Implementation of Client-Centric Consistency Control and Synchronization System For Distributed Replication(For Mobile Clients)

Aye Nyein Mon, Thinn Thu Naing
University of Computer Studies, Yangon
ayenyeinmonlay@gmail.com, ucsy21@most.gov.mm

Abstract

Distributed System is a collection of independent computers that appears to its users as a single coherent system. Distributed processing has become more and more important and attractive with progress in the areas of computer networks and distributed system. A consistency model is a contract between processes and data store. If processes agree to obey certain rules, the data store promises to work correctly. Normally, a process that performs a read operation on a data item, expects the operation to return a value that shows the results of the last write operation on that data. In the absence of a global clock, it is difficult to define precisely which write operation is the last one. This leads to a range of consistency models. This paper intends to develop client-centric consistency model by using monotonic-write consistency control and vector clock time synchronization algorithm.

1. Introduction

An important challenging in distributed system is the consistency for replication. This means that when one copy is updated, we need to ensure that the other copies are updated as well; otherwise the replicas will no longer be the same. To achieve high performance of operations on shared data, designers of parallel computers have paid much attention to different consistency models for distributed shared memory systems. Consistency models for shared data are often hard to implement efficiently in large-scale distributed systems. Moreover, in many cases, simple models can be used, which are also often easier to implement. There are several consistency models that try to find the best trade-off between protocol efficiency and ease of use. In general, existing models may be divided into two groups: data-centric and client-centric. Data-centric models specify the restrictions imposed on the order in which data is updated on individual's servers. There are several data-centric consistency models defined mainly as a result of research in the field of Distributed Shared Memory (DSM), e.g.: atomic consistency, sequential consistency, causal consistency, processor consistency, PRAM consistency. Client-centric consistency models specify the requirements concerning data consistency that are based only on the history of interaction between individual clients and the system (servers).

The structure of this paper is as follows. Section 2 presents motivation of the system. Section 3 describes consistency models. Section 4 indicates vector clock time synchronization algorithm. System overview is explained in section 5. In section 6, implementation of the system is given. Conclusion is shown in section 7 and references are given in section 8.

2. Motivation

There are many examples in which consistency appears only in a distributed form. In many database systems, most processes hardly ever perform update operations but only read data from the database. Only one, or very few processes perform update operations. As another example, consider a worldwide naming system such as DNS. The DNS name space is partitioned into domains, where each domain is assigned to a naming authority, which acts as owner of that domain. Only that authority is allowed to update its part of the name space. These examples can be viewed as a high tolerance for inconsistency. If no updates take place for a long time, all replicas will gradually become consistent. This form of consistency is called eventually consistency. Eventual consistency requires only that updates are guaranteed to propagate to all replicas. Write-write conflicts are easy to solve since a small number of processes can perform updates. Therefore, this consistency is cheap to implement. Eventual consistent data stores work fine as long as clients always access the same replica. However, problems arise when different replicas are accessed. This problem can be alleviated by introducing client-centric consistency. In essence, client-centric consistency model provides guarantees for a single

client concerning the consistency of accessing to a data store by that client. No guarantees are given concurrent accesses by different clients.

3. Consistency Models

A consistency model defines which interleaving of operations (i.e., total orderings) are acceptable (admissible). A data store that implements a particular consistency model must provide a total ordering of operations that is admissible.

In general there are two types of consistency models. They are data-centric consistency model and client-centric consistency model.

3.1 Client-Centric Consistency Model

Data-centric consistency aims to provide the system-wide consistency view of the data store. When there are concurrent updates by many processes, data-centric models are used. Consider the special case where there are no concurrent updates or if they occur they can easily be resolved. Client-centric consistency models provide consistency guarantees from a single client. Client-centric consistency models originate from the work on Bayou. Bayou is a database system developed for mobile computing, where it is assumed that networked connectivity is unreliable and subject to various performance problems. Wireless networks and networks that span large areas, such as the Internet, fall into this category. Bayou essentially distinguishes four different consistency models.

(1) Monotonic-Read
(2) Monotonic-Write
(3) Read Your Writes
(4) Write Follow Reads

In essence, client-centric consistency provides guarantees for a single client concerning the consistency of accesses to a data store by that client. No guarantees are given concurrent accesses by different accesses.

