Simulation Study of a New Optimistic Concurrency Control Approach for Replicated Databases

Nyein Nyein Ei
University of Computer Studies, Monywa
nyeinnyeinei@gmail.com

Nyein Nyein Myo
University of Computer Studies, Mandalay
vicky.mdy@gmail.com

Abstract

Replication of the database is the process of creating and maintaining multiple instances of the same database, and the process of sharing data or changes in database design between databases at different locations without having to copy the whole database. A protocol for concurrency control ensure that incorrect behavior cannot occur as a result of multiple clients being able to access the same time.Optimistic concurrency control (OCC) is a concurrency control method applied to transactional systems such as relational database management systems and software transactional memory. We proposed OCC algorithm to solve this issue of tolerant restart for abort transactions. And we proposed a framework for lazy replicated database scheme. In this paper, we compare new optimistic concurrency control protocol with traditional protocol for replicated databases. And we present the commit times between the slaves that are connected the simulation system of our protocol.

1. Introduction

The data is replicated for performance and availability at multiple network nodes [1]. The main purpose of replication is to increase availability and continuity, as a service is available even though some of its replicas are not functional. There are two replication methods. These are Lazy replication and egar replication. In this system, we use the lazy replication method.

To be consistent, we use snapshot database in each replica and use for transaction conflict validation database named certification database (CertDB) [2]. We propose a new Optimistic Concurrency Control algorithm in replicated distributed environment which is especially for abort transactions. An optimistic concurrency control in replicated databases mechanism uses Snapshot Isolation (SI) for deadlock free environment.

We group the mechanisms of concurrency control into two broad classes: pessimistic methods of concurrency control and optimistic methods of concurrency control [3]. Likewise, the positive category may be categorized as either lock-based or time-stamp-based ordering. This classification is described in following Figure 1.

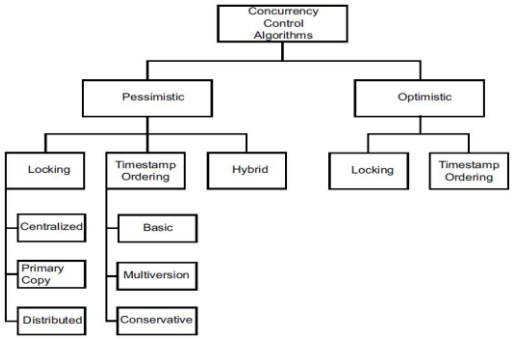

Figure 1.Hierarchical of Concurrency Control Protocols

The update transactions are submitted and executed directly on the master site; the refresh transaction is sent to the slaves once the update transaction commits [3]. The sequence of execution steps is as follows: (1) an update transaction is first applied to the master replica, (2) the transaction is committed to the master, and (3) the slaves are sent the refresh transaction.Figure 2 contains a single copy of the server for database items, the ordering can be established with simplicity by using timestamps. The server site would tag every refresh transaction on a timeline according to the commit order of the actual update transaction and the clients would use the refresh transactions in time timeline order.

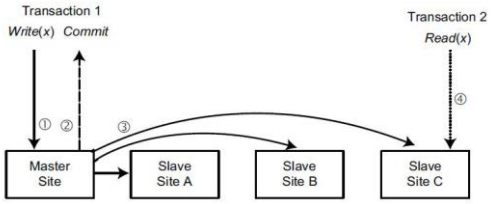

Figure 2. Lazy Single Master Replication Protocol Actions

Informally, positive management of concurrency is based on the assumption that apologizing is more effective than asking for permission [3]. Transactions are executed without synchronization but it is checked to ensure that it maintains atomicity until a transaction is permitted to commit.

The transaction commits if validation succeeds; otherwise the transaction is aborted and restarted. This paper proposes new, optimistic techniques for controlling concurrency. The traditional optimistic method of controlling concurrency aborted and restarted transactions updated by the slaves. And there is no favor, for late slaves.

Our proposed system sends massage to aborted transactions that marked. If the slave wants to update again, it send massage ‘YES’. If the other transaction

wants to update during the aborted transaction updating, the master commits these transaction.

Snapshot Isolation is a popular and efficient protocol for concurrency control [6]. The Snapshot Isolation Protocol produces multiversion schedules.

The organization of the thesis is as follows. Section 1 expresses the introduction of database replication and its nature, motivation. Principle of Optimistic Concurrency Control are briefly reviewed in section 2. Related works are reviewed in section 3. The Comparative Studies of Time Sequence Executions in Transactions are presented in section 4. In section 5, Overview and implementation of our Simulation Studiesare also discussed. And section 6, conclusions are drawn.

