CloSSer

Cloud Storage Services

Icaro Alzuru
CISE Dept., University of Florida
Gainesville, USA
ialzuru@ufl.edu

Qi Cai/Huixiang Chen
ECE Dept., University of Florida
Gainesville, USA
{qcai, Stanley.chen}@ufl.edu

***Abstract*—In a time of big data, enterprises and individual users are more willing to store or backup their data into cloud storage. In most cases, only a single cloud storage vendor is chosen as the service provider. As a result, users often suffer from single point of failure and vendor lock-ins. To better handle this problem, we created a product to replicate data across multiple cloud platforms with fault tolerance function in case of data corruption or server crash. In addition, to decrease the cost of storage and bandwidth, we implement file deduplication function. To make our system friendly yet powerful to different levels of users, we design two kinds of interfaces. For clients, we abstract our system as a mountable device with the help of FUSE. Interactions with the device are through Linux file system commands so that all the network communications are hidden from the users. For network administrators, we provide a graphical interface to configure and watch the system. It's worth mentioning that, our system is designed in layers so that it is very easy to be extended and support more cloud platforms. In addition, all the computation is done at the client side so that the storage nodes are not required to have computational and coordination ability. Experimental results show that, our system can achieve satisfactory storage, bandwidth and delay performances.**

Keywords— distributed file system, fault tolerance, recovery, deduplication, FUSE

I. INTRODUCTION

With the explosion of Big data, distributed storage systems which save data over a set of storage nodes across a network are applied more and more widely. The most popular distributed file system is cloud storage, for example, Amazon S3 [24], Dropbox [4], Google drive [8, 9, 10] and Onedrive [1, 2] to which enterprises and individual users can outsource their data backups.

In most cases, only one cloud storage vendor is chosen as the service provider, which can be problematic. Although it seems to simplify the system design and decrease the cost. As your system only has to support one kind of cloud platform, both the development cost and expenses paid to cloud vendor are less than the situation when multiple cloud platforms are used. However, using a single provider will lead to single point of failure and vendor lock-ins problems [17]. The former problem happens when the service of one cloud provider is temporarily unavailable or fails forever because of catastrophe. While the latter problem is caused by the high cost of changing to a new server provider. Typically, the storage service provider charges the users for storage space as well as upload and download bandwidth. After a significant amount of data have been accumulated on a cloud platform, users have to use twice as much the bandwidth as that of the storage space in order to first download the data from old cloud and then upload it to a new cloud. And the cost of this migration is usually prohibitive. That is called the vendor lock-ins effect.

To handle the single server failure problem, we use replication and fault tolerance techniques in our system. In contrast with using a single storage provider, we replicate the data across multiple clouds. As a result, even if the service of one provider is unavailable, we can use the redundant information on other working clouds. To remain a certain level of redundancy, we need to recover the data after corruption or server crash. That's why we introduce the fault tolerance function [28].

To replicate data across servers, we will be charged more for the storage space, upload and download bandwidth cost than using one server. So we implement the file deduplication function to reduce costs and relief the vendor lock-ins effect. The idea of file deduplication is that, when the file we are going to upload has the same content as a file already on the cloud, we don't need to upload it again. Instead, we just record this relationship. As the storage cost to keep this relationship is much less than that of storing a file, the savings of storage space and upload bandwidth is significantly. This advantage becomes especially obvious when the file size goes larger. In practice, large files like movies, music or photos are very likely to have the same content between people with same interests. In this situation, the file deduplication function can effectively reduce the cost and helps relieve the vendor lock-ins effect.

As both, individual users and enterprises can benefit from our system, we need to design friendly yet powerful interfaces to meet their different requirements. For individual users, they just want to backup their file into multiple clouds with just one effort. So we abstract our system as a mountable storage device and expose APIs so that users can interact with the device by standard Linux commands. As all the network communication is hidden from the users, it is pretty easy to use. For the network administrators in large companies, in addition to backup files, they want to frequently watch the

configuration, availability and content of the distributed file system. So we develop a graphical interface for this requirement.

The contributions of our work are summarized as follows:

A. We extend the NCCloud framework to support more cloud platforms including Dropbox, Google Drive and Microsoft OneDrive.
B. We implemented replication and fault tolerance functions for the cloud platforms mentioned above so that single node failure problem can be handled.
C. We designed file deduplication function to reduce the cost of storage and upload bandwidth, and to help relieve the vendor lock-ins effect.
D. We implement command line and graphic interfaces to cater the requirements of individual users and network administrators in large companies

The rest of the paper is organized as follows. In Section II, we introduce the technical background of our system, including FUSE, Microsoft OneDrive and Google Drive. In Section III, we talk about the architecture of our system. In section IV, we describe in details the layer design of our cloud storage system and the key APIs. In section V, we evaluate the performances of our cloud storage systems.

II. RELATED WORK

In this section, we will first introduce the FUSE file system, and then introduce the python APIs for various cloud storages.

FUSE file System: Filesystem in Userspace (FUSE) [11-12] is an OS mechanism that let non-privileged users create a fully functional filesystem in userspace. The FUSE project was started in October 2004 as a fork of A Virtual Filesystem (AVFS), which was first released in 1998. FUSE provides python interface for user to write their own user-space file systems.