3.1.1 Monotonic-Write

In many situations, it is important that write operations are propagated in the correct order to all copies of the data store. This property is expressed in monotonic-write consistency. In a monotonic-write consistent data store, the following condition holds;

A write operation by a process on a data item x is completed before any successive write operation on x by the same process.

Thus completing a write operation means that the copy on which a successive operation is performed, reflects the effect of a previous write operation by the same process, no matter where that operation was initiated. In other words, a write operation on a copy of data item x is performed only if that copy has been brought up to date by means of any preceding write operation, which may have taken place on other copies of x.

Monotonic-write consistency is shown in Figure 1. In Figure 1, process P performs a write operation on x at local copy L_1, presented as the operation $W(x_1)$. Later, P performs another write operation on x, but this time at L_2, shown as $W(x_2)$. To ensure monotonic-write consistency, it is necessary that the previous write operation at L_1 has already been propagated to L_2. This explains operation $W(x_1)$ at L_2, and why it takes place before $W(x_2)$.

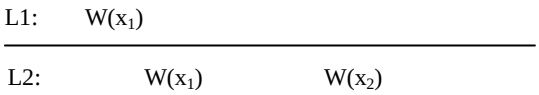

Figure 1. A monotonic-write consistent data store

4. Vector Clock Time Synchronization Algorithm

Lamport timestamps do not guarantees that if a.TS < b.TS that a indeed happened before b. Given two events, we cannot say whether they are causally related from their timestamps. Vector Timestamps is needed. Vector clock is a vector of integer numbers maintained at each process in a group of communication processes. The size of the vector is equal to the number of processes in the group. The timestamp is updated using three rules:

- Whenever a process i observes a significant event, it increments the i^{th} element of its timestamp.
- Whenever a process sends a message to another process, it also adds its timestamp to the message.
- Whenever a process receives a message from another process, it also adjusts its timestamp so that all its elements are larger than the corresponding elements of the timestamp in the message but not smaller than their previous values.

Timestamps can be compared to each other. Timestamp A is smaller (or larger) than timestamp B if each element of A is numerically not larger (or not smaller) than the corresponding element of B, and at least one element of A is numerically smaller (or larger) than the corresponding element of B. Timestamp A is unrelated to timestamp B if it is neither smaller nor larger.

5. System Overview

5.1 Overview of the System

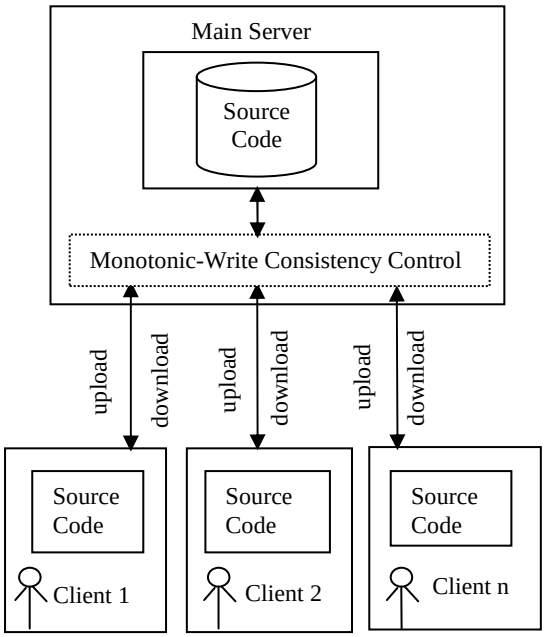

Figure 2. Overview of the system

The source code is uploaded to the main server. Each data item has an associated owner. Each client updates to his own source code data without affecting to other user's data. To ensure the monotonic-write consistency control, updated codes (write operations) are propagated in the correct order to the clients at several sites. Whenever a client initiates a new write operation at a server, the server is handed over the client's write set. It ensures that the identified write operations are performed first and in the correct order. After performing the new operation, the operation's write identifier is added to the write set. With monotonic-write consistency, guarantees are given that if an update is performed on a copy of the source code data, all preceding updates are performed first. The resulting source code becomes the most recent version and includes all updates that have led to previous versions of the source code.

