Cloud Assisted File Sync

CS 219 Current Topics in Cloud Computing – Course Project Report

Prashant Rajput
8047441669
prashanthrajput@ucla.edu

Chidambaram Muthappan
704774938
cmuthapp@ucla.edu

Jeya Vikranth Jeyakumar
404749568
vikranth94@ucla.edu

Liwen Wen
504160412
wenliwen64@ucla.edu

ABSTRACT

Cloud Storage Services have become very popular in the recent years. In this project we identify few problems with the standard file hosting services and detail solutions for the same. We have designed our own cloud sync service to demonstrate the feasibility and usefulness of our ideas. The major issues identified are redundant data sync, cost to storage and large network utilization between client and server. We solve the first problem through delta encoding and the last two by making the storage peer to peer. Our experimental results is promising and shows that our key ideas are working.

KEYWORDS

Cloud Storage, Peer to Peer, Distributed Storage

ACM Reference format:
Prashant Rajput, Chidambaram Muthappan, Jeya Vikranth Jeyakumar, and Liwen Wen. 2017. Cloud Assisted File Sync. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 5 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In recent years there has been a growing public fascination with the Cloud Storage Systems. Cloud storage services, such as Dropbox , OneDrive (used to be SkyDrive) and GoogleDrive, are expanding their market daily.With a rush of providers to enter the market and an increasing offer of cheap storage space, it is to be expected that cloud storage will soon generate a high amount of Internet traffic. We witness a gold rush to offer on-line storage capabilities, with players like Microsoft, Google and Amazon entering the market at the end of April 2012. They face a crowded scenario against popular solutions like Box.com, UbuntuOne, and Dropbox. The latter, active since 2007, currently counts over 50 million users, uploading more than 500 million files daily.

Very little is known about the architecture and the performance of such systems, and the workload they have to face.Considering commercial offers, little is known, with most players providing proprietary solutions and not willing to share information. So in this paper we first understand the underlying typical architecture of cloud storage systems.

The aim of this paper is to identify and address some of the problems of modern mobile cloud storage systems. Our work consists of two major components: 1) identifying the performance bottlenecks 2) proposing a new cloud storage service framework which integrates a few techniques to enable efficient sync operations in cloud storage services.

2 BACKGROUND

A typical architecture of cloud storage services as shown in figure 5, which includes three major components : the clients, the control server and the data storage server. The file system on the server side has an abstraction different from that of the client. Metadata (including the hashes, modified time etc.) and contents (often split into chunks) of user files are separated and stored in the control and data storage servers respectively. The key operation of the cloud storage services is data sync, which automatically maps the changes in usersâĂŹ local file system to the cloud via a series of network communications. During the sync process, metadata are exchanged with the control server through the metadata information flow, while the contents are transferred via the data storage flow. In a practical implementation, the control server and the data storage server may be deployed in different locations. For example, Dropbox builds its data storage server on Amazon EC2 and S3. Another important flow, namely notification flow, pushes notifications to the client once changes from other devices are updated to the cloud.

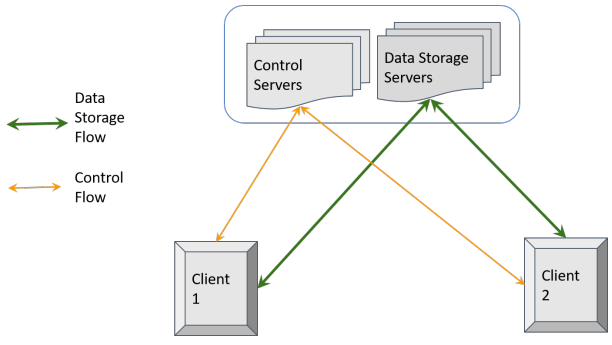

Figure 1: Typical Architecture of a Cloud Storage System

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3 DESIGN

3.1 Key Ideas

We have two key ideas in our project. The first major idea is to make data transfer peer to peer. We eliminate the need for data to be stored in the server. Only a small amount of metadata is stored in the control server and this greatly reduces the need for having a server with high storage capacity. So the clients synchronize with each other by transferring data between themselves using SCP. This also reduces the network traffic between the clients and the server. Secondly, we use delta encoding to synchronize the data. This means that each time a change is made the whole file doesnâĂŹt have to be exchanged. Instead only the new changes made is exchanged and is used for synchronization. This greatly decreases the size of data being exchanged.

