Co2Cloud: A Consistency Model for Collaborative Works on Cloud

Yin Nyein Aye, Thinn Thu Naing
University of Computer Studies, Yangon
yinnyeinaye.ptn@gmail.com, thinnthu@gmial.com

Abstract

Consistency maintenance is an important issue in collaborative work systems that are activated in both traditional distributed system and cloud system. Collaborative Works are computer based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared context. Consistency maintenance of shared documents under the constraints of short response time and support for free and concurrent editing in distributed environment is one of the fundamental and challenging issues. In this paper, we describe a consistency model for constructing collaborative software development on cloud.

1. Introduction

Cloud computing (CC) is a computing technology that uses the Internet and central remote servers to maintain data and applications. CC allows people to use applications without installing them on their computers and allows access to saved files from any computer with an Internet connection. CC technology involves more efficient computing by centralizing storage, memory, processing and bandwidth to simultaneously work on a project-regardless of their location because the services and storage are provided over the Internet (or cloud).

An emerging class of cloud-based collaborative services, such as online document processing provides users with anywhere available and concurrent access to shared state. Collaborative work (CW) means that group works in a common task. It is used mainly in the business settings and is now aided by computers, which is known as computer supported collaborative work (CSCW). Its purpose is to facilitate group communication and productivity [4]. Nowadays, the implementation of CSCW on cloud is a challenging issue among researchers in various fields such as artificial intelligence, computer science, network communication, distributed systems and so on. Shared objects are replicated on different sites. Each user works on his own copies. This implies the divergence of different copies. The convergence of data on different copies does not necessarily mean a consistent state. CSCW applications may have a huge variation in requirements for reliability and consistency. Moreover, the consistency of the data not only depends on the local operations but also on the operations taken on the other copies.

Many types of consistency over the past 30 years and a wide variety of consistency models have been explored in the computer science research community, many of these are tried to specific implementations. Frequently, one needs to understand how a system operates in order to understand what consistency it provides in what situations. The need for different consistency levels is depended on in a variety of applications. This paper is reported not all data needs to be treated at the same level of consistency and use cases in which consistency could be applied. [10]

There are so many consistency models for Cloud based Applications. These are the following consistency models;

- Google App Engine Datastore- read-your-writes eventual consistency
- Amazon S3 storage -eventual consistency
- Azure Table and Blob Storage -strong data consistency
- SimpleDB- Monotonic Write Consistency, Inter-Element Consistency

Many cloud data storage platforms (or particular operations within a platform) use techniques to achieve high availability and low latency that avoid two-phase commit and/or synchronous access to a quorum of sites. [15] Thus they can't guarantee strong consistency. In the cloud environment, replicated architecture is widely adopted in collaborative systems which are to meet the requirement of high responsiveness. Shared documents are replicated at the local storage of each collaborating site, so that operations can be performed at local sites immediately and then propagated to remote sites. Most of the collaborative applications require several functionalities. [6] First, better coordination among users, there should be mechanisms so that users are generally aware of what other participants are doing. Second, a concurrency control mechanism should be provided for keeping the shared data consistent even though users may attempt to make simultaneous, conflicting changes. Third, because of the interactive nature of a collaborative application, it should ensure interactive response time to users' actions while maintaining shared data consistency. In distributed systems describes alternative consistency models (e.g., eventual consistency, read-write monotonicity, or session consistency). [13]

This research will address the problem of consistency by proposing a model which considers on software development. The remaining paper is structured as follows. Section 2 presents proposed system architecture, section 3 explains consistency model for target scenario and section 4 describes related work. Finally, section 5 addresses the conclusion and future work.

2. Proposed System Architecture

As Figure1 shows collaborative works on cloud consists of four parts.

1) Developing Workgroup: developers in different areas prepare writing code together and give different views on the same topics.

2) Management Workgroup: managers in different roles collaborate with each other to work which make them to cooperate with others.

3) Outsourcing Workgroup: Developers in outsourcing complete the task with connection of managers which manage the task to finish successful.

