Implementation of Home-based lazy release consistency system for a distributed application

Zar Zar Moe, Thinn Thu Naing
zarzarmoe.88@gmail.com, ucsy21@most.gov.mm

Abstract

Distributed processing has more and more important and attractive with progress in the areas of computer network and distributed system. Database management system typically allows many transactions to access the same database at the same time. When multiple users are simultaneously updating over a shared database, data integrity and consistency problem become arise. This system database is stored on single server machine and copy of this database is stored on multiple client cache. Consistency control is performed on those databases. The goal of this system is to prevent inconsistent retrievals among users who are simultaneously accessing on share database. This system will implement the Home-based lazy release consistency control and vector timestamp synchronization by using train ticket sales system as case study.

1. Introduction

The problem of using shared data is that if the data has been modified in the meantime, modifications will not have been propagated to cached copies, making those copies out of date. The user always for want of the latest version of data, we need to do something about concurrent access to guarantee state consistency. In real time database system that is necessary to use consistency controls system to manage correctness and accuracy of data. So, consistency control is the important part of distributed system. Distributed system has two consistency models. There are data centric consistency model and client centric consistency model.

2. Data Centric Consistency Model

A consistency model is a contract between a distributed data store and its processes. If the processes obey the rules, the data store will perform correctly. Data centric consistency model divided into two part 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, causal consistency and FIFO consistency.

At weak consistency model, synchronization occurs only when shared data is locked and unlocked. Weak consistency models are general weak consistency, release consistency and entry consistency [5].

To avoid the disadvantage of release consistency control and to implement the lazy release consistency control, home-based lazy release consistency approach is used in this system. We present two main control algorithms in this system, they are:

(i) Server Control Algorithm
(ii) Client Control Algorithm

2.1. Home-based lazy release consistency

Home based lazy release consistency is a simple home based multiple writer protocol that implements LRC. The home node of the data (server) contains its master copy. This home node always hosts the most updated contents of the data, which can then be fetched by a non-home node (clients) that need an updated version. At a release, which marks the end of a critical section, a processor immediately generates the *copy of data* that it has modified since its last release. It then sends this copy *of update data* to their home processor, where they are immediately applied to the home's data. The home's data is never invalid, but it may be written protected. The copy of update data can be discarded by the creating and home processor as soon as it is applied to the home processor's data. The main advantage of HRC over LRC is that after communicating copy of data to the homes, they can be discarded [2,3,4,5].

2.2. Vector Timestamps synchronization

We use vector timestamps to indicate which user to allow for data modification. Vector timestamps are managed like vector clocks. Send and receive events

are replaced by release and acquire (of the same lock) respectively. A *lock grant* message (that is sent from releaser to acquirer to give acquire the exclusive ownership) contains the current timestamp of the releaser [1,5].

- Just before executing a release or acquire in *p*: $V_p[q]:= V_p[q] + 1$
- A *lock grant* message *m* is time-stamped with $t(m)=V_p$.
- Upon acquire for every *q*: $V_p[q]:= \max\{ V_p[q], t(m)[q] \}$

2.3. Server Control Algorithm with Home-based Lazy Release Consistency

//When the users request to read or write operation, the system check:

```
Begin
  1. If read operation then
       begin
            -sends grant read lock message to the user
            -sends latest data and its event timestamp Tj to the user
       end
     Else If write operation then
       begin
            If the item is already write locked then
                 -sends wait message to the user and put on a queue
            Else
                 -sends grant write lock message to the user
       end
     End

  2. If receive update from the client then
       begin
            If waiting user then
                 -sends copy of update and event time Tj=max (Ti, Tj) to the waiting users
                 -update database and event time Tj=max (Ti, Tj)
             Else
                 -update database and event time Tj=max(Ti, Tj)
       end
     End

  3. If receive release read or write lock message then
       begin
            If waiting user in queue then
                 -sends grant write lock message to the user
             Else
                 -sends grant read lock message to the user
       end
     End
End
```