5.2 Client and Server Side Algorithms

5.2.1 Client-Side Algorithm

Upon sending a request to server S_j at client C_i,

Initially W=0;
If (isWrite(op)) then
W[i]=Wc_i[i];
Else
W=max(W,Rc_i);
End if;
Send <op,W>to S_j

Upon receiving a reply from server S_j at client C_i,

If (isWrite(op)) then
Wc_i[i]=Wc_i[i]+1;
Else
Rc_i=max(Rc_i,W);
End if;

5.2.2 Server-Side Algorithm

Upon receiving a request<op,W> from client C_i at server S_j,

If (Vs_j<W) then
Perform previous write operation
Else
Perform new write operation
Vs_j[i]=Vs_j[i]+1;
Timestamp operation with Vs_j
Send <op,Vs_j> to C_i
End if;

Note: W is a vector representing a write
Wc_i is a vector representing all writes issued by the client
Rc_i is a vector representing writes relevant to reads issued by the client.
Hs_j is a history of the writes
Vs_j is a vector representing all write performed by the server

5.3 Sequence Diagram for Code synchronization

Use Case (from Client 1)

Firstly, client 1 logs in and downloads his own source code from the main server. Client 1 gets not only his source code but also his peer user's source code. Client 1 updates to his own source code without affecting to other user's data. Then, client 1 uploads the updated source code to the main server at time t1. The main server gets the updated source code with the latest updated time. And then, client 1 synchronizes the updated source code to all other active users. So, the other active users can get the updated source code of peer users.

Use Case (from Client 2)

Client 2 also logs in and downloads his own source code from the main server. Client 2 gets not only his source code but also his peer user's source code. Then, client 2 updates his source code. And

then, client 2 uploads the updated source code to the main server at time t2. The main server gets the updated source code with the latest updated time. And then, client 2 synchronizes the updated source code to all other active users. So, the other active users can get the updated source code of peer users.

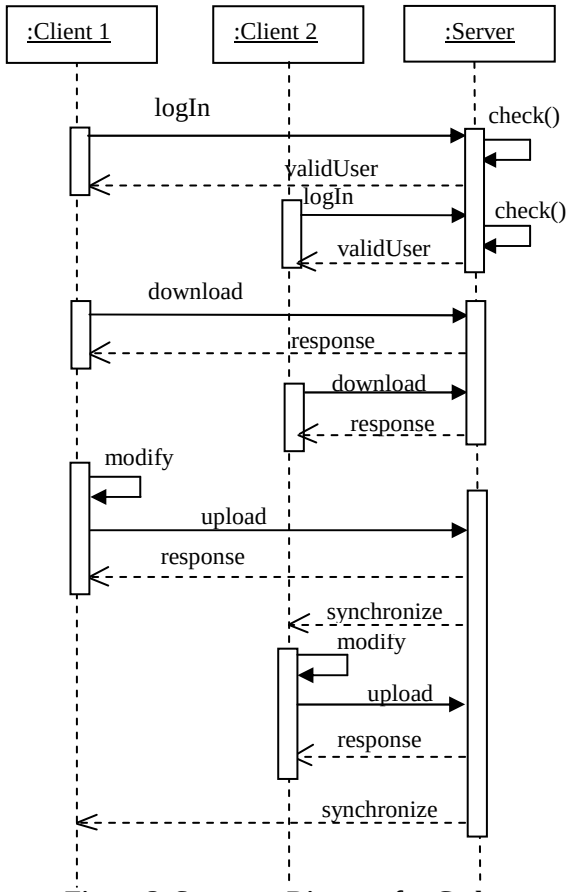

Figure 3. Sequence Diagram for Code Synchronization

6. Implementation of the System

As a case study, two clients are developing a window form application (calculator application) together. But, this system can use multiple users. Client 1 is developing form design and calling function associated with the buttons. Client 2 is developing function implementation associated with the buttons. Initially, this form includes three textboxes and three buttons as shown in figure 4.