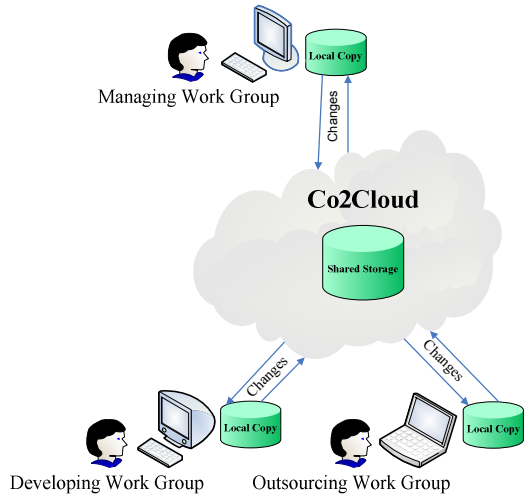

Figure1. System Architecture

4) Shared storage: developers, managers and developers in outsourcing are encouraged to complete the task with consistent state.

Multiple views of software development can be synchronously, semi-synchronously and

asynchronously edited by different developers. View versions can be incrementally merged, and view updates broadcast to other developers and incrementally incorporated as required in their alternative versions. Consistency management is required to keep all of these views consistent under change.

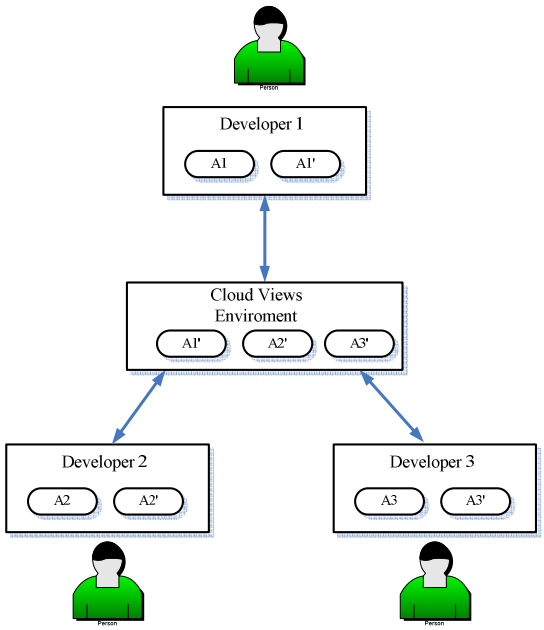

Figure2.View editing on cloud

Developers may create and modify new versions of a component based on their current version. The current version (i.e. new changes are locked out) and allows it to be exported for other developers to use. Another developer may subsequently import and merge. Figure 2 illustrates editing and merging approach to collaborative development on cloud. Developer 1 edits data (denoted by A1') and Developer 2 edits data (denoted by A2'). Developer 3 also edits data (denoted by A3'), so they can modify it at the same time. After updating, developer 1, 2 and 3 freeze their edit data. In cloud environment, the last edit data (A1', A2', A3') is exported for other developers to use.

Basic workflow of software development on cloud is shown on figure 3.

1) First, developer retrieve data that he/she want to edit from cloud.

2) Edit the data.

3) When he/she wants to upload his/her edited data, first he/she must check to data on cloud storage that may be changed by other persons.

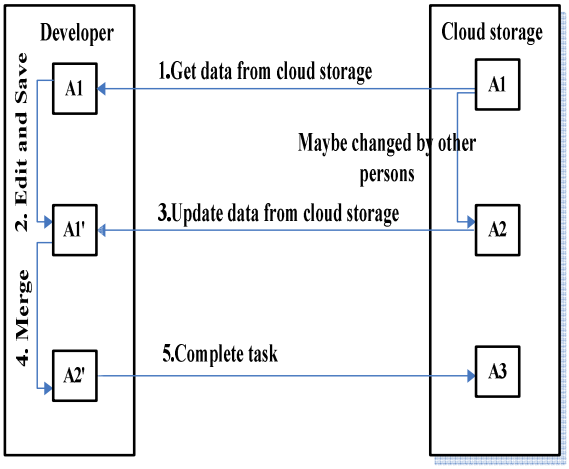

Figure3.Basic Workflow of software development on cloud

4) Merge data from cloud storage and edited data.

5) Complete task to send cloud.

3. Consistency Model

Maintaining consistency within a single database node is relatively easy for most databases. The real problems start to surface when you try to maintain consistency between multiple database servers.[13] Collaborative Editing allows people to work simultaneous on the same document or source base (e.g., Google Docs, Version control, Wiki's). The main functionality of such a system is to detect conflicts during editing and to track the history of

changes. Traditionally such systems work with strong consistency. Most parts of the document which are frequently updated by several persons would be best handled by strong consistency guarantees to avoid conflicts all together. If a client makes a write operation on server A, we do not make sure that this is consistent with server B, or C, or D. Therefore, distributed shared systems are designed as different consistency models to achieve high performance of operations on shared data. Consistency models for shared data are often hard to implement efficiently in large-scale distributed systems. Moreover, in many cases simpler models can be used which are also often easier to implement.