2. Principle of Optimistic Concurrency Control

In the following Figure 3, we show the phases of Optimistic Transaction Execution. Optimistic algorithms delay the validation phase until just before the write phase. Every transaction's read, compute, and write operations are performed freely without specific database updates. Initially each transaction updates itself on local copies of data objects. The validation phase consists of checking if the consistency of the database would remain with those updates. When the answer is yes, then the adjustments are made global. Otherwise, the transaction is aborted and has to restart.

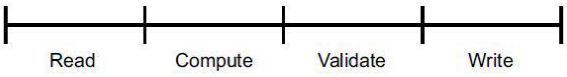

Figure 3. Phases of Optimistic Transaction Execution

For OCC the main component is the method of validation. To verify the principle of serial equivalence, the execution sequence of the concurrent transactions must be verified.

The basic concept is that if T_i transaction comes prior to T_j transaction, then T_i 's writings will not impact T_j's read process, and T_i should not overwrite T_j. Any of the following conditions must hold to satisfy this requirement.

Rule 1: T_i's write set does not intersect T_j's read set, and T_i completes its write phase before T_j begins its writing phase.

Rule 2: T_i's write set does not intersect T_j's read set or write set, and T_i completes its read phase before T_j ends its read phase.

If the checking condition is not satisfied, the validating transaction will be aborted and restarted.OCC protocols have the good properties of being non-blocking and deadlock-free. For database systems, these properties make them especially good. Between the transactions is delayed because conflict resolution until a transaction is near to its completion, there will be more information available on making the conflict resolution. Because some near-to-complete transactions have to be restarted, OCC protocols have the problem of unnecessary restarts and heavy restart overhead. Therefore, the major concernoptimistic concurrency control protocols is to design protocols that reduction the number of transactions to be restarted.We proposed optimistic concurrency control algorithm to solve this issue of tolerant restart for abort transactions [4]. And we proposed an architecture for lazy replicated database scheme [4].

3. Related Works

S. Elnikety at al., generalized isolation of the snapshot as used in Oracle and other databases in a way that fits replicated databases. Under GSI [7] they can be assured of Serializability. They suggested architecture and algorithms that use local concurrency controls to hold global weak SI in lazy replicated systems.K. Daudjee et al., proposed strong session snapshot isolation, a criterion of correctness which prevents reversals of transactions. They studied through performance; the cost of implementing the technique in lazy replicated systems is quantified [8].

A. Srivastava et al. proposed a new protocol for Concurrency Control in a distributed replicated environment. And they also proposed mechanism with some more assumptions, such as sending an extra message during the execution phase, but there is a significant one after completion of execution at local copy.Their approach increases the efficiency of the current transaction over O2PL and MIRROR in decreasing execution time and thus decreases the waiting time of transactions in the time queue [5]. Marius Cristian MAZILU suggested two methods for upgrading propagation, using immediate propagation.Such approaches can increase data freshness by up to five times compared with the deferred approach [10], they showed results.

4. Comparative Studies of Time Sequence Executions in Transactions

The Figure 4 shows a possible sequence of steps of execution of traditional optimistic concurrency control protocol in the following:

The transaction for Read is submitted at site slaves 1 or 2, where it is performed.

The transaction for update is submitted at site Master, and it is executed. In slave, the user wants to update x. The user sends the readsets of x with transaction. If not the other transaction wants to update x, the master commits the transaction 1 from slave1. And then the master user propagates the updated database to all slaves' database.

If the other transaction wants to update x during the transaction 1 updating x, the master aborts the later transaction from other slaves. If the updating finish, the master commits the transaction 1 from slave1. And then the master's user propagates the updated database to all slaves' database.

If there are 3 slaves want to update the item x and Slave1 is early than Slave 2 and 3, master commits the Slave 1's transaction. And then, the master refreshes to all slaves. The certifier will perform the above procedure.

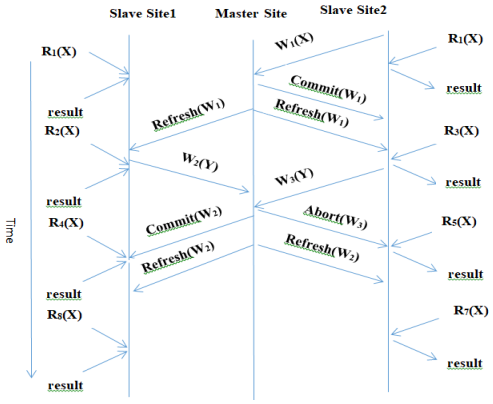

Figure 4. Time sequence executions of transactions of traditional protocol

The Figure 5 shows a possible sequence of steps of execution of our proposed protocol in the following:

The transaction for Read is submitted at site slaves 1 or 2, where it is performed.

