A study of Causally-Consistent Lazy Model for a Data-Centric Distributed Application

Me Me Win, Win Win Zaw
University of Computer Studies, Yangon
myacho.87@gmail.com

Abstract

In distributed database system, replicated databases are composed of many copies of databases distributed across different sites. Distributed applications frequently use replications to achieve higher level of performance, reliability and availability. The main challenge of database replication is to keep the data copies to be consistent in the presence of updates. In this paper, we study how the replicated databases can be prevented inconsistent data among users who access the replicated database simultaneously. Since, causal consistency perform operations that are causally related must be seen in causal order by all processes. Our study is focused on causally-consistent lazy model for a data-centric distributed application. For case study, we consider on Sale Ordering System.

Keywords: Replicated Database, Lazy Master Replication, Causally-Consistent, Data-Centric consistency model

1. Introduction

Nowadays, computers are widely used in business and society all over the world. Business environment has an increasing need for distributed database and client-server applications as the desire for reliable, scalable and accessible information. Distributed database systems provide an improvement on communication and data processing. So, distributed systems become popular. Using database in a single server causes fail and corruption in servers.

Therefore, databases are replicated and distributed across several sites. In this paper, replicated databases are stored on multiple server machines and multiple clients are accessed this database using online and to perform consistency control on that databases. The purpose of this study is to prevent inconsistent data among users who access the databases simultaneously. Users want to read up-to-date information so the system needs to guarantee consistency when users are making concurrent access. So, consistency control is important part of distributed system. Distributed system has two consistency models. They are data-centric consistency model and client-centric consistency model.

In the following, Section 2 describes related work with causally-consistent lazy replication. Section 3 describes data-centric consistency model. Section 4. describes case study overview. Section 5 describes the proposed system. Section 6 describes performance evaluation. Section 7 is conclusion and Section 8 describes all references of this paper.

2. Related work

Andrew Berry is described theory of causality. Maintenance of causality information in distributed systems has previously been implemented in the communications infrastructure with the focus on providing reliability and availability for distributed services. This approach has a number of advantages, moving causality information up into the view and control of the application programmer is useful. [1]

Michel Raynal and colleagues are proposed causally consistent distributed services when multiple related services are replicated to meet performance and availability requirements. Causal consistency is well suited for distributed services such as cooperative document sharing. Causal consistency is attractive because of the efficient implementations that are allowed by it. Causally consistent distributed services allow service instances to be created and detected according to service in the distributed system. [2]

Ladin and colleagues are proposed a scheme that provides data freshness guaranteed to read and write operations. In their work, if all writes are forced operations, they are directed to a single “primary” site, which orders them. This site uses a Two-Phase commit protocol to propagate in order, each forced write’s update to a majority of replicated sites. Subsequent operations are guaranteed to the effect of the forced write’s. In this work there is no notion of sessions and thus no provision of session guarantees. [4]

3. Data-Centric Consistency Model

A consistency model is essentially a contract between processes and the data store. It says that if processes agree to obey certain rules, the store

promise to work correctly. Process that performs read on data item expects the value of last write. So, the system needs to use consistency models. But difficult to define which write was last without global clock. Distributed systems have two consistency models. There are data-centric consistency model and client-centric consistency model.

Data-centric consistency model divided into two parts of consistency models. These are strong consistency models and weak consistency models. At strong consistency model, operations on shared data are synchronized. Strong consistency models are strict consistency, sequential consistency and causal consistency and FIFO consistency. At weak consistency model, synchronization occurs only when shared data is locked and unlocked. Weak consistency models are generally weak consistency, release consistency and entry consistency. Causally-Consistent Lazy Replication approach, is weaker than strict consistency and sequential consistency but stronger than FIFO consistency model, is used in this system. [3]

