If a data item is replicated to more than one guest, backup server should receive that data item form a guest which has the last version of that data.

3.3 Necessary Changes in the Guest Computers

When a guest computer requests a data to replicate form the primary server, it receives name of that data along with data. The guest computer should save that name along data. So in the failure time, if the backup server requests a data item by its name, the guest computer know which data corresponds with that name, and transfers those data to the backup server.

4. IMPLEMENTATION REMARKS

As we mentioned, the primary sever should name data item which is replicated in the guest computers. To name data items, primary can use a counter; a data item is named according to the counter. After naming a data item, the counter is increased by one. To assign this name to data items for example in a relational database, it is better to add a column named *data_name* to each table in database. So name of each record is inserted in this column. All records in a data item have the same name and the value of *data_name* attribute of these records is equal to each other.

To clarify the issue, suppose a primary server, its backup server, and two guest computers are connected to each other like figure 3.

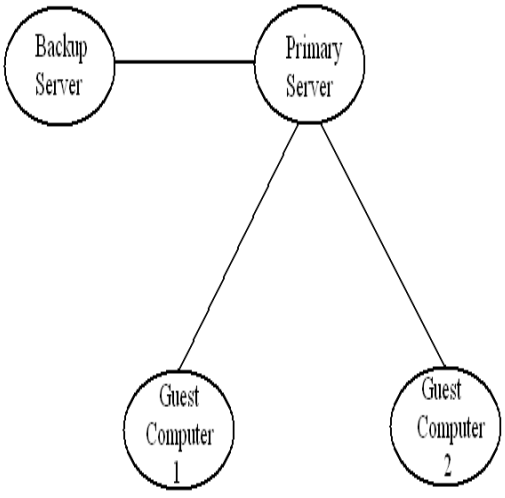

Figure 3. A distributed database architecture

There is a database named *Student* in the primary server which some records in it are shown in table 1.

Table 1. some records in the Student table

Row	Name	Std_Number	data_name
1	Aaaa	1111	-
2	Bbbb	2222	-
3	Cccc	3333	-
4	Dddd	4444	-
5	Eeee	5555	-
6	Ffff	6666	-
7	Gggg	7777	-
8	Hhhh	8888	-
9	Iiii	9999	-

Suppose guest 1 requests records number 1, 3, and 5 to replicate from the primary server. The primary server images these records as a data item and names it *Q1*. So the value of *data_name* attribute for these records is equal to *Q1* and this name is sent to the guest 1 and backup server. This process is shown in figure 4.

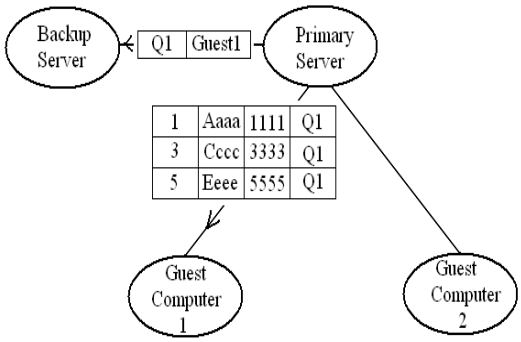

Figure 4. Replicating data to guest 1 from the primary

Now suppose guest 2 request records number 3, 7, 8. Records number 3 is member of data item *Q1*, so the primary server send all records in data item *Q1* to the guest 2. After that it assign a name *Q2* to records number 7 and 8, and send them to the guest 2. This process is shown in figure 5.

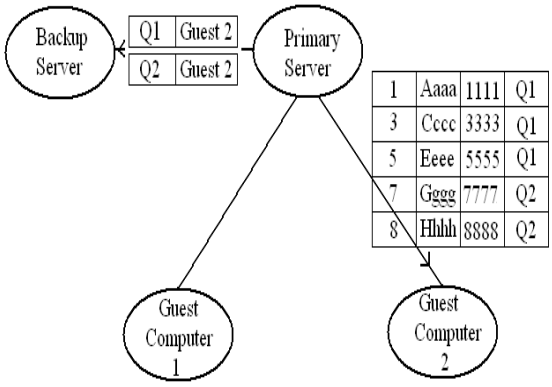

Figure 5. Replicating data to guest 2 from the primary

So if a guest requests from the primary server some records which are subset of a data item *Q1*, all records in that data item are sent to the guest.