3.2 Architecture

Our design consists of a Control server and a number of clients connected to the server. The control server stores only the metadata and provides the control signals to the clients for synchronization. The clients pull the delta file generated using SCP and get synchronized among themselves. We also have implemented a lock mechanism which prevents more than one client syncing at the same time. We have a server process running in all the clients and the control server and the client process which runs only in then clients.

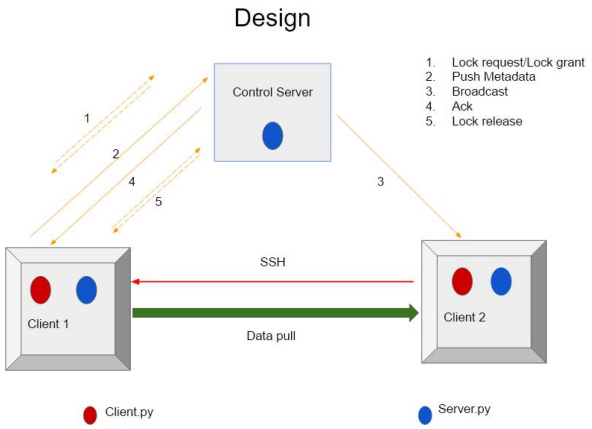

Figure 2: Design Architecture

4 IMPLEMENTATION

4.1 Clients

Clients contain two separate folders to aid in synchronization âĂŞ RemoteSync and Temp. RemoteSync is the folder which has the files to be synced across all the clients. All the intermediate files that are created for synchronization are stored in the Temp folder. Whenever a file is opened a copy of that particular file is created in the Temp folder before any changes is made. Also the delta file generated after the changes are made is stored in the Temp folder. The contents in the temp folder are cleared once the synchronization is complete. The client has client process and server process running in it and these processes will be explained in detail in section 2.3 and 2.4.

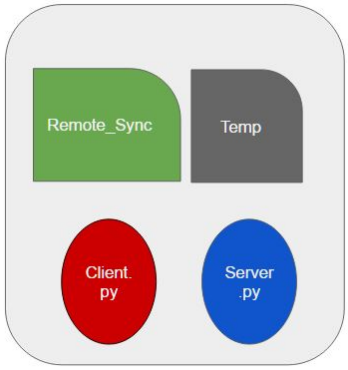

Figure 3: Client

4.2 Control Server

The control server consists of three small files âĂŞ Host, Sync, Timelog. The host file has a list of all the clients connected in the network along with their IP addresses and their SSH passwords. Sync has the list of all the files present in the RemoteSync folder and their corresponding lock flags. The Timelog file maintains a record of the filename, timestamp of when the edit was made and when the lock was released. The control server has the server process running on it which will be discussed in detail in section 2.4

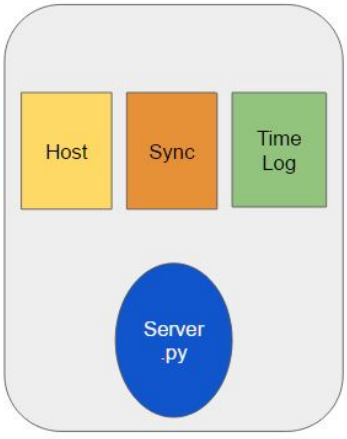

Figure 4: Control Server

4.3 Client Process

The client process runs on the client machine that is currently being used. It keeps monitoring the RemoteSync folder to see if

any file is being inserted, modified or deleted. It is responsible for automatically creating a copy of the file being modified and the delta file in the Temp folder. It is also responsible for informing Control Server about any changes to files in RemoteSync folder

4.4 Server Process

Server process is a multithreaded program which helps in isolating failures. It runs on all machines including the control server and all the clients and it uses port 2122. On the server side the server process is responsible for receiving and broadcasting control signals to the clients. It grants and releses locks to the clients based on their requests. It also creates the log entries in the Timelog file. On the client side the server process is responsible for listening to the instructions from the control server. It initiates the sync process between all the clients based on the broadcast signal from the control server.