3.1 Causal consistency

Operations that are causally related must be seen in causal order by all processes. Concurrent writes may be seen in different order on different machines. Concurrent writes may be seen in different order by different processes. Concurrent operations that are not causally related. Implementation must keep dependencies between causal operations. Potential causality can be applied to a memory system by interpreting a write as a message-send event and a read as a message-read event. A memory is causally related events. Causally unrelated events (concurrent events) can be observed in different orders. For example, the following is a legal execution history under CC but not under SC:

P1:	W(x)a		W(x)c		
P2:	R(x)a	W(x)b			
P3:	R(x)a			R(x)c	R(x)b
P4:	R(x)a			R(x)b	R(x)c

Figure [1]: Causally Consistent

Note that W(x)a and W(x)2 are causally related as P_2 observed the first write by P_1. Furthermore, P3 and P4 observe the accesses W(x)b and W(x)c in different orders, which would not be legal under SC.

Among the uniform models, CC appears to be one of the more difficult to implement in hardware. This can probably be explained by the fact that most other models have been designed with a hardware implementation in mind. However, this does not imply that a CC implementation of one of the simpler uniform models. [5]

3.2. Causally-Consistent Lazy Replication

Causally-Consistent Lazy Replication performs two operations:

(i) Processing Read Operations

(ii) Processing Write Operations

(i) Processing Read Operations

When a client **C** wants a read operation **R** to be performed, the timestamp **DEP (R)** of the associated read request is set equal to **LOCAL(C).i.e.,**

DEP (R): = LOCAL(C)

This timestamp reflects what the client currently knows about the data store. The request is sent to one of the copies $\mathbf{L_i}$ is stored in that copy's read queue. An incoming read request **R** at a copy $\mathbf{L_i}$ is stored in that copy's read queue. In order to process **R**, it is necessary that knows about the state. For each copy, **DEP(R) [j] < = VAL (i) [j].i.e.,**

DEP(R) < = VAL (i) (j)

Copy $\mathbf{L_i}$ returns the value of the requested data item to the client, along with **VAL (i). i.e.,**

Data & VAL (i)

The client subsequently adjusts its own vector timestamp **LOCAL(C)** by setting each of the entries LOCAL (C) to the value max **{LOCAL(C) [j], VAL (i) [j]}. i.e.,**

LOCAL(C):=max {LOCAL(C) [j], VAL (i)}.

[3]

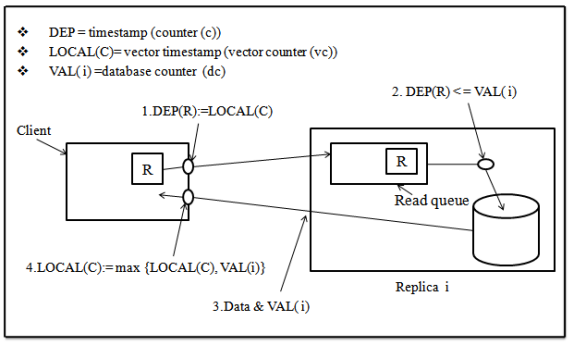

Figure [2]: Performing a read operation at a replica Copy

(ii)Processing Write Operations

When a client **C** wants to write operation to be performed, the timestamp DEP **(W)** of the associated write request is set equal to **LOCAL (C). i.e.,**

DEP (R): = LOCAL(C)

The request is sent to one of the copies $\mathbf{L_i}$**.** An incoming write request **W** at a copy $\mathbf{L_i}$ is stored in that copy's write queue. When a copy receives a write request **W** from a client, it increments **WORK (i) [j],** but leaves to other entries intact. The write request receives a timestamp **ts(W)** by $\mathbf{L_i}$, **ts(W)[i]**

set equal to **WORK(i)[j]** is set equal to **WORK(i)[j]** and the other entries **ts(W)[j]** is set to **DEP(W)[j].**

i.e., WORK (i) (j): = WORK (i) (j) + 1

This timestamp is sent back acknowledgement to the client, which adjusts **i.e., LOCAL(C)** by setting each entry to max **{LOCAL(C) [k], ts (W) [k]}. i.e.,**