2.4. Client Control Algorithm with Home-based Lazy Release Consistency

```
Begin
     1. If read operation then
          begin
           -fetch data from the server (if needed)
           -receive update data and event time Ti=max (Ti, Tj)
           -sends release read lock message to the server
        End

     2. If write operation then
          begin
           -execute the buying process
           -sends update data and its update time Ti to the server
           - sends release write lock message to the server
        End

     3. If receive update from the server then
          begin
           -update the local cache and event time Ti=max (Ti, Tj)
        End

End
```

2.5. Use case for Read-Read condition between two clients

When client1 request to read data, client1's cache will fetch data from server, server responses update data. Similarly, when client2 requested to read the same data, client2's cache will fetch data from server, server responded the update data. If client1 sends

release lock message, then server check waiting user in queue. If user is waiting to write request, then server allow buying ticket.

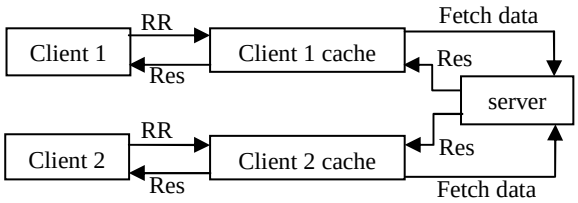
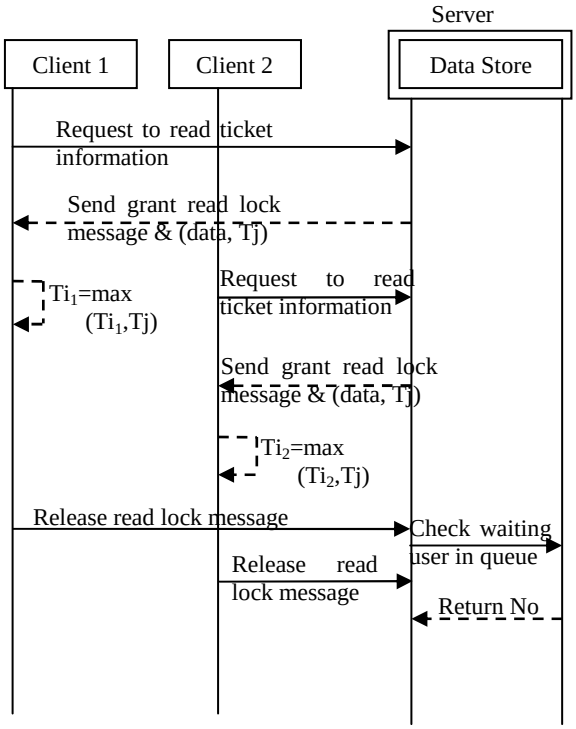


Figure 1. Reading phase

Note: RR - read request
Res – response

2.6. Sequence diagram for Read-Read condition between two clients



In this sequence diagram, when client1 request to read ticket information, server sent update data at latest event time Tj and then client1 is update its event time. Similarly, when client2 requested to read the same data, server returned the update data at latest event time Tj. So, client2 is updating its event time. If client1 sends release lock message, then server check waiting user in queue. If user is waiting to buy ticket, then server allow to buy ticket.

2.7. Use case for Write-Write condition between two clients

When client1 request to write data, server check another user is writing data. If no user isn't writing data, server allows to write data. So, client1 write data and update locally. Then client1 synchronizes data to server and update event time. While client1 is writing data, if client2 requested to write data then server will send wait message to the client2. When client1 has updated data, server checked waiting user is existed. If waiting user existed then server synchronizes data to client. When client2 receives the update data, it updates cache locally. At the time, server checked another user is waiting to write data. If user is waiting to write data then server allows writing data.

