

Project Report

Cloud Storage and Replicated File Distributed System

CS550 – Advanced Operating System

Nov 28th 2017

Jinyang Li

A20317851

Introduction

Replication in computing involves sharing information so as to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility. [1] Cloud computing provides a simple way to access servers, storage, databases and a broad set of application services over the Internet. A Cloud services platform such as Amazon Web Services owns and maintains the network-connected hardware required for these application services, while user provision and use what user needed via a web application.[2]

Background Information

Simply put, cloud computing is computing based on the internet. Where in the past, people would run applications or programs from software downloaded on a physical computer or server in their building, cloud computing allows people access to the same kinds of applications through the internet. Cloud, in general, is kind of Internet based services. So why a Replicated File Distributed System over Cloud would be good? It provides lots of advantages such as Flexibility, Disaster recovery, Security and Environmentally friendly.

Problem Statement

Although there's lot of cloud storage service providers in market, such as Amazon Web Services, Aliyun, Tencent Cloud (QCloud) however, there's still a nice practice to develop an IIT Cloud file storage. This system would provide transparent replication. Allow user to Write, Read, Open, Close file but didn't know file are stored in N servers. The servers interact with each other to establish a writer lock, and a reader lock.

Demo purpose: Store duplicated simple text files in several file servers for demo purpose. Write operation just simply appends more text to the file. Client gets connected to any file server in a graphical user interface.

Related Work

Amazon Web Services, Aliyun, Tencent Cloud (QCloud) are all Cloud based RFDS. AWS's broad range of storage solutions for backup and recovery are designed for many of workloads today. This flexibility allows users to focus on business's needs, and eliminates the challenges around figuring out what storage systems to purchase.

Proposed Solution

In this proposal, we are requested to develop a Replicated File Distributed System and proposing to develop such a RFDS system based on Internet.

1# This System is software based system. Using C# to build user interface client, and use C# make server communication backgrounds node. For lock manager, we would choose either SQL database because it naturally supports locks or developing a new one.

2# We would evaluation the efficiency of our system and simulate catastrophic to one of the server nodes to ensure security and data recovery ability.

Compile Guide

To Compile Source Code, you need to install Visual Studio 2017 Community and .NET Runtime package. Also you need add reference to NetworkCommsDotNet dll shipped together in source code.

Just heading to .sln, open it with Visual Studio and build as release. Everything should have no problem.

Usage

To start Client, just double click client application, it would open with a GUI.

To upload a file, click upload button in GUI and select a file.

To perform other operations such as read/write, click associated buttons but remember to open it first.

Click refresh list after ANY operations except open/write to view the newest changes!

To start server, either run it via cmd with port parameter or double click it, and enter port manually. The ip addresses are hardcoded so you need to change it to real ip when use it on outside network other than localhost.

A simple batch file that opens 3 servers and 1 client(start it in Build Folder):

```
rmdir ServerA /s /q
```

```
rmdir ServerB /s /q
```

```
rmdir ServerC /s /q
```

```
mkdir ServerA
```

```
mkdir ServerB
```

```
mkdir ServerC
```

```
Copy *.dll ServerA\
```

```
Copy *.dll ServerB\
```

```
Copy *.dll ServerC\
```

```
Copy SimpleRFDS.FileServer.exe ServerA\
```

```
Copy SimpleRFDS.FileServer.exe ServerB\
```

```
Copy SimpleRFDS.FileServer.exe ServerC\
```

```
cd ServerA
```

```
start SimpleRFDS.FileServer.exe 6666
```

```
cd ..
```

```
cd ServerB
```

```
start SimpleRFDS.FileServer.exe 6667
```

```
cd ..
```

```
cd ServerC
```