4.5 Control Signals and Metadata

We have six different control signals in our design. Each signal has its own purpose.

- **Get lock**
  This signal is from Client to Control server and this is sent when any change is made to the files in the client.
  GET LOCK:FILENAME:CHECKSUM
- **Grant lock**
  This signal is from Control Server to the Client and is sent whenever the client requests a lock for a particular file and that file is not locked by any other client.
  GRANT LOCK:FILENAME:VERSION_NUMBER:CHECKSUM
- **Push**
  This control signal is sent from the client to the Control Server and it contains the path of the generated delta file and the IP address of the client.
  PUSH:CLIENT_IP:DELTA_FILE_PATH:CHECKSUM
- **Broadcast Metadata**
  The Control server broadcasts the delta file path and the particular clientâĂŹs IP address to the remaining clients along with the SSH username and password.
  PUSH:DELTA_FILE_PATH:CLIENT_IP:USR:PWD:CHECKSUM
- **Delete**
  Delete control signal is sent from Control server to the Clients only if a file is completely deleted from the RemoteSync folder
  DELETE:FILE_NAME:CHECKSUM
- **Release Lock**
  The client request the Control server to release the lock once the synchronization has completed
  RELEASE LOCK:FILENAME:CHECKSUM
- **Get Version List**
  Command used to get up-to-date version list from the master.
  GET FILE LIST:CHECKSUM
- **Sync File**
  Command used to Sync files that are not up to date. Meta data is sent to master in this step.
  SYNC FILE: FILE_NAME:VERSION_NUMBER:CHECKSUM
- **Send File**
  Master instructs a client to push a file to another client.
  SEND FILE: FILE_NAME:VERSION_NUMBER:TARGET_IP:USERNAME:PASSWORD:HASH
- **Ack**
  This is just a basic acknowledgement signal sent from Control server to clients to inform them that it has received their signals.
  ACK:CHECKSUM

4.6 Security

One of the loop holes in this implementation is the security of traffic between the clients and the control server. It could lead up to potential risks like man in the middle attack. So in order to create a secure tunnel and transfer our meta data we use sslv23 over the socket communication between control server and the clients. All the hosts including the clients and the server have the same certificates. Considering this is just a personal cloud sync system, self signed certificates are sufficient. The data transfer between two clients is done through ssh which is secured by in itself.

4.7 Network Partition

Network partition is one of the problem that is faced by almost all the distributed applications. We have created a robust method to overcome such network partition scenarios when the client is again connected to the Control server. There are two situations that may arise when a client is again connected to the Control server:

(1) Other client's may have created new files.
(2) The content of already existing files may have been modified.

For handling network partition, we came up with the following architecture:

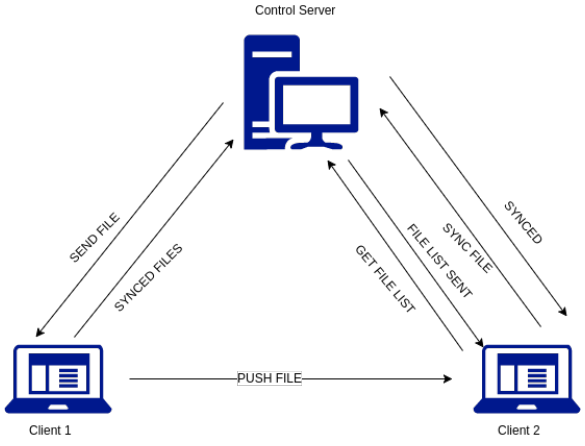

Figure 5: Network Partition

In this figure, assume that Client 2 just recovered from a Network Partition, the sequence of steps for recovery are as follows:

(1) GET FILE LIST: First, the Client 2 requests the up-to-date version log from the Control Server.