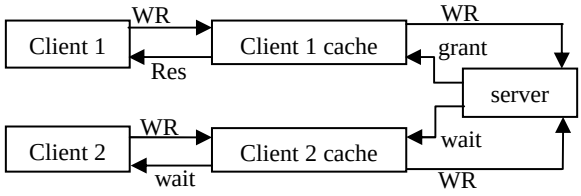


Figure 2. Waiting phase

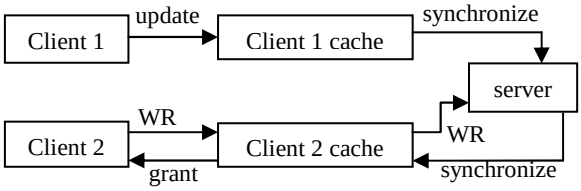


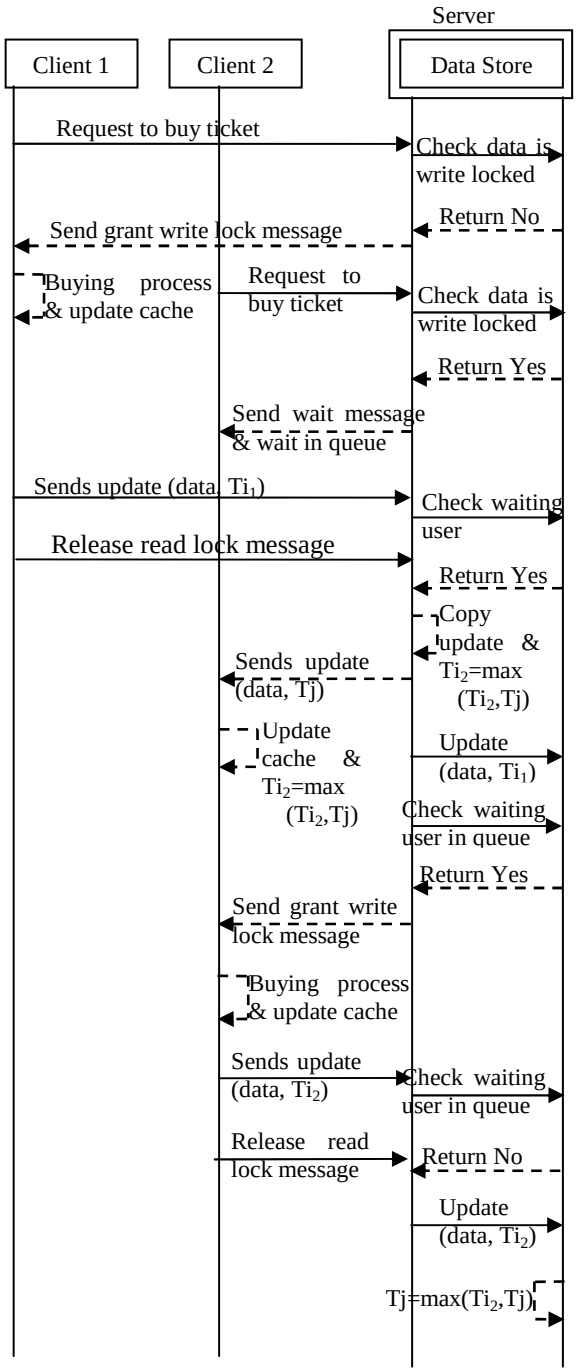
Figure 3. Updating phase

Note: WR - write request
Res – response

2.8. Sequence diagram for Write-Write condition between two clients

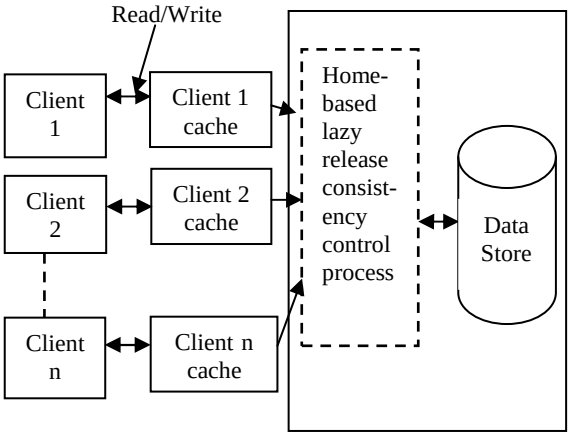
In this sequence diagram, when client1 requested to buy ticket, server check another user is buying ticket. If no user isn't buying ticket, server allows to buy ticket. So, client1 buy ticket and update locally. Then client1 synchronizes data to server and update event time. While client1 is buying ticket, if client2 requested to buy ticket then server will send wait message to the client2. When client1 has bought ticket, server checked waiting user is existed. If waiting user existed then server synchronizes data to client at latest event time Tj. When client2 receives

the update data, it updates cache locally. At the time, server checked another user is waiting to buy ticket. If user is waiting to buy ticket then server allows buying ticket.