LOCAL(C):= max {LOCAL(C) [j], ts (W)}

A write request **W** can be carried out if $\mathbf{L_i}$ has processed all updates on which **W** depends. For each entry **DEP (W) [j] <= VAL (i) [j]. i.e.,**

DEP(R) < = VAL (i) (j) [3]

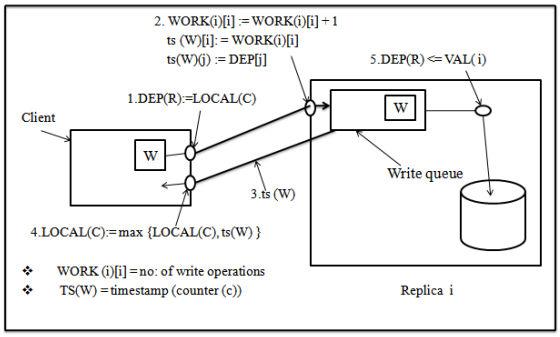

Figure [3]: Performing a write operation at a replica Copy

4. Case Study Overview

In this paper, the case study is an implementation of data-centric consistency model using Causally-Consistent Lazy Replication approach. It is a consistency control scheme for client-server systems. Data store is physically distributed and replicated across three servers. Among these servers, one server is primary server and remainders are replica servers.

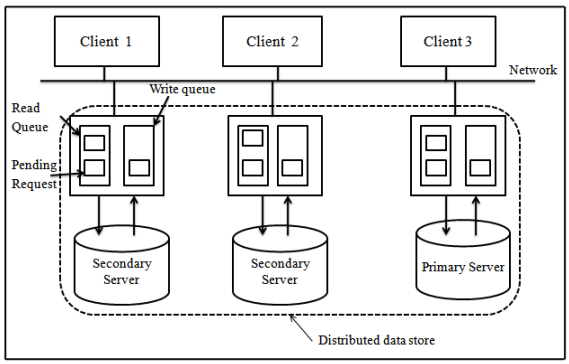

Figure [4]: Case Study Overview

These replica servers are also known as secondary servers. Primary server stores primary copy and secondary servers store secondary copy. Primary copy is stored at master site and secondary copies are stored at slave sites. Each server of the data store consists of a replicated database and two queue of pending operations. They are read queue and write queue. The read queue consists of read operations and the write queue consists of write operations. Clients can access book information from any server (primary server and secondary servers). Clients can perform read operations themselves. But write operations cannot perform by clients. Write operations are performed by the system of primary server. The case study overview is shown in Figure [4].

5. The Proposed System

The proposed system uses Causally-Consistent Lazy Replication approach that is performed on Client-Server architecture. The case study for this approach is Book Ordering System. The system has three servers. They are primary server and secondary servers (replica servers). Primary server is located to the master site that stores primary copy. Secondary servers are located to the slave site that stores secondary copies. The replicated database of primary server contains authoritative data that could be permanently stored without violating the global consistency model of the data store. Updating can only be performed on the primary server and the update data are propagated towards the other replica servers (secondary servers). Secondary servers are read only servers.

Each server of the data store consists of replicated database and two queues of pending operations. These two queues are read queue and write queue. The read queue consists of read operations that need to be held back until the replicated database is consistent. Eg. User request read operation that has already seen the effect of write operation. The write queue consists of write operations that need to be held back until the replicated database is consistent. A write operation can be carried out only if the replicated database of primary server is updated with all causally preceding operations on which that write operation depends. Updating can be performed only on the primary server and the updates are propagated towards the secondary servers.

Clients can access book information by using online. Clients can choose two types of requests that are read request and write request. Read request is used for querying book information from the server. Write request is used for ordering book from the server. For read operation, clients can get information by querying from any servers (primary server and secondary servers). Clients can perform read operations at any server but client cannot perform write operations themselves. Write operations are only performed by primary server. Causal consistency can perform operations that are causally related must be seen in causal order by all processes. To ensure, data is causally-consistent,
case study is used Lazy Master Replication for

update propagation.