Python APIs for manipulating cloud storages: Many cloud storage companies provides python APIs to access and manipulates their storages, including Microsoft, Dropbox, Google, etc. This provides convenience for developers. We will first introduce these APIs and their authentication processes in this section, which will be used in our implementation.

A. OneDrive Python API

Microsoft OneDrive provides REST API for accessing, and interacting with the data stored within a user's OneDrive [1]. On top of this, Microsoft also provides Python API by wrapping the RESTful versions [2], which is what we will use in our implementation for CloSSer [3, 18, 19]. To use the OneDrive API, an access token that authenticates the app to a particular set of permission for a user is needed. One example authentication flow is shown in Figure 1 [1]. The required parameters are: (1) client_id, which is the client ID value created for you application; (2) scope, a separated list of scopes the application requires; (3) redirect_uri, which is the redirect URL that the browser is sent to when authentication is complete.

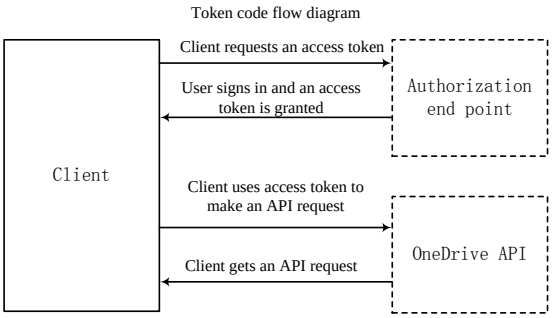

Figure 1. OneDrive authentication flow

B. Dropbox Python API

Dropbox also provides python API for accessing and manipulates a User's Dropbox account [4] [7]. The Core API is based on HTTP and OAuth. The Core API uses OAuth V2, which is the next evolution of the OAuth protocol, provides specific authorization flows for web apps, desktop apps, and mobile devices [5-6]. Compared to the complex authorization process of OneDrive, Dropbox takes care of the authorization in the backup. To get a Dropbox python client, all you need to do is to provide an *access_token*: *client = dropbox.client.DropboxClient(access_token)*. As the leader in cloud storage industry, Dropbox provides fast, efficient and user-friendly APIs for users.

C. Google Drive Python API

Besides, Google also provides REST APIs that help developers develop apps that integrate with Google Drive [8]. Similar to Dropbox, Google Python API also use OAuth V2 as the authentication tool. Similar to Dropbox and OneDrive, the core functionality of Google Python API is to provide download and upload files in cloud storage. However, different from Dropbox and OneDrive which creates folders directly; in Google Drive, to create a folder, first you need to create a file, and then change the metadata type of "file" to "folder".

D. Box Python API

Box (Box.net) [14] is an online file sharing and content management service for business. This company provides 10GB of free storage for personal accounts. It also provide python API [15] for file management. Very similar to OneDrive python API, Box API also use OAuth as the authentication tool and need client_id/client_secret/access_token as the input parameters. But it also has some difference: for example, Box API does not provide file download support, during which we have to implement the download function by ourselves. In our platform, we will also implement the CloSSer support for Box.

III. SYSTEM ARCHITECTURE

A. Architecture Overview

Our CloSSer system contains three layers, see Figure 2. Although it is inherited from NCCloud [3, 18, 19], we have introduced many improvements in each layer.

The top one is interface layer through which users interact with the local file system layer. As we have mentioned above, individual users and network administrators in large companies have different requirements on this layer. To cater their needs, we design graphical interfaces for network administrators, while expose simple yet powerful Linux file system command lines to individual users. The implementation of graphical interface is an advantage over NCCloud.

The middle one is the local file system layer. With the help of FUSE, the command issued by users in the form of Linux file system calls can be translated to operations on cloud storage nodes. While at the same time necessary information can be computed and stored locally. In this way we can customize the function of our local file system to enable computation, communication and coordination with cloud storage nodes. As we what we will introduce in the next section, we add deduplication function in this layer to reduce storage and bandwidth cost. This is an advantage over NCCloud.

The bottom one is the cloud storage platform layer. We need to communication with those storage nodes through protocols specified by the service providers to realize basic file operations, like upload, download and delete. In our CloSSer system, we developed operation libraries for the most popular storage providers including Dropbox, Google Drive and Microsoft OneDrive. In contrast, NCCloud doesn't support those platforms.

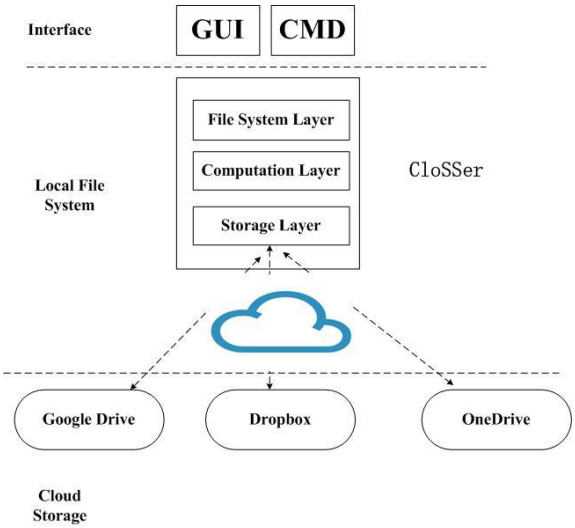

Figure 2. System Architecture of CloSSer

B. Layered design of CloSSer

As we have mentioned above, the most important part of the CloSSer is the local file system as it undertakes command interpretation, computation, communication, and coordination duties. We further divide this local file system into three sub layers. In the rest part of this section, we will introduce the functions of each layer.