2.9. System overview

This system is an implementation of proving data centric consistency model using home-based lazy release consistency approach. It is a consistency control schemes for client server system. Client acquires a lock, and then client fetches data from server machine, and then execution the update locally. When client release a lock, a processor immediately generates the copy for the update data. It then sends these update data to their home processor. When other client fetch the same data from server machine, the home processor of that data responds the most updated contents of the data. Train ticketing sales system as case study is used to generate consistent transactions for the shared database system.



2.10. Performance evaluation of the system

In this system, user can search the information about of the trip plan by choosing the source, destination of the cities and the date of the day, which user wants to travel. After satisfying the result data, if user requests to buy ticket, server checks another user is buying ticket and waiting user in queue. If no user is buying and no waiting user in queue, then user can buy the ticket. If the user is sure to buy the schedule, user must fill schedule number, seat type, number of seat, user name, nuc-no, phone, e-mail. Thereupon valid data added in the server database as shown in table 1.

log_id	client	timestamps	validate
1	1	12:54:54	Invalid
2	1	12:54:59	Valid
3	2	12:55:14	Invalid
4	2	12:56:20	Valid
5	1	12:57:18	Invalid

Table 1: Dynamic application performance

While the client1 reserves the ticket, another user (client2) will wait in the queue. Thereupon invalid data added in the server database as shown in table 1. After client1 reserves the tickets, the schedule's available seat will be changed and its database will be updated, and then client1 synchronizes to the server. Therefore, schedule's available seat number which user's bought ticket number (eg.11) changed in server database and

client1 database but client2 database does not change as shown in table 2, table 3, and table 4.

Seat _id	Schedule _ id	Seat type	Available seat	price
1	1	upper	50	3100
2	1	normal	150	1550
3	2	upper	50	3100
4	2	normal	150	1550
5	3	upper	50	3100
6	3	normal	150	1550
7	4	upper	46	1900
8	4	normal	150	950
9	5	upper	50	1900
10	5	normal	150	950
11	6	upper	48	1500
12	6	normal	150	800
13	7	upper	48	1300

Table 2: Database changed in client1

Seat _id	Schedule _ id	Seat type	Available seat	price
1	1	upper	50	3100
2	1	normal	150	1550
3	2	upper	50	3100
4	2	normal	150	1550
5	3	upper	50	3100
6	3	normal	150	1550
7	4	upper	46	1900
8	4	normal	150	950
9	5	upper	50	1900
10	5	normal	150	950
11	6	upper	48	1500
12	6	normal	150	800
13	7	upper	48	1300

Table 3: Database changed in server

Seat _id	Schedule _ id	Seat type	Available seat	price
1	1	upper	50	3100
2	1	normal	150	1550
3	2	upper	50	3100
4	2	normal	150	1550
5	3	upper	50	3100
6	3	normal	150	1550
7	4	upper	46	1900
8	4	normal	150	950
9	5	upper	50	1900
10	5	normal	150	950
11	6	upper	50	1500
12	6	normal	150	800
13	7	upper	48	1300