Now suppose the primary server fails. The backup server knows data item *Q2* is replicated in guest 2, and requests this data item from guest 2. Data item *Q1* is replicated in both guest 1 and 2. The primary requests this data item from the guest which has the last version of that data item. So the primary requests this data item from guest 2 and updates its data. This process is shown in figure 6.

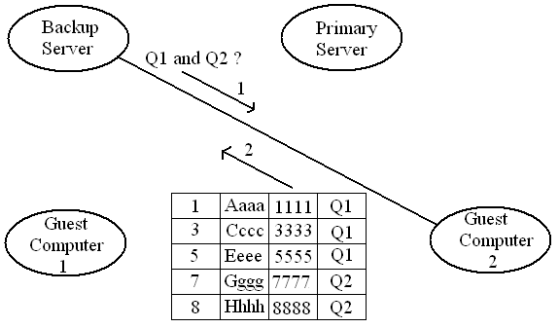

Figure 6. Transferring data to the backup server

5. EVALUATION

In this section we want to evaluate our method. As we mention in previous section, when data in the primary server is replicated in some guest computers, this replicated data can be used to recover system form a failure state. As a result processing load and network traffic between primary server and backup server decreases. This improvement, impose a processing load in backup server and guest computers in the failure time. On the other side, to apply our method to distributed database, some additional operations should be done in the primary server, backup server, and guest computers. These operations impose some additional cost to the system.

In this section we want to measure network traffic cost which is imposed to the system. First we are going to measure the network traffic between primary and backup server in mirroring method. In this method primary server sends the transactions on data to the backup server in specific interval time. After that we are going to apply our method to the system and measure the network traffic which is imposed to the system. Finally we can compare the network traffic in our method with mirroring method.

To evaluate our method we use SQL Server 2005 database system. Our primary server has 200000 records. We replicate 500 records to a guest computer and perform 50 update transactions to these records. The network traffic to update backup server in the mirroring method is equal to 250 Kbyte. If we apply our method to the system, this value is equal to 55 Kbyte. We continue to increase replicated data and perform the same 50 update transaction to them and monitor network traffic in our method and the mirroring method. The result is shown in table 2. figure 7 shows this result as a diagram. As you see, the amount of replicated data effects in the result. As the replicated data gets increase, the difference between network traffic in the mirroring method and our method gets increase. So the amount of replicated data effects in performance in our method, and improves it.

Table 2. Relationship between network traffic and replicated data in the mirroring and our method

Network Traffic between Primary and Backup Server (MByte)		Number of Replicated Records
Our Method	Mirroring Method	
0.055	0.25	500
0.102	0.25	1000
0.508	3.125	5000
1.016	5.75	10000
3.555	16.812	30000
5.078	30.687	50000
10.148	55.312	100000
20.297	159.812	200000

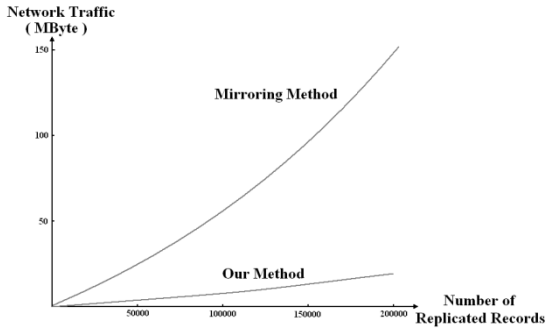

Figure 7. Diagram of relationship between network traffic and replicated data in the mirroring and our method

6. METHOD CONSTRAINS

There are some constraints in our method. First, if the primary server and one of the guest computers which has a data which is replicated only on that guest computer get fail concurrently, the backup computer cannot recover all data, and as a result data may be inconsistent. As you know this condition takes place very seldom. Second, our method imposes some over load in the primary server. So the primary server should have a suitable hardware to continue its work. Third, the recovering system from failure needs more time than previous methods. Because the backup computer should gets updated information from the guest computers, if the number of guest computers is high, this process may needs very much time.