File System Layer. The duty of file system layer is to translate the file related system calls, like read, write or delete into the functions we defined in our local file system. For example, when we finish writing a file and close it, it is the time to upload the modified file into clouds for backup. When we initialize a file, we must test if it exists on the cloud. If so, we have to download it and then read it or write it based on the original data.

Computation Layer. The computation layer implements the workflow to some basic cloud operations and provides services to the file system layer above. Currently, the supported functions are to download a file from clouds, to upload a file to clouds, to replace a failure node with a spare node in the cluster and rebuild the data of the failure node on this spare node, to perform file deduplication. We will describe the algorithms of those functions with more details in the following system design and implementation section.

Storage Layer. The storage layer provides services to the computation layer above by specifying a unified interface for all the cloud platforms. In this way, the implementation variations among different clouds are hidden from the computation layer. A command can achieve the same result logically. For example, on Dropbox it's very easy to create a directory and put data into it. On the contrary, on Google Drive both directories and files are treated as files. It uses different metadata to differentiate them. In spite of the low level differences, with the help of storage layer, we can specify a unified function interface named upload file and then implement various libraries related to different cloud platforms. In this way, the storage layer hides the diversity of clouds from computation layer. The computation layers can treat all the cloud nodes equally.

IV. SYSTEM DESIGN AND IMPLEMENTATION

In this section, we introduce the design and implementation details of our file system. In the first section, we show the definition and signatures. In the second section, we discuss the features of the cloud platforms we developed. In the third section, we introduce the algorithm of the workloads in computation layer. And how they are implemented based on the storage layer. In the last section, we introduce the graphical interface we designed.

1. Specifications of storage layer APIs

CloSSer provides cloud backup for several cloud storages including Google Drive, Dropbox, and OneDrive. In our implementation, different cloud storage in the storage layer provides the same interfaces for the workflow in the upper layer. Inside each cloud storage, since the python API is different for each cloud, the implementation for these interfaces are different.

The uniform interface is:

1) *syncMirror*

Input: *setting, nodeid, path*

Output: *True*/*False*

This function synchronizes all file entries on the cloud to local directories: to synchronize all files ending with *.node0* or *.pt* (pointer files for deduplication).

2) *checkHealth*

Input: *setting, nodeid*

Output: *True*/*False*

This function checks whether a *bucketname* exists in the cloud storages. For example, the *bucketname* for Onedrive is *huixiangOneDrive*/, if there is a folder in Onedrive storage named *huixiangOneDrive*/, this function will return true.

3) *uploadFile*

Input: *setting*, *nodeid*, *localpath*, *dest*

Output: *True*/*False*

This function uploads a file to the bucket in the cloud storage from *localpath*. If it is successfully uploaded, returns *True*; else returns *False*.

4) *uploadMetadata*

Input: *setting*, *nodeid*, *metadatapath*, *dest*

Output: *True*/*False*

This function uploads a metadata file to the bucket in the cloud storage from *metadatapath*. The metadata file stored in cloud storage is named *dest*+"*.metadata*". If successfully uploaded, returns *True*; else returns *False*.

5) *uploadFileAndMetadata*

Input: *setting*, *nodeid*, *localpath*, *metadatapath*, *dest*

This function uploads a file and metadata file to the bucket in the cloud storage, using a file from *localpath* and *metadatapath* respectively. The file is named *dest*, the metadata file is named *dest*+".metadata". If successfully uploaded, returns *True*; else returns *False*.

6) *downloadFile*

Input: *setting*, *nodeid*, *src*, *path*

Output: *True*/*False*

This function downloads a file in the cloud storage to local file system. *src* is the filename stored in the bucket. The file downloaded is stored in *path* in local file system.

7) *downloadMetadata*

Input: *setting*, *nodeid*, *src*, *metadatapath*

Output: True/False

This function downloads a metadata file in the cloud storage to local file system. *src*+".metadata" is the filename stored in the bucket. The file downloaded is stored in *metadatapath* in local file system.

8) *existsFile*

Input: *setting*, *nodeid*, *name*

Output: *True*/*False*

This function checks whether a file exists in cloud storage bucket. If it exists, returns *True*; else returns *False*.

9) *deleteFile*

Input: *setting*, *nodeid*, *name*

Output: *True*/*False*

This function deletes a file from cloud storage bucket. If successfully deleted, returns *True*; else returns *False*.

10) *downloadPointers*

Input: *setting*, *nodeid*

Output: *True*/*False*

This function first connects to a cloud specified by *nodeid*. This *nodeid* is used as an index in *setting.nodeInfo[nodeid]*. After the connection, it downloads all files end with extension *.pt* under the bucket specified for the node. If the download succeed returns *True*, else returns *False*.

11) *detectFile*

Input: *setting*, *filename*, *nodeid*.

Output: *retState.*

This function detects if there is a file specified by *filename* existing on a cloud specified by *nodeid*. The *nodeid* is used as an index for *setting.nodeInfo[nodeid]*. If the file exists returns *True*, else returns *False*

12) *deletePointer*

Input: *setting*, *nodeid*, *name*.