Clients can perform operations that are possible four types: Read-Read Condition, Read-Write Condition, Write-Read Condition and Write-Write Condition. Write-Write Condition can be classified into two types upon ordering: same book ID and different book ID.

(i) Read-Read Condition

The system allows reading book information concurrently for all clients. So, multiple clients can read book information at any server (primary server and secondary servers).

(ii) Read-Write Condition

Client 1 requests to query book information at replica server. Client 1 can read all book information. At the same time, Client2 requests to order book at replica server. If client's order of book quantity is less than or equal to book quantity in the database then this order is valid. Otherwise order is invalid. Replica server checks Client 2's order is valid or invalid. If order is invalid then the system sends "Order Invalid" message to Client2. If order is valid then the replica server sends Client 2's request to the primary server. When the primary server receives this request, it updates its own copy. Updates from the primary server are successfully finished then these updates are propagated lazily to other secondary copies. If Client2's ordering is successfully finished then Client 1 can read new book information by the system is auto changing. Client 1 can read synchronizing up-to-date book information that is consistent.

(iii) Write-Read Condition

Client requests to order book at replica server. Replica server checks client's order is valid or invalid. If order is invalid then the system sends "Order Invalid" message is sent to client. If order is valid then the replica server sends client's request to the primary server. When the primary server receives this request, it updates its own copy. Updates from primary server are successfully finished then these updates are propagated lazily to other secondary copies. After update propagation is successfully finished, data from all sites are synchronized and consistent. At this time, client requests to read book information at replica server. Replica server sends all book information to client as shown in Figure [5].

The variables are used in sequence diagram are:

- DEP = timestamp (counter (c))
- LOCAL (C) = vector timestamp (vector counter (vc))
- VAL (i) = database counter (dc)
- W (i)[i] = number of write operations
- ts (W) = timestamp (counter (c))

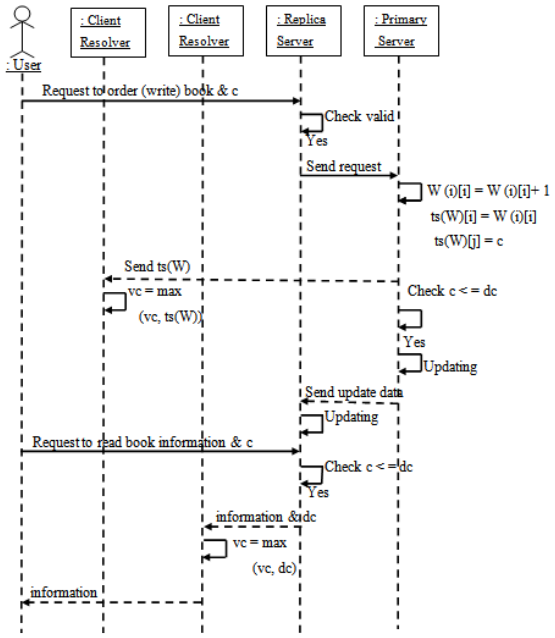

Figure [5]: Sequence diagram of Write-Read Condition

(iv)Write-Write Condition

Write-Write Condition can be classified into two types upon ordering:

(a)Write-Write Condition (same book ID)

(b)Write-Write Condition (different book ID)

(a)Write-Write Condition (same book ID)

Client 1 and Client 2 request to order book at replica server. If two requests of book IDs are the same then the system checks their arrival time. If Client 1's arrival time is less than Client 2's arrival time then Client1's request is access to order. According to their arrival times, Client2's request is pending in the queue. Replica server checks Client1'order is valid or invalid. If order is invalid then the system sends "Order Invalid" message to Client1. If order is valid then the replica sends Client1's request to the primary server. When primary server receives this request, it updates its own copy. Updates from primary server are successfully finished then these updates are propagated lazily to other secondary copies. If Client 1's order is successfully finished then the system checks waiting requests in the queue. Client 2's request is waiting in the queue and the system starts to perform the Client 2's request. Replica server sends Client2's request to the primary server. When primary server receives this request, it updates its own copy. Updates from primary server is successfully finished, the updates are propagated

lazily to other secondary copies. After update propagation, both two clients are seen synchronizing up-to-date data that are consistent. Data from all sites are synchronized and consistent.