The transaction for update is submitted at site Master, and it is executed. In slave, the user wants to update x. The user sends the readsets of x with transaction. If not the other transaction wants to update x, the master commits the transaction 1 from slave1. And then the master user propagates the updated database to all slaves' database.

If the other transaction wants to update x during the transaction 1 updating x, the master aborts the later transaction from other slaves. The system marks this aborted transactions. And the system takes the readsets from aborted transactions in certifier database. If the updating finish, the master commits the transaction 1 from slave1. And then the master's user propagates the updated database to all slaves' database. The system sends massage to aborted transactions that marked. If the slave wants to update x again, it send massage 'YES'. If the other transaction wants to update x during the aborted transaction updating x, the master commits these transaction. And then the master propagates updated database to all slaves' database.

If there are 3 slaves want to update the item x and Slave1 is early than Slave 2 and 3, master commits the Slave 1's transaction. The system marks the abort transactions from Slave 2 and Slave 3. The master takes the readsets of abort transactions in certifier database. And then, the master refreshes to all slaves. Here, the certifier needs to identify the priority for send the retry message. Receive time of readsets in certifier database is identified for that priority. So, the certifier will send retry message to slaves that early abort transaction. If these slaves want to update again, the certifier commits for the reply message and refresh to all slaves. And the certifier will consider sending the retry message the later transaction. The certifier will perform the above procedure.

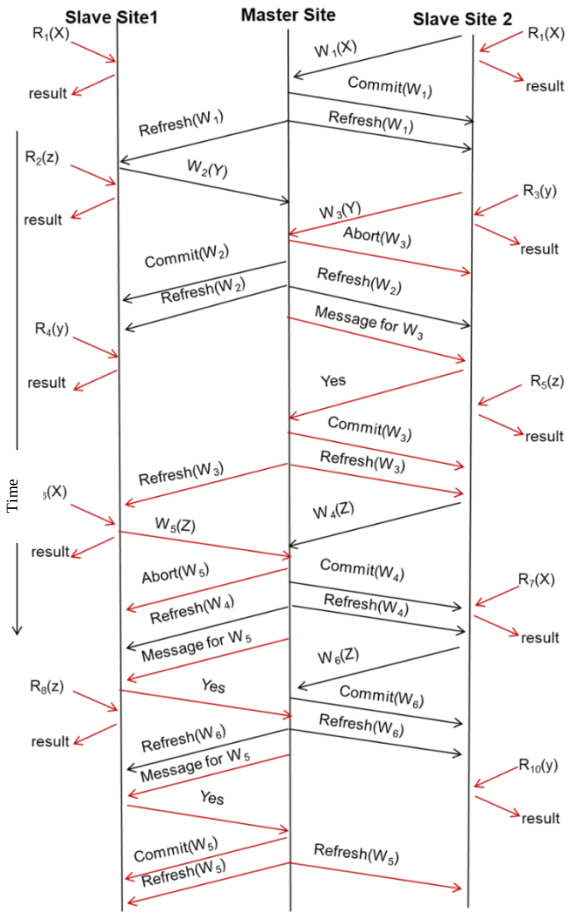

Figure 5. Time sequence executions of transactions proposed protocol

5. Simulation Study

5.1. Primary Northwind Database

Table 1. Category Table

Category ID	Category Name	Description	Picture
1	Beverages	Soft drinks, coffees, teas, beers, and ales	<Binary data>
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings	<Binary data>
. . .	. . .	. . .	. . .
8	Seafood	Seaweed and fish	<Binary data>

The simulator used 5 tables in Northwind Database. Table 1 shows the Category Table in Northwind Database as example. The following Figure 6 shows that how to relate between 5 tables of the Northwind Database.

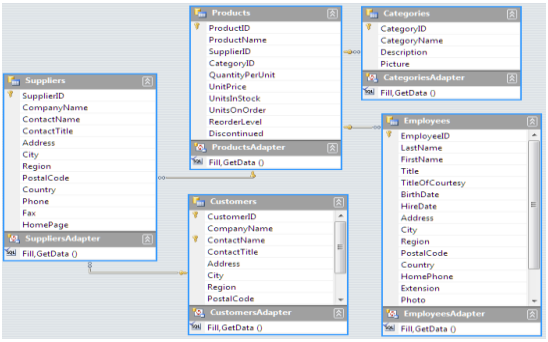

Figure 6. Table Relationships in Northwind Database

This simulator is developed with C# language in the Microsoft Visual Studio 2010 and the database in MS Access 2007. The machines used for the database certifier and replicas for the database are Dell Optiplex 755, running Window 7 32-bit. Each one has a 2.2GHz Intel Core 2 Duo Processor with 2 GB of DDR2 memory and a 250 GB SATA 7200RPM drive. Both devices have a single gigabit Ethernet interface and one gigabit Ethernet switch is connected to each.