Output: *True*/*False.*

This function delete the pointer file related to the file specified by name on a given cloud specified by *nodeid*. The *nodeid* is used as an index for *setting.nodeInfo[nodeid]*.

The naming rule for a pointer file is *filename + '.pt'*. For example, if the input arguments are *setting*, *0*, *'myfile.txt'*, then this function will delete the pointer file with name *myfile.txt.pt* on the cloud *0*. If the deletion succeeds, returns *True*, else returns *False*.

13) *downloadPointer*

Input: *setting*, *nodeid*, *filename*

Output: *True*/*False*

This function downloads the pointer file of the file specified by filename from the cloud specified by *nodeid*. The *nodeid* is used as an index for *setting.nodeInfo[nodeid]*. The naming rule for a pointer file is *filename + '.pt'*. For example, if the input arguments are *setting*, *0*, *'myfile.txt'*, then this function will download the pointer file *'myfile.txt.pt'* from cloud 0. If download succeeds, returns *True* else returns *False*.

2. Features of cloud storage platforms

A. Dropbox module

Dropbox provides very simple yet powerful APIs for the developers. You can use both, web authentication or secret access token to connect to the application server. After that, you can perform any operations on the client object returned. To upload a file into Dropbox, you should first open a file and get its file handle. After that, you pass the destination file path on Dropbox, and the file handle as arguments for file uploading. To download a file you should first open both the destination file on local file system and the source file on remote cloud. After that you can read the content from the remote file and write that to the local file. The Dropbox API supports list directory contents by returning the whole metadata contents of a directory. The Dropbox API also supports file search by providing a specific search function.

B. Google Drive module

Google Drive [8, 9, 26] is a "file storage and synchronization service created by Google" [10]. In CloSSer, we use it as a storage resource but we do not exploit its capacities for collaborative editing of documents, spreadsheets, and presentations.

Google provides two API flavors to interact with Google Drive, the first is a full featured Rest API [8], and the second one is PyDrive [13]: a wrapper library of google-api-python-client that simplifies many common Google Drive API tasks. The Drive Rest API can be integrated in Java, .Net, PHP, and Python, but the method we preferred was PyDrive due to its simpler syntax.

We implemented for Google Drive the 13 functions mentioned in section IV, using PyDrive. Some challenges faced when implementing these functions were:

- In Google Drive there are no real buckets or folders, these are special files which are configured through their metadata as "parents" of the files they have inside "children". This implies that we need to get the Id of the parent folder or bucket in order to list or search its files, and when uploading a file, its metadata must be configured to indicate its parent folder.
- PyDrive does not include a delete or remove file function. In order to implement this functionality we had to use a Rest API call.
- PyDrive requires the user downloads the file "client_secrets.json" from the Google website in order to verify the credentials. This file must be copied in the GoogleDrive directory of the root folder of CloSSer.

C. OneDrive Module

The main problem in implementing the function is the connection to the cloud. If connected multiple times, there will be an error "address in use". Thus the cloud must be connected only once. That is why we hard-coded the connection code (from line 20-line 37) in mydrive.py.

The bucket is implemented in OneDrive as a folder in the root directory. checkHealth is used to check whether this folder exists, which will be used in workflow.py. Upload and download is supported in OneDrive python API, so we can just directly call them.

In the cloud platform, normal file ends with .node0, metadata file ends with .node0.metadata.

D. CloSSer's Administrator Interface

Although it is still a work in progress, we developed a web interface to manage the main functionalities of CloSSer. System administrators usually deal with many data sets or applications which need to be backed up or deployed for synchronization, and the console work becomes tedious for some administrators.

The Administration console was implemented using the Django framework [16], which enables the development of dynamic web sites using Python. This framework was chosen because CloSSer was developed in Python and Django is the most used framework for Python web programming.

When the user access the administration console, the administrator login and password must be inserted, see Figure 3 below.

Figure 3. Login screen for the Administration Console

After the credentials of the administrator have been verified, the management console is displayed, see Figure 4.

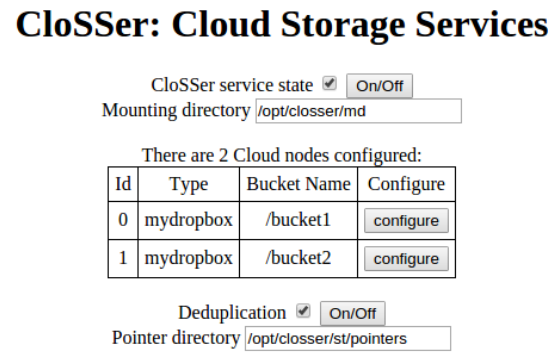

Figure 4. Administration Console

The console reads some parameters of the configuration file of CloSSer and allows the administrator to change them. A more advanced version of this console should allow the execution of simple operations like upload and delete of the files in the mounting directory and working in parallel with several configuration files, that is to say several mounting directories. We believe this kind of interface can increase the productivity of the CloSSer administrators.

3. Algorithms of the Computational Layer Workflow

A. Deduplication algorithm

The first workflow we introduce is for deduplication. This is the basis to understand our file system. In the view of whether a file is duplicated, we categorize the status of a file into 3 types. As is shown in Figure 5, there are three files named F1, F2 and F3. Among them, F1 and F2 share the same content, while the content of F3 is different from the rest two files. We use a directed graph to describe this relationship. We define F1 as an original file as it only has input degree. We define F2 as a pointer file as it only has output degree. We define F3 as a non duplicated file as there is no other file sharing the same content with it.