$$\begin{array}{ccc} [\ \text{site1 state}\ (t)\ ] & \xrightarrow{\text{map1}} & [\ \text{site2 state}\ (t)\ ] \\ \downarrow \text{op1} & & \downarrow \text{op2} \\ [\ \text{site1 state}\ (t+1)] & \xrightarrow{\text{map2}} & [\text{site2 state}\ (t+1)] \end{array}$$

At the time t+1, the user interacts with the application at site 1 (“op1”) and it transitions to a new state. To maintain consistency, site 2 must also transition to a new state which is consistent with that of site 1. We could either apply an operation “op2” to the old state of site 2 or a mapping “map2” to the new state of site 1. [9]

A consistency model is a contract between processes and the data store. It says that if processes agree to obey certain rules, the store promises to work correctly. We need to consider how consistency is actually implemented. Two issues play a role to keep consistent. The first issue is the actual distribution of updates and how updates are propagated. The second issue is how data items are kept consistent. In most cases, applications require a strong form of consistency. There are various alternatives for implementing consistency. [14]

3. 1 Consistency Model for target scenario

In the target scenario, a shared document is replicated at multiple sites connected by cloud. The user at each site can update his/her local data by issuing add, delete and undo operations anytime and anywhere. Local updates are executed immediately for fast response time. We want the results of operation performed to be consistent possibly being read and update concurrently by many developers. Two main operations are considered: Read and Write. The Read operation represents a query over the contents. The Write operation updates data. A Write may involve creating, modifying or deleting data items.

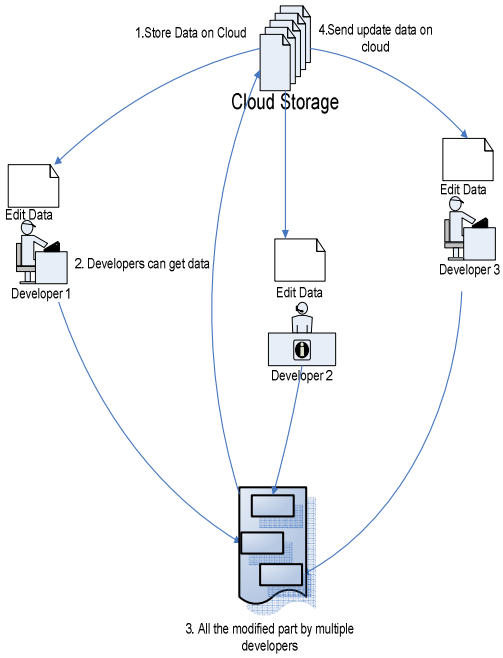

Figure4. Target Scenario

In the figure 4, it is important that write operations are propagated in the correct order to all copies of the data store. A write operation by a developer on a data item (A) is completed before any successive write operation on A by the same process (i.e. a wrote operation on a

copy of data item is performed only if that copy has been brought up to date by mean of any preceding write operation, even if taken place on another copy of A.

Consider a single source code file containing 10 functions. The developer1 updates function number 1 at developing work group. Then developer 2 updates function number 7 in management group. Each of these updates represents a different version. The developer1 should observe the changes to function1 in source code. If an update is performed on a copy, all preceding updates will be performed first. The monotonic-write consistency resembles data-centric FIFO consistency. The essence of FIFO consistency is that write operations by the same process are performed in the correct order everywhere. But monotonic write consistency is used for a collection of concurrent processes.

If a single data item is written in on location then a new value of that single data item is written in a different location, then this problem doesn't really occur. Monotonic-write consistency is shown in Figure 5. In this target scenario for updating software source code (SC) performs a write operation on local copy (lc1) at developer 1 site presented as the operation Write (lc1).SC performs another write operation on lc2 at developer 2 site, shown as Write (lc2). And then SC also performs another write operation on lc3 at developer 3 site, described as Write (lc3).

By using monotonic write consistency, it is shown that a write operation by a SC process on a local data item local copy (lc) is completed before any successive write operation on lc by the same process. It implies a copy must be up to date before performing a write on it. Write operations by the same process are performed in the same order no matter where that operation was initiated.