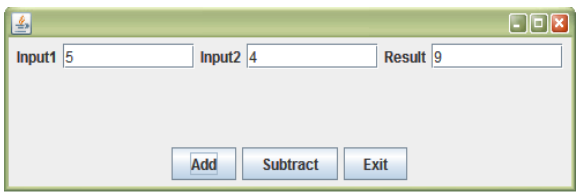

Figure 4. Calculation Form

Each client has own source code data. Firstly, if a client wants to download source code data, he fills in the login box. If the user name and password are correct, the client can download the source code.

Client 1 logs in and downloads his source code from the main server at a time. Client1 also gets the source code of his peer user, client 2. Client 2 also logs in and downloads his source code from the main server at a time. Client 2 also gets the source code of his peer user, client 1.

Client 1 updates his own source code by adding the new button to this form and calling function associated with the new button. Then, he uploads the updated source code to the main server at a time. Client 2 updates his own source code by adding new function implementation. Then, client 2 uploads his updated source code to the main server at a time. Client 1 synchronizes his updated source code to the active user, client 2. Client 2 also synchronizes his updated source code to the active user, client 1. If the updated source code is successfully synchronized, the message, “Synchronization successful”, is given to the clients. After receiving the message, client 1 compares timestamps whether the timestamp of old version of source code data is smaller than the timestamp of new version of source code. If so, client 1 refreshes the code with the peer user’s new source code. Client 2 also compares the timestamps and refreshes the code with the peer user’s new code. Finally, clients at several sites get the newest version of source code with the latest updated time.

To be able to run the updated source code, client 1 and client 2 save source code data. If the clients run the updated source code, the clients at several sites can get the updated result as shown in figure 5.

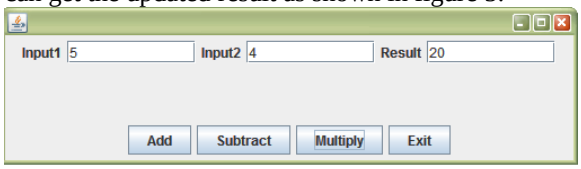

Figure 5. Update Calculation Form

At the server site, the timestamps must be compared to each other whether the previous uploaded time is smaller than the current uploaded time. Then, clients’ current update log information is saved to the server. This includes user name, the update time and the update code at each time as shown in figure 6. The show updated source link shows the updated source code by the clients. So, clients can see the updated source code at each time.

Name	Update Log Information	
mama	2010-09-21 : 10:22:30	Show updated source
ayeaye	2010-09-21 : 10:23:38	Show updated source
mama	2010-09-24 : 12:40:18	Show updated source

ayeaye	2010-09-25 : 12:49:02	Show updated source

Figure 6. Main Server site's Information

7. Conclusion

In this paper, client-centric consistency control by using vector clock is presented. Client-centric consistency is characterized by the lack of simultaneous updates. The emphasis is more on maintaining a consistent view of things for the individual client process that is currently operating on the data-store. So, there are no write-write conflicts because each data item has an owner who is allowed to update the data. By using monotonic-write consistency control, write operations are propagated in the correct order to all copies of the data store. By using vector clock, consistency on shared data is more to be synchronized. This system helps mobile clients can get up-to-date data from different sites at different times.

8. References

[1] A.S.Tanenbaum and M.van Steen, "Distributed Systems Principles and Paradigms", Prentice Hall, New Jersey, 2002.

[2] C.J.Fidge, "Fundamentals of distributed computing systems, "IEEE computer" r, 24(8): 28-33, August 1991.

[3] George Coulouris, Jean Dollimore, Time Kindberg, "Distributed System Concepts and Design", 3rd Edition, ISBN 0201-61918-0, 2001.

[4] Jerzy Breze Zinki, Cezary Sobaniec, Dariusz Waw rzyniak, "From Session Causality to Causal Consistency", Institute of Computing Science, Poznan University of Technology Piotrowo 3a, 60-965 Poznan, Poland.

[5] M.Spreitzer, D.B.Terry, A.J.Demers, K.Petersen, M.Theimer, and B.W.Wetch, "Session guarantees for weakly consistent replicated data", In proceedings of the Third International Conference on Parallel and Distributed Information Systems (PDIS 94), Austin, Texas, September 28-30, 1994, pages 140-149, 1994.