The overall simulator can be roughly divided into two conceptual parts: the slaves and the servers. The server replicates the homogenous Northwind Database to the slaves. The slaves represent the source of transactions: Slaves are simple sources of transactions. Slaves can submit one transaction, wait until the transaction is processed, and send them to servers.

5.2. Slaves Site of Simulator

In slave site, the user can read only the snapshot database which was assigned previously. That snapshot database is not new database. If the slave does not connect to the server, the snapshot database does not have in slave site. If the slave user wants to update database, the user need to connect to server.

In connection to Master site, the user needs to insert the server name. The system inserts the slave name and IP address automatically. If the slave user doesn't know host name the Master computer, database replication process cannot be done and the slave user cannot also continue this process (update, insert or delete).

After connection to server, the server replicates and sends to the slave which is connected. In the slave log that snapshot database has been received from master site.

After the slave has received the snapshot database from server, the user can readonly these snapshot database. The slave user does not need to request to the server for readonly transaction. So, the response time can be good for readonly transaction. If the user wants to update the database, the user can click the update button. Readonly updated snapshot database in slave site can be seen in Figure 7.

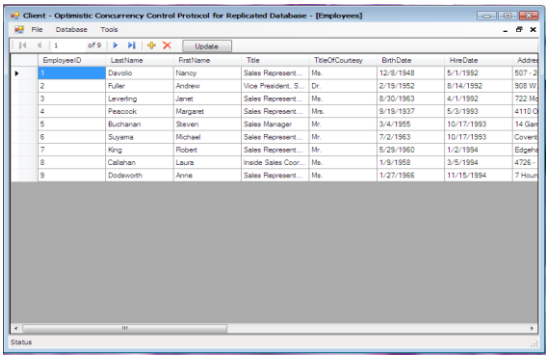

Figure 7. Readonly Updated Snapshot Database in Slave Site

After updating the data, the system commits in the local and sends the readset to the server. The slave waits for the server decision.

If there is only one update transaction in server, the server sends the updated database and message that all data tables are successfully updated by server to all slaves. The showing message updated database from server can be seen in the following Figure 8.

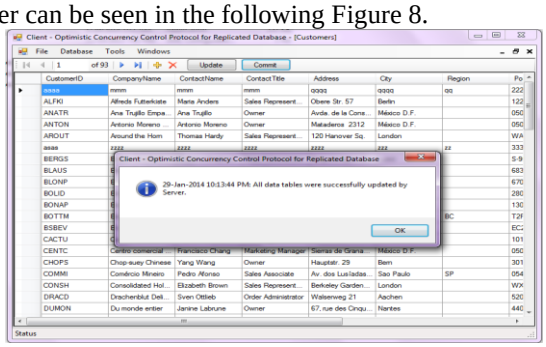

Figure 8. Showing Message Updated Database from Server

If there are two or more update transactions of the same data at same time in server, the certifier commits the former slave's update transaction and aborts the later slave's update transaction. The certifier saves the readsets of the aborted transaction in certifier database. The certifier updates the primary database, the certifier refreshes the database to all slaves and send message that "Update request is aborted from server. Do you want to retry again?" That can be seen in Figure 9.

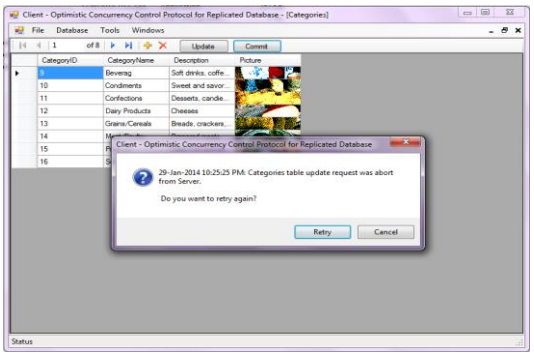

Figure 9. Showing message to retry message for aborted transaction

If the user wants to retry to update, the user can click retry button. This condition can get good response time.

The system give the favor for the late slave to update again that does not need to resend the readsets. If not, the user can click cancel button. If the slave wants to update this item, this slave restarts. This condition cannot get good response time.

5.3. Master Site of Simulator

In Master Site, there are the primary Northwind database and the certifier. The server user can see the certifier log file in Figure 10 that which slave is connected and snapshot database has been sent to this slave.