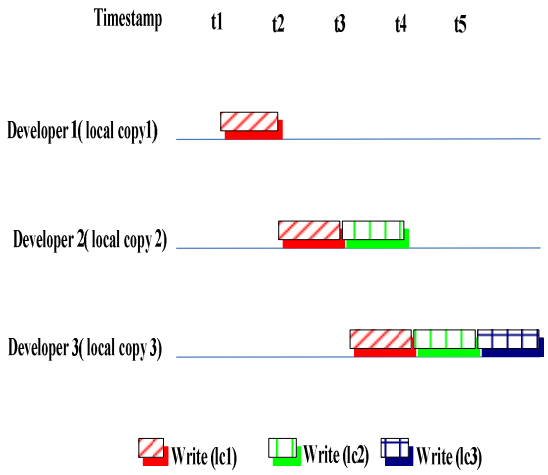

Figure5. Monotonic-write Consistency

3.2 Algorithm for Monotonic Consistency Model

Monotonicity means that if $x_i \geq y_i$ for all i, then $f_{op}(x_1; : : : ; x_k) \geq f_{op}(y_1; : : : ; y_k)$. It satisfies a weaker but natural consistency condition, called monotonic consistency. A write operation by a process on a data item x is completed before any successive write operation on x by the same process.

Definition 1

A task is a set of data in cloud. Suppose M is the set of total data in cloud and T_i is a subset task of M. Suppose there are t tasks $t_1,t_2,\ldots,t_n$ total in cloud and M is the set of total data then

$$\sum_{i=1}^{n} T_i = M \qquad (1)$$

Table1. Definition of Notations

Notation	Definition

op	Operation
k	Input
f_{op}	Function of operation
S	Number of update operations
d	Number of data items
v	Value

In table 1 is shown by the definition of notations for monotonic consistency algorithm.

Algorithm: Monotonic Consistency

```
1.  procedure Update(op)
2.      begin
3.          Let x1,....xd be the inputs to op.
4.          for i←1 to d do
5.              yi← Read(xd)
6.          end
7.      Write (op,fop(y1,...,yd)
8.  end

9.  procedure Write(op,v)
10.     begin
11.         Write(op,v)
12.         Let op1,..,opi be update operations
13.             for i ←1 to S do
14.                 Update(opi)
15.     end
16. end

17.  procedure Read(op)
18.     begin
19.         return Read(op)
20.  end
```

In figure 6 is shown to complete the task with monotonic write consistency. Developer 2 starts working on a class PASSAGE. He/she get data as a local copy from Co2Cloud. Co2Cloud displays the class current version at developer 2 site. And then he/she starts modifying the class. In the meanwhile developer 1 starts working on the same PASSAGE class. Co2Cloud makes integration the two write operations by two developers. There is no need for complex communication because of using monotonic write consistency both developers are aware of which parts have been changed.

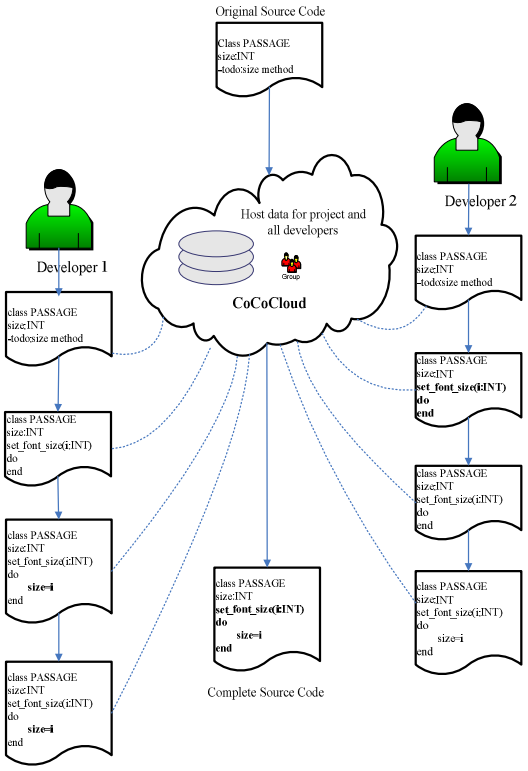

Figure6. Co2Cloud

4. Related Work

Distributed consistency constraints either ensure strong consistency or weaken the consistency in small intervals which in turn can lead to inconsistencies. A great deal of work has been done on distributed consistency constraint and limited divergence of replica. In this paper [14] develops and analyzes a transaction management and replication protocol based on implementation of the Paxos with Combination

and Promotion (Paxos-CP) that provides true consistency. In [3] describes four per-session guarantees are proposed to aid users and applications of weakly consistent replicated data. These session guarantees present individual applications with a view of the database that is consistent with their own actions, even if they read and write from various, potentially inconsistent servers. Google's BigTable [5] provides eventual consistency guarantees. Recently, some research efforts have been focused on providing stronger guarantees: Yahoo's PNUTS [7] provides monotonicity guarantees and snapshot isolation on a per-record basis.