7. CONCLUSION

This paper presents a new method to decrease network traffic between primary server and backup server. In this method replicated data is used for backup process. Transactions on data which is replicated in the guest computers are not transfer from primary server to backup server. Instead, in the primary server failure time, this replicated data are transferred from the guest computers to the backup server.

Reduction of network traffic between primary server and backup server causes fewer transactions execute in the backup server. In addition, the company pays less money for charging the link.

REFERENCES:

[1] A. Sleit, W. AlMobaideen, S. Al-Areqi, and A. Yahya, "A Dynamic Object Fragmentation and Replication Algorithm in Distributed Database Systems"**,** *American Journal of Applied Sciences*, Vol. 4, No. 8, 2007, pp. 613-618.

[2] J. Holliday, D. Agrawal, and A. E. Abbadi, "Partial database replication using epidemic communication". *In Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 485–493. IEEE Computer Society, 2002.

[3] T. Loukopoulo and I. Ahmad, "Static and adaptive distributed data replication using genetic algorithms", Journal of Parallel and Distributed Computing, 64(11):1270–1285, 2004.

[4] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin, "An Architecture for a Global Internet Host Distance Estimation Service", *in Proc. of the IEEE INFOCOM '99 Conf.*, March 1999, pp. 210-217.

[5] A. Vigneron, L. Gao, M. Golin, G. Italiano and B. Li, "An Algorithm for Finding a k-median in a Directed Tree", *in Information Processing Letters*, Vol. 74, No (1,2) , April 2000, pp. 81-88.

[6] T. Loukopoulos and I. Ahmad, "Static and Adaptive Data Replication Algorithms for Fast Information Access in Large Distributed systems", *in Proc. of the 20th IEEE Int. Conf. on Distributed Computing Systems*, Taipei, Taiwan, 2000.

[7] S. Jamin, C. Jin, T. Kurc, D. Raz and Y. Shavitt, "Constrained Mirror Placement on the Internet", in Proc. of the IEEE INFOCOM 2001 Conf., Alaska, USA.

[8] S. Abdul-Wahid, R. Andonie, J. Lemley, J. Schwing, and J. Widger, "Adaptive Distributed Database Replication Through Colonies of Pogo Ants", *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 358.

[9] T. Loukopoulos, I. Ahmad, and D. Papadias, "An Overview of Data Replication on the Internet", *IEEE Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks*, 2002, pp. 27-32

[10] A. Silberschats, H. F. Korth, S. Sudarshan, "Database System Concepts", McGraw-HILL, 2006.

[11] I. Gashi, P. Popov, and L. Strigini, "Fault tolerance via diversity for off-the-shelf products: A study with SQL database servers",

IEEE Transactions on Dependable and Secure Computing, Vol. 4, No. 4, Oct./Dec. 2007, pp 280–294.

[12] C. Wang, F. Mueller, C. Engelmann, and S. Scott, "A job pause service under lam/mpi+blcr for transparent fault tolerance", *IEEE In International Parallel and Distributed Processing Symposium*, March 2007, pp. 1-10.

[13] C. Mohan, K. Treiber, and R. Obermarck, "Algorithms for the management of remote backup databases for disaster recovery", *In Proc. Of 9th International Conference on Data Engineering*, 1993, pp 511-518.

[14] T. Rizzo, A. Machanic, J. Skinner, L. Davidson, R. Dewson, J. Narkiewicz, J. Sack, and R. Walters, "Pro SQL Server 2005", chapter 15, Apress, 2006.