The idea of file deduplication is very intuitive: we don't upload a file with the same content than a file that has already be uploaded to the cloud. Instead, we upload the information that record which nodes share the same content. We still use the situation in the Figure 5 as an example. Suppose we have already uploaded F1 on the cloud. When we try to upload F2, the system we detect that F1 and F2 share the same content, so that instead of uploading F2, a pointer File named F2.pt will be uploaded. The content of F2.pt is F1, which specifies which file shares the same content as F2. As it is costs much less space to store file name than to store file content, we can save a significantly amount of storage with the help of deduplication.

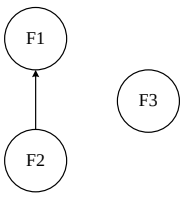

Figure 5. File categorizations

Although the idea of deduplication is intuitive, when it comes to modify the content of files the situation becomes complicated. This is because, when files point each other, for the purpose of deduplication, they have dependencies which can be modeled by a graph. When one node in the graph is modified, the structure of the graph may be changed dramatically. For example, in Figure 6, files named F1, F2, F3, F4, F5, and F6 share the same content, and they form a dependent relationship described by this graph. After we change the content of F1, the previous relationship doesn't hold any more. However, as the contents of files F2 to F6 are same, a new dependent relationship should be built as shown by Figure 7. In this situation, F2 is elected as an original file with its content uploaded to the cloud. The files F3 to F6 are still pointer files with updated pointer information. In this situation, their pointer information will change from F1 to F2.

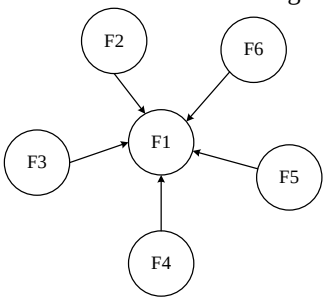

Figure 6. Relationship of files before changing F1

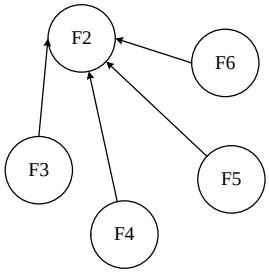

Figure 7. Relationship of files after changing F2

B. The algorithm of *uploading* a file

Based on the deduplication algorithm we introduced above, there are 6 following cases when we try to upload a file. Which is described in Algorithm 2.

1. Uploading a non duplicated file or a new file into the cloud. In this case, we upload the whole file content with corresponding metadata.
2. Modifying the content of a non duplicated file so that it has the same content with an existing file on the cloud. Or uploading a new file with the same content as with an existing file on the cloud. In both cases, we should upload pointer files of corresponding files. In former case, we should also delete the content and metadata before modification.
3. Modifying the content of a pointer file so that after the modification, it has the same content as another file on the cloud. In this case, we just have to update the content of pointer file from old original file to a new original file.
4. Modifying the content of a leaf file so that after the modification it won't share the content with any other file. In this case, we first delete its pointer file and upload the whole content and related file metadata.
5. Modifying the content of an original file so that it won't share its content with any other file on the cloud. In this case, we first update the content and metadata of the file to its latest status. And then we pick one of its previous pointer files and make it as an original file. After that, we update the rest pointer files so that the file contents of them are the name of the new original file.

Algorithm 2. Uploading a file

Input: file A
The original file A (if exists) on the cloud is A_{old}
The new file A on the cloud is A_{new}

if ((A_{old} does not exist || A_{old} has no duplication) && A_{new} has no duplication) **then**
    upload A_{new} on the cloud
    upload A_{new}.*metadata* on the cloud
else if (A_{old} has no duplication && A_{new}==B) **then**
    delete A_{old} on the cloud
    delete A_{old}.*metadata* on the cloud
    A.pt=B.name //let A points to B
    upload A.pt on the cloud
else if(A_{old} is a pointer file && A_{new}==B) **then**
    A.pt=B.name
else if(A_{old} is a pointer file && A_{old}.pt==B.name && A_{new}!=B) **then**
    delete A_{old}.pt
    upload A_{new}
    upload A_{new}.*metadata*
else if(A_{old} is an original file && several files points to A && A_{new} has no duplication) **then**:
    upload A_{new}
    upload A_{new}.*metadata*
    B=choose(A.*FilePointerList*)
    upload B
    upload B.metadata
    for F **in** A.*FilePointerList*:
        F.pt=B.name //let F points to B
else if(A_{old} is an original file && several files points to A && A_{new}=C) **then**:
    delete A_{old}
    delete A_{old}.*metadata*
    B=choose(A.*FilePointerList*)
    upload B
    upload B.*metadata*
    for F **in** A.*FilePointerList*:
        F.pt=B.name
    A.pt=C.name
    upload A.pt
endif

6. Modifying the content of an original file so that after the modification it has the same content as another file. In this case, we first delete the content and metadata of the out of date file and upload the pointer file. After that, we pick up one pointer file as the new original file and update the content of the rest pointer files so that they point to the new original file.

The seven cases above cover all the situations we can meet when uploading a file. We have also tested all the cases in part E of section V. Production Evaluation.