(2) FILE LIST SENT: The master then sends this version log which is parsed by the client 2 for discovering new files that were added and those which were updated.
(3) SYNC FILE: The client upon discovering a file that needs to be synced, sends a request to master for syncing.
(4) SEND FILE: The master then connects to the client that made the latest update to that file and asks it so push the updated file to client 2.
(5) PUSH FILE: Client 1 (Client who made the latest changes) pushes this updated file to Client 2.
(6) SYNCED FILES: Client 1 then informs the Control server that the file has been synced.
(7) SYNCED: The Control Server then informs the client 2 that the sync operation has been performed successfully.

5 FUNCTIONALITY

This section explains in detail the steps involved in synchronization. Assume that initially the files in the RemoteSync folder are synced between all the clients. When a file to be edited is opened in Client A, immediately a copy of the file is generated and stored in the Temp folder in the same client before any change is made. After we make the necessary changes and save the file, the client A asks the Control Server for a Lock. The Control server grants the lock and the synchronization process is initiated. Client A creates a Delta file in its Temp folder by comparing the initial copy stored in the Temp folder with the newly updated copy. This Delta file contains only the changes made to the edited file. The Client A then sends the push metadata to the Control server containing the path of the delta file and the client IP address. Control server once it receives the push signal from the Client A, broadcasts the metadata including the clientâĂŹs SSH username and password to the remaining clients to initialize the synchronization.

Once the remaining clients receive the broadcast message from the Control server, they SSH into Client A and pull the Delta file from the Temp folder. The Delta file is then used to synchronize their copies of the file to the latest version. The control server then sends the ACK message to Client A to indicate the completion of the Synchronization and then Client A finally asks the Control Server to release the lock on the file.

6 EVALUATION

6.1 Tests on File Types and Editors

We tested our system on various kinds of files (text, image, audio and video) and different editors. The result is shown in Table. 1. Strange file close behaviors by Gedit cannot be properly handled by our prototype. The times Gedit calls 'close()' is not predictable (sometime once, sometime twice). Everytime it calls twice the incorrect Δ data will be stored and synchronized, which results in corrupted synchronized file on other devices. So using a more robust way to detect file change becomes a critical future improvement to be done.

6.2 Traffic between Client and Master

To compare the traffic pattern difference between our design and Dropbox, we applied different sizes of changes to a file and measured the packets exchanged between our master and client devices via Wireshark (v2.2.3-0-g57531cd). In Fig. 6, we can observe the quite distinct patterns from two systems. As the size of Δ data goes up, Dropbox induced increasing packets change between client and master (server). However, constant number of packets change is observed in our system, which is mainly metadata of the change. It seems that 2MB Δ data generates the most packets. The possible explanation could be the change of the encoded data chunks at 2MB hits a high point given the specific data encoding schemed adopted by Dropbox.

Table 1: Functionality Tests

File Type	Editor	Result
Text	Atom	✓
	Sublime	✓
	Nano	✓
	Touch	✓
	Vim	✓
	Gedit	X
Image		✓
Audio		✓
Video		✓

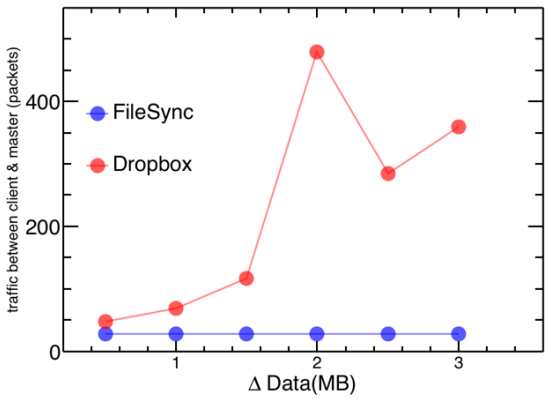

Figure 6: Traffic between Client and Master v.s. Δdata

7 RELATED WORK

There are many studies and research done on the existing cloud storage services [3]. Recently a large number of measurement research efforts have been conducted on cloud storage services [2, 4, 5]. Focusing on personal cloud, Drago et al. give a large scale measurement for Dropbox [3], and then compare the system capabilities for five popular cloud storage services in [4]. There are also many studies about the system design for cloud storage services [1, 7, 8]. Delta encoding [6] is also not a new idea but it poses big challenge when implemented with the cloud storage system where files are split into chunks and stored in a distributed way.