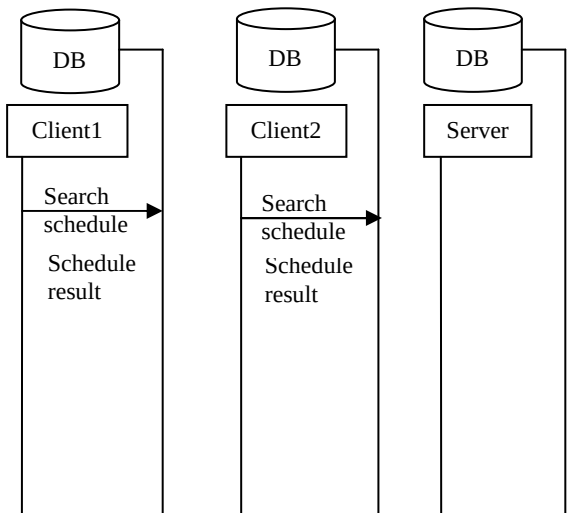
Table 4: Database unchanged in client2

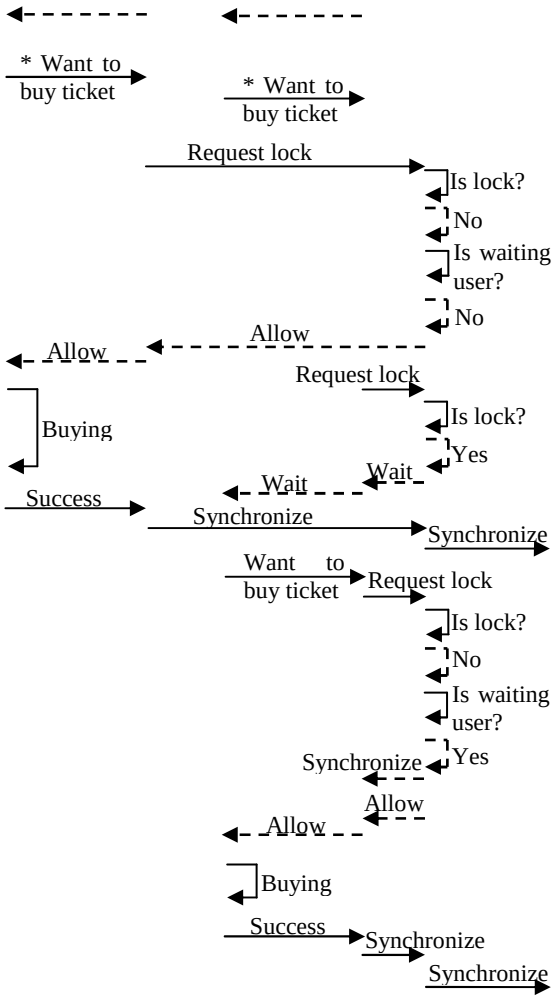
If waiting user in queue, server synchronizes to the waiting user (client2). Thereupon client2 database changed as shown in table 5.

Seat _id	Schedule _ id	Seat type	Available seat	price
1	1	upper	50	3100
2	1	normal	150	1550
3	2	upper	50	3100
4	2	normal	150	1550
5	3	upper	50	3100
6	3	normal	150	1550
7	4	upper	46	1900
8	4	normal	150	950
9	5	upper	50	1900
10	5	normal	150	950
11	6	upper	48	1500
12	6	normal	150	800
13	7	upper	48	1300

Table 5: Database changed in client2

2.11. Sequence diagram for system workflow





This system use vector timestamps to indicate which user to allow for data modification. In this sequence diagram, client1 is doing first event and client2 is doing second event at star (*). In this system, maximum time event is first event doing. So, first event is allowed for buying and second event is waiting.

3. Conclusion

This system aim to show how consistent data can be in shared data by updating multiple users. User may have up-to-date information by means of this proposed system. This system is essential for Myanmar railway transportation in Yangon to Mandalay, Naypyitaw, Mawlamyaine, and Pyay because it save time, voluminous paper work and avoids human errors. This system can be effectively used for the staff on an existing system during calculating and maintaining.

4. References

[1] George Colouris, Jean Dollimore, Tim Kindberg: " Distributed System Concepts and Design", Third edition, www.personedu.com, Person Education Ltd, Edinburgh Gate, Harlow, Essex CM20 2JE England, ISPNO 0-201-61918-0, 2010.

[2] Alan L. Cox , Eyal de Lara , Charlie Hu , and Willy Zwaenepoel, "A Performance Comparison of Homeless and Home-based Lazy Release Consistency Protocols in Software Shared Memory"

[3] L.Iftode, "Home-based shared virtual memory", PhD thesis, Department of Computer Science, Princeton University, June 1998.

[4] Vadim Iosevich and Assaf Schuster, "Multithreaded Home-based Lazy Release Consistency over VIA"

[5] Andrew S.Tanenbaum, and Maarten Van Steen, "Distributed systems principles and paradigms", International edition, Prentice Hall, New Jersey, 2002.