C. The *algorithm* of downloading a file

Algorithm 3. Downloading a file

Input: file A

```
if (type(A)==Leaf File) then
    download Orig(A)
    download Orig(A).metadata
else
    download A
    download A.metadata
endif
```

The algorithm first checks if file A is a pointer file on the cloud, if true, then download the content and metadata of original file A points to; if not, just download its own content and metadata.

D. The *algorithm* of deleting a file

Algorithm 4. Deleting a file

```
Input: file A

if (type(A)==Leaf File) then
    delete A.pt on the cloud
else if(type(A)==Normal File & A has no duplication) then
    delete A on the cloud
    delete A.metadata on the cloud
else if(type(A)==Normal File && A has duplication) then
    B=choose(A.PointerFileList)
    change B as the original file
    for F in A.PointerFileList:
        let F points to B
endif
```

The algorithm of deleting is executed based on the type of the input file A. If A is a pointer file, just delete the corresponding pointer on the cloud; if A is a normal file and has no duplication file, delete the content and the metadata of A; if A is a normal file and has duplication file, then we first pick up one of the pointer files B as the new original file and update the contents of the rest pointer files so that they point to the new original file.

E. The algorithm of rebuilding a file

Algorithm 5. Rebuilding a node

```
S=choose(SpareNodeList)
A=choose(SurvialNodeList)
Files=download(A)
S.upload(Files)
```

The workflow of rebuilding a failure node works as follows: first find a node S in the list of spare nodes to replace the failure node; second find an available node A in the list of survival nodes and download all the files in to local directory; then upload the files to the S to recover the lost data.

V. PRODUCT EVALUATION

A. Black Box Tests

The functional correctness of the developed software was verified through Black Box Tests. For each of the methods described in the interface we run tests to verify the correct execution of the methods.

B. Upload and Delete Performance Comparison

Because we developed Google Drive, Dropbox, and OneDrive components for CloSSer, the performance of these 3 cloud storage services was compared in the following way: we selected 75 songs, which occupied 500 MB, and registered the execution time for the upload and deletion processes of this dataset, in each cloud.

It is important to highlight that the performance of these tests can be affected by external conditions like network traffic or servers utilization. Because of that, every test was executed 3 times for each cloud and what we present in each case is the average execution time.

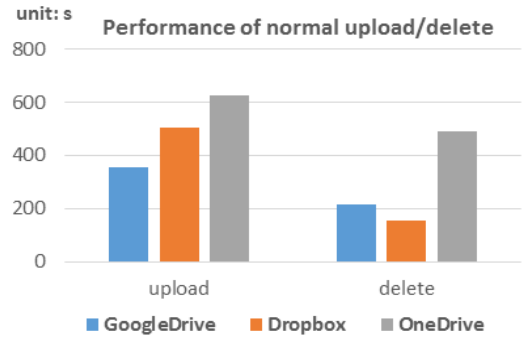

Figure 8. Upload and delete performance

It is important to know the performance of the available cloud services and their APIs. As it can be observed in the graph, using CloSSer with Google Drive as storage provider was the faster cloud service for uploading new content, but CloSSer with Dropbox was faster for deletion, removing files. This results suggests that Dropbox could be more efficient for short life content while Google Drive would more suitable for online services where the user experience can be affected by the file availability.

C. Rebuild Performance

Rebuild is the process of detecting a failing cloud and replacing it by a spare node. In this process, all the files of the CloSSer drive are copied to the new node. The configuration file is updated to indicate the new role of the spare node.

In this experiment, we configured 2 data nodes and 1 spare node, and simulated the failure of one of the nodes, renaming its bucket. The rebuild process was executed after this. It was measured the time taken by the 3 under study clouds: Google Drive, Dropbox, and OneDrive to replicate the data in the spare node.

The rebuild process is slow because first verifies the consistency of the mirror directory, copying all the files from a working cloud to this local folder and then uploads all the files to the spare node.

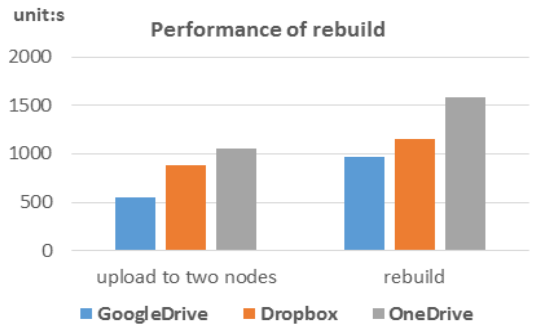

Figure 9. Rebuild performance

As expected from the results shown in the previous study, Google Drive was faster when uploading the files to the 2 data

nodes (in order to setup the experiment) and also executing the rebuild, which is basically copying the files locally and executing the upload in the spare node.