Table 2: Feedback

Sr. No.	Feedback	Status
1.	Stronger Security	Implemented
2.	Handle Network Partition	Implemented
3.	Backup Server	Implemented
4.	Multiple Clients	Implemented
5.	Solve multiple client conflict	Implemented
6.	Support Gedit	Not Implemented
7.	More Evaluation	Not Implemented

8 FEEDBACK

(1) Stronger Security: Communications between Control Server and Client are secured by SSLv23 and communication between clients is secured by SSH.
(2) Handle Network Partition: We implemented Lamport's logical clock for recovering a client after network partition.
(3) Backup Server: Can be done by running the Control Server code on any server with a few configuration changes of IP address made in the code.
(4) Multiple Clients: Just run the client code on any new client and change the configuration of IP address in the code.
(5) Solve multiple client conflict: Already has a lock mechanism implemented for solving conflicts.
(6) Support Gedit: Cannot be implemented with delta mechanism.
(7) More Evaluation: Could not be done due to time constraints.

9 CONCLUSION

A peer to peer storage cloud synchronization system that offers fast and instant file sync service is designed and implemented. In the presented system, server only stores and broadcasts meta-data pertinent to the changes made on every client, while the client-client communication takes care of the heavy data exchange. The prototype is tested on a variety of file types and editors and shows robust performance. The evaluation reveals that the peer-to-peer data exchange pattern is promising to make this system scale up to serve a large number of clients, given the limited server resource.

REFERENCES

[1] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, and Leonidas Rigas. 2011. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11)*. ACM, New York, NY, USA, 143–157. https://doi.org/10.1145/2043556.2043571

[2] Idilio Drago, Enrico Bocchi, Marco Mellia, Herman Slatman, and Aiko Pras. 2013. Benchmarking Personal Cloud Storage. In *Proceedings of the 2013 Conference on Internet Measurement Conference (IMC '13)*. ACM, New York, NY, USA, 205–212. https://doi.org/10.1145/2504730.2504762

[3] Idilio Drago, Marco Mellia, Maurizio M. Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. 2012. Inside Dropbox: Understanding Personal Cloud Storage Services. In *Proceedings of the 2012 Internet Measurement Conference (IMC '12)*. ACM, New York, NY, USA, 481–494. https://doi.org/10.1145/2398776.2398827

[4] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. 2010. CloudCmp: Comparing Public Cloud Providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*. ACM, New York, NY, USA, 1–14. https://doi.org/10.1145/1879141.1879143

[5] T. Mager, E. Biersack, and P. Michiardi. 2012. A measurement study of the Wuala on-line storage service. In *2012 IEEE 12th International Conference on Peer-to-Peer Computing (P2P)*. 237–248. https://doi.org/10.1109/P2P.2012.6335804

[6] Andrew Tridgell. 1999. *Efficient algorithms for sorting and synchronization.* Australian National University Canberra.

[7] Michael Vrable, Stefan Savage, and Geoffrey M. Voelker. 2009. Cumulus: Filesystem Backup to the Cloud. *Trans. Storage* 5, 4, Article 14 (Dec. 2009), 28 pages. https://doi.org/10.1145/1629080.1629084

[8] Michael Vrable, Stefan Savage, and Geoffrey M. Voelker. 2012. BlueSky: A Cloud-backed File System for the Enterprise. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*. USENIX Association, Berkeley, CA, USA, 19–19. http://dl.acm.org/citation.cfm?id=2208461.2208480