(b) Write-Write Condition (different book ID)

Client 1 requests to order book at replica server. At the same time, Client 2 requests to order book at replica server. Since, two clients request are not the same, the system allows two clients to access processing concurrently. Replica server checks that their orders are valid or invalid. If orders are invalid then the system sends "Order Invalid" messages to the clients. If orders are valid then the system sends their requests to the primary server. When primary server receives this request, it updates its own copy. Updates from primary server are successfully finished then these updates are propagated lazily to other secondary copies. After update propagation is successfully finished, both two clients are seen synchronizing up-to-date data that are consistent. Data from all sites are synchronized and consistent.

6. Performance Evaluation

To quantity the performance, we use several measures. Completion time is the total execution of the application in a given system. The performance measures are Computation time, Synchronization time, Communication time and Network handling time. In these performance measures, we are used the communication time for performance evaluation.

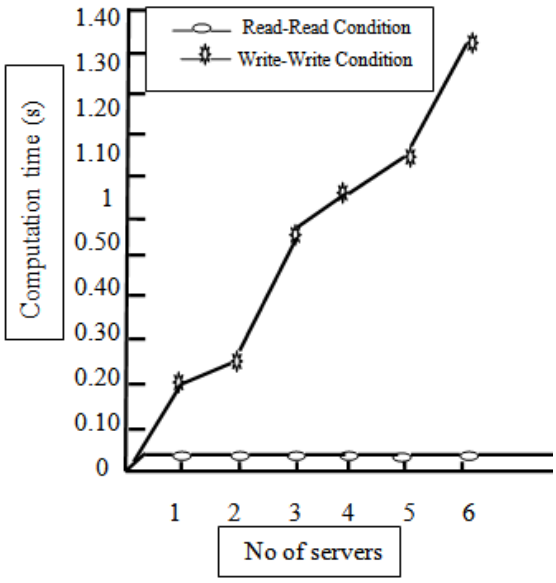

Figure [6]: Computation time

7. Conclusion

To study how we can implement a distributed database replication consistency management system using causally-consistent lazy replication. By using causally-consistent, it can help to get consistent data when updating copies of databases across different sites. Replication can give high availability from creation many copies of data at many servers that reduces data access latency and replica transparency. Replication can give good performance and reliability when one copy crashes by node failures and network partitions. Implementation must keep dependencies between operations using causally-consistent. Causally-consistent is difficult to implement in hardware among consistency controls but that reduce operation cost using lazy replication. Consistency across multiple related objects is not considered and hence vector timestamps are sufficient in the lazy replication protocol.

8. References

[1]. Andrew Berry, "An application-level implementation of causal timestamps and causal ordering," *IEEE Trans. Distrib.System.Engng 2*, May 1995.pp.74-86. Printed in the UK.

[2]. Ahamad, M…., M. Raynal, "An adaptive protocol for implementing causally-consistent distributed services*," in Proceedings of the 18th IEEE International Conference on Distributed Computing Systems*, May 1998 , pp. 86-93.

[3]. Andrew S. Tanenbaum, Maarten van Sten, "*Distributed Systems Principles and Paradigms*," International edn, Prentice Hall, 2002.

[4]. B.Lislov, L.Shrira, R.Ladin and S.Ghemawat, "Providing High Availability Using Lazy Replication," in *ACM Transactions on Computer-Systems*, 10(4): 360-392, 1992.

[5]. David, Mosberger, "Memory Consistency Models," *Oper.System.Rev*, vol.27, no.1, pp.18-26, Jan 1993.

[6]. Mustague Ahamad, Ranjit John, "Evaluation of Causal Distributed Shared Memory for Data-race-free Programs," *Georgia Institute of Technology*, College of computing, Atlanta, GA 30332-0280, 1994.