Grove [2], ORESTE [9], COAST [12] and DECAF [13] use optimistic concurrency control with system guaranteed state consistency. The ORESTE algorithm considers a shared document consisting of a set of objects, which are fully replicated at all the participating processes. Each event modifies exactly one shared object. Locally generated timestamps are used to order all the events and information about community and masking of events is used to minimize rollback.

Many papers have described particular architectures and algorithms for consistency. In this paper [15] describes what kinds of inconsistency are seen in the results returned from operations, and how frequently these situations arise. As previously mentioned, most collaborative applications are used in only distributed computing. CC allows people to use applications without installing them on their computers and allows access to saved files from any computer with an Internet connection. So we propose a consistency model for constructing collaborative software development on cloud to complete the task with fast access and without conflict.

5. Conclusion and Future Work

The developers are tried to express their tasks by using only monotonic operations to handle each changing the set of possible situations and what the code must be written in this target scenario to keep consistency. This paper has addressed the importance of efficiency when designing a consistency algorithm for use in software development on cloud. We propose an algorithm that is more efficient and suitable for use in this research. For future work, we are planning to further investigate issues concerning consistency and system performance.

References

[1] E.A. Brewer. "Towards robust distributed systems (abstract)", in: PODC Conf., ACM, Portland, OR,USA, 2000, pp. 7.

[2] C. A. Ellis and S. J. Gibbs. "Concurrency control in groupware systems". In Proceedings of the ACM SIGMOD'89, pages 399–407, 1989.

[3] D.B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer,and B. B. Welch, "Session Guarantees for Weakly Consistent Replicated Data" ,Computer Science Laboratory,Xerox Palo Alto Research Center Palo Alto, California.

[4] G.Convertino, U.Farooq, M.B Rosson, and J.M. Carroll. "Old is Gold: Integrating Older Workers in CSCW", Proceedings of the 38th Hawaii International Conference on System Sciences, 2005, pp. 1-10.

[5] F. Chang et al. " Bigtable: A Distributed Storage System for Structured Dat"a. In Proc. of OSDI, pages 205–218, 2006.

[6]G.DeCandia,D.Hastorun,M.Jampani,G.Kakulapati, A.Lakshman, A.Pilchin,S. Sivasubramanian, Peter Vosshall and Werner Vogels, "Dynamo: Amazon's Highly Available Key-value Store",SOSP'07, October 14–17, 2007, Stevenson, Washington, USA.

[7] B. F. Cooper et al. "PNUTS: Yahoo!'s hosted data serving platform". In Proc. of VLDB, volume 1, pages 1277–1288, 2008.

[8] S.Greenberg and D.Marwood , " Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface", In Proceedings of the ACM Conference on

Computer Supported Cooperative Work, pp.207-217 , Chapel Hill, North Carolina.

[9] A.Karsenty and M.Beaudouin-Lafon. “An algorithm for distributed groupware applications”. In Proceedings of the 13th ICDCS, pages 195–202, 1993.

[10] T. Kraska, M. Hentschel, G. Alonso, D. Kossmann. “ Consistency rationing in the cloud: pay only when it matters”, Proceedings of the VLDB Endowment (PVLDB) 2(1) (2009) 253–264.

[11] I. Marsic*, X. Sun†, C. Correa*, and T. Liu‡ *Department of Electrical and Computer Engineering and the CAIP Center Department of Mathematics, ‡Rutgers Center for Operations Research (RUTCOR) , “Maintaining State Consistency Across Heterogeneous Collaborative Applications” , Rutgers University, Piscataway, NJ 08854.

[12] C.Schuckmann, L. Kirchner, J. Schummer, and J. M. Haake. “Designing object-oriented synchronous groupware with COAST”. In ACM CSCW'96, 1996.

[13] A. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002

[14]S.Patterson,A.J.Elmore ,F.Nawab,D.Agrawal,A.El Abbadi, “Serializability, not Serial: Concurrency Control and Availability in Multi-Datacenter Datastores”,August 27th - 31st 2012, Istanbul, Turkey.

[15] H.Wada, A.Feketez, L.Zhaoy, K.Lee and A.Liu, “Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: the Consumers’ Perspective”,CIDR’11 Asilomar, California, January 2011.