D. Scalability *Test*

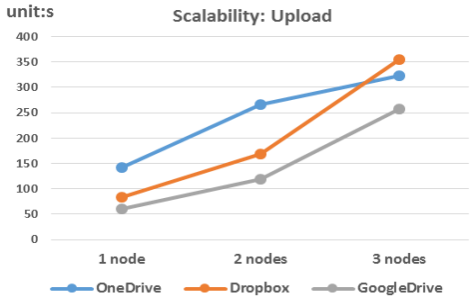

Figure 10. Upload scalability test

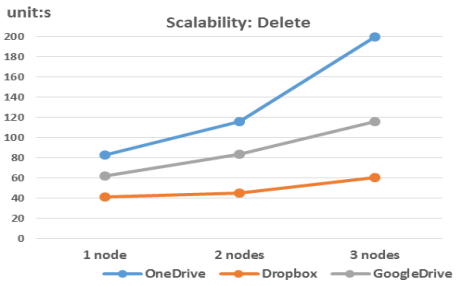

Figure 11. Delete scalability test

We also tested the scalability of our system by varying the number of nodes from 1-3, and record the execution time of uploading 100 mp3 files and deleting. From previous experiment, the conclusion that GoogleDrive is the fastest in uploading and Dropbox is the slowest in deleting still applies to the scenario of multiple nodes. What is more, of all three cloud platforms, Dropbox has the best scalability in deleting: the time used for deleting almost does not change when the number of nodes increases.

E. Test cases for *deduplication* module

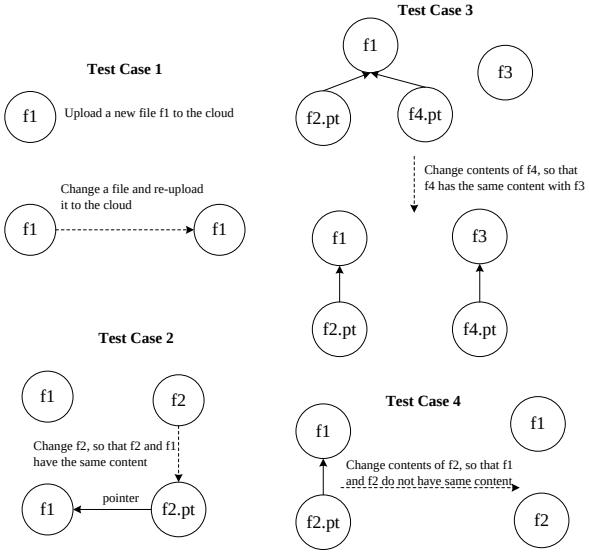

Figure 10. Test cases for the deduplication module - Part A

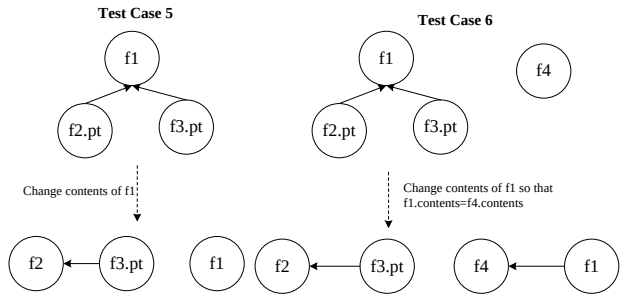

Figure 11. Test cases for the deduplication module - Part B

Table 1. All test cases for deduplication

Test1	First upload a file f1 to the cloud, change the contents of f1 and re-upload. Expected results: file uploaded, changed and updated successfully in the cloud.
Test2	Currently there are two different files f1 and f2 in the cloud, change contents of f2 so that f2 and f1 have the same contents. Expected results: f2 becomes a pointer, pointing to f1.
Test3	Currently there are four files, f1, f2, f4 have the same contents (i.e. f2 and f4 points to f1), f3 has different contents with the other three. Change the contents of f4 so that f3 and f4 have the same contents. Expected: f2 points to f1, f4 points to f3.
Test4	Currently there are two files, f2 points to f1, changes contents of f1. Expected results: f2 becomes a separate file (no longer pointer file).
Test5	Currently there are three files, f2 and f3 points to f1, change the contents of f1. Expected results: f3 points to f2, f1 becomes a separate file.
Test6	Currently there are four files, f2 and f3 points to f1, f4 is different. Change the contents of f1 so that f4 and f1 have the same contents. Expected results: f3 points to f2, f1 points to f4.

For all kinds of cloud storages, we tested the function of deduplication using the test cases shown in Figures 10 and 11. Each test case is described in Table 1.

We have finished and passed all the test cases on all cloud platforms, results shown that the function of our implementation module works successfully.

VI. CONCLUSION

We developed CloSSer, a distributed file system to backup data across multiple storage nodes in a replication method. We also implemented the fault tolerance function to fight against data corruption and server crash. To relieve the vendor lock-ins effect, we designed the file deduplication function to decrease the cost of storage and communication bandwidth. To meet different requirements of individual users and network administrators of large companies, we designed a simple yet powerful command line interface and a detailed and comprehensive graphical interface. To take advantage of state-of-the-art cloud storage techniques, CloSSer supports Dropbox, Google Drive and Microsoft OneDrive cloud platforms. The rich function set of CloSSer makes it an ideal platform for both research and commercialization.