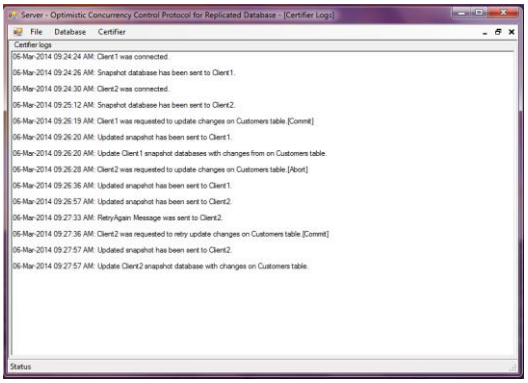

Figure 10. The Certifier Log File in Master Site

If one of the slaves requests to update changes on database and there is no conflict, the certifier commits and updates propagation to all slaves. In the case of update propagation, after one of the slaves's committing just finish, the master user must click the Update All submenu in Certifier menu immediately. If so, the response time can be good.

In the case of two update transactions of the same data at same time, the certifier commits the former slave's update transaction and aborts the later slave's update transaction. The certifier saves the readsets of the aborted transaction in certifier database. The certifier updates the primary database and propagates the updated database to all slaves.

Table 2. Comparison of Commit Times for Traditional and Proposed Protocols

		Request Time	Abort Time	Re-send Request Time	Commit Time	Reduce Time
Traditional Protocol	Client 1	9:25:12			9:26:19	
	Client 2	9:26:28	9:26:28	9:27:33	9:28:40	
		Request Time	Abort Time	Massage Time	Commit Time	
Proposed Protocol	Client 1	9:25:12			9:26:19	
	Client 2	9:26:28	9:26:28	9:27:33	9:27:36	00:01:04

In the above Table 2, we compare the commit times for traditional and proposed protocols according certifier log file in our simulator. The system have send the snapshot database to client 1 and 2. And the system send the snapshot database to every connected clients every update database. In client 1 and 2, client 2 late for update database. So client 2 is abort. In traditional method, client 2 need to re-send the database file that is want to update. In proposed protocol, client 2 does not need to re-send the database file. Client 2 only say 'Yes' message. So, our method reduce time for sending file about 00:01:04.

6. Conclusions

OCC protocols have the nice properties of being non-blocking and deadlock-free. The entire range of proposed algorithms has been evaluated using a detailed simulation system. Comparative study shows that the proposed algorithm has better than traditional approaches for most workloads. In this paper, we present the simulation studies of a new approach OCC for replicated databases.

This system is limited that the homogenous Northwind Database was used to test proposed OCC protocol in lazy replicated databases. The response time of committed transactions are not considered. The response time of aborted transactions are only considered. The certifier sends retry message that is validated in certifier database which is not fully automatic.

References

[1] J. Gray, P Helland, P. O'Neil and D. Shasha, "The Drangers of Replication and a Solution", the proceedings of the ACM SIGMOD Conference at Montreal, pp. 173-182, 1996

[2] Mihaela A. Bornea at al., "One-Copy Serializability with Snapshot Isolation under the Hood", 2011

[3] M. Tamer Özsu, Patrick Valduriez, "Principles of Distributed Database Systems", Third Edition, ISBN 978-1-4419-8833-1

[4] NyeinNyeinEi, NyeinNyeinMyo, "A New Approach Optimistic Concurrency Control for Replicated Databases", The 3rd International Symposium on Society, Tourism, Education and Politics (ISSTEP), pp.429-436, 2013

[5] A. Srivastava, U. Shankar and S. Kumar Tiwari, "A Protocol for Concurrency Control in Real-Time Replicated Databases System", IRACST – International Journal of Computer Networks and Wireless Communications (IJCNWC), ISSN: 2250-3501, Vol.2, No.3, June 2012.

[6] R. Normann and L.T. Ostby, "A Theroratical Study of "Snapshot Isolation", ICDT 2010.

[7] S. Elnikety at al., "Database Replication Using Generalized Snapshot Isolation"

[8] K. Daudjee at al., "Lazy Database Replication with Snapshot Isolation", VLDB 2006

[9] E. Pacitti and E. Simon, "Update Propagation Strategies to Improve Freshness in Lazy Master Replicated Databases", the VLDB Journal, Vol 8, pp. 305–318, 2000

[10] Marius Cristian MAZILU, "Database Replication", Database Systems Journal vol. I, no. 2, 2010

[11]http://office.microsoft.com/en-001/access-help/field-properties-quick-reference-HA010231953.aspx#BM0a