REFERENCES

[1] OneDrive REST API, https://dev.onedrive.com/README.htm
[2] OneDrive Python API, https://github.com/OneDrive/onedrive-sdk-python
[3] Chen H C H, Hu Y, Lee P P C, et al. NCCloud: a network-coding-based storage system in a cloud-of-clouds[J]. Computers, IEEE Transactions on, 2014, 63(1): 31-44.
[4] Dropbox Web API tutorial, https://www.dropbox.com/developers-v1/core/start/python
[5] OAuth V2, http://oauth.net/2/
[6] OAuth, http://oauth.net/
[7] Dropbox Python API, https://github.com/dropbox/dropbox-sdk-python
[8] Google REST API, https://developers.google.com/drive/web/about-sdk
[9] Google Drive home page, https://www.google.com/drive/
[10] Google Drive, https://en.wikipedia.org/wiki/Google_Drive
[11] FUSE file system, http://fuse.sourceforge.net/
[12] FUSE file system, https://en.wikipedia.org/wiki/Filesystem_in_Userspace
[13] PyDrive, https://pypi.python.org/pypi/PyDrive
[14] Box Company, https://en.wikipedia.org/wiki/Box_(company)
[15] Box python API, https://github.com/box/box-python-sdk
[16] Django, https://www.djangoproject.com/
[17]Abu-Libdeh H, Princehouse L, Weatherspoon H. RACS: a case for cloud storage diversity[C]//Proceedings of the 1st ACM symposium on Cloud computing. ACM, 2010: 229-240.
[18]Yuchong Hu, Patrick P. C. Lee, Kenneth W. Shum "Analysis and Construction of Functional Regenerating Codes with Uncoded Repair for Distributed Storage Systems." Proceedings of IEEE INFOCOM, Turin, Italy, April 2013.
[19] Yuchong Hu, Henry C. H. Chen, Patrick P. C. Lee, and Yang Tang "NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-Clouds" Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST '12), San Jose, CA, February 2012.
[20] Chan-I Ku; Guo-Heng Luo; Che-Pin Chang; Shyan-Ming Yuan, "File Deduplication with Cloud Storage File System," in Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on , vol., no., pp.280-287, 3-5 Dec. 2013
[21]Manogar, E.; Abirami, S., "A study on data deduplication techniques for optimized storage," in Advanced Computing (ICoAC), 2014 Sixth International Conference on , vol., no., pp.161-166, 17-19 Dec. 2014
[22]Chappell D. The windows azure programming model[J]. David-Chappel & Associates, Tech. Rep, 2010.
[23]Chappell D. Introducing the windows azure platform[J]. David Chappell & Associates White Paper, 2010.
[24]Varia J. Architecting for the cloud: Best practices[J]. Amazon Web Services, 2010.
[25] Drago I, Mellia M, M Munafo M, et al. Inside dropbox: understanding personal cloud storage services[C]//Proceedings of the 2012 ACM conference on Internet measurement conference. ACM, 2012: 481-494.
[26]Hamburger E. Google Drive vs. Dropbox, SkyDrive, SugarSync, and others: a cloud sync storage face-off[J]. The Verge, 2012.
[27] Meyer D T, Bolosky W J. A study of practical deduplication[J]. ACM Transactions on Storage (TOS), 2012, 7(4): 14.
[28] Saltzer J H, Kaashoek M F. Principles of computer system design: an introduction[M]. Morgan Kaufmann, 2009.

II. APPENDIX PROGRESS AND PROJECT MANAGEMENT

A.1 Current Status

Development of the code: 100 percent complete.

Evaluation of our system: 100 percent complete.

Milestones	Weekly plan	State	Deliverables
Implementation of Spark GPU translator	Sep. 11 - Oct.10	It is not feasible after detailed research, so we gave up developing the translator. However, we have developed a suite of spark GPU benchmarks, which is introduced in our mid-term report. The benchmark will be useful for future developers who are interested in performance of integrating GPU with Spark.	1)Spark- GPU benchmark program libraries. 2)Characterization performances and figures.
Debug of NCCloud file system and customize it for CloSSer	Oct. 10 - Oct. 16	Hit.	Bug free and customized file system for CloSSer
Implementation of Dropbox APIs for the storage layer of CloSSer	Oct. 17 - Oct. 23	Hit	Dropbox cloud interface library.mydropbox.py
Implementation of GoogleDrive APIs for the storage layer of CloSSer	Oct. 24- Oct. 30	Hit	Google Drive cloud interface library. googledrive.py
Implementation of Microsoft OneDrive APIs for the storage layer of CloSSer	Oct. 31 - Nov. 6	Hit	OneDrive cloud interface library. myonedrive.py
Implementation of file deduplication function	Nov. 7 - Nov. 13	Hit	Add new functions into workload.py
Extensions of local, mydropbox, googledrive and myonedrive libraries to support deduplication function	Nov. 14 - Nov. 20	Hit	Improved myonedrive.py, mydropbox.py and googledrive.py
Functional test and performance evaluation	Nov. 21 - Nov. 27	Hit	Bug free CloSSer cloud file system
User graphical Interface extension	Nov. 28 - Dec. 6	100% done	

A.2 Team Coordination (each member's duty), (will be discussed with each team)

Member	Duty
Icaro Alzuru	1. Implementation of Google Drive library 2. Implementation of graphical interface 3. Evaluate the performances of the whole system 4. Test the functions of Google Drive library.
Huixiang Chen	1. Implementation of Microsoft OneDrive library 2. Implementation of box library 3. Test the functions of myonedrive library 4. Test the functions of box library
Qi Cai	1. Customize and debug the NCCloud file system for CloSSer cloud file system 2. Implementation of mydropbox library 3. Implementation of file deduplication function 4. Test file deduplication function and mydropbox library

A.3 Milestones, Weekly Plans, and Deliverables (will be used in the final report as in 6.1 above, miss/hit/more)
Already covered in A.1