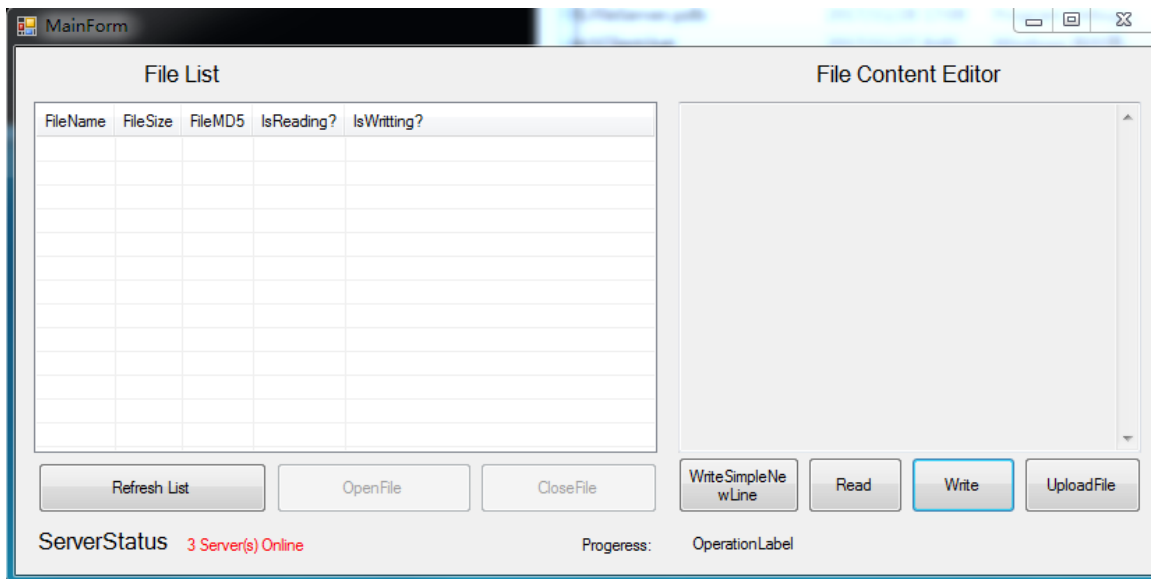
```
start SimpleRFDS.FileServer.exe 6668
```

```
cd ..
```

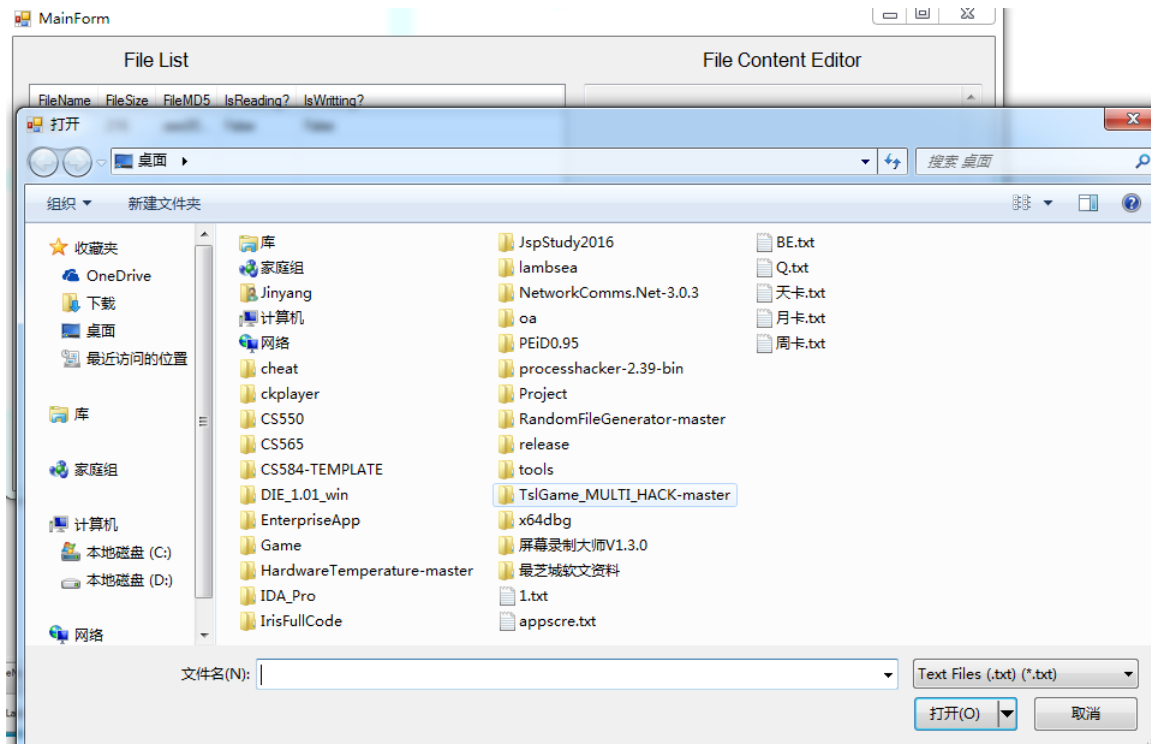
```
start SimpleRFDS.Client.exe
```

Sample Output

Main GUI



Upload A File



Read A file

The screenshot shows the 'MainForm' application window. It has two main panes: 'File List' on the left and 'File Content Editor' on the right. The 'File List' pane contains a table with the following data:

FileName	FileSize	FileMD5	IsReading?	IsWriting?
1.txt	216	aee85...	False	False

Below the table are three buttons: 'Refresh List', 'OpenFile', and 'CloseFile'. The 'File Content Editor' pane shows two URLs: `http://api.frostflames.com/fillup.php?admin=a4501150` and `http://fz.frostflames.com/fillup.php?admin=a4501150`. Below the URLs are three labels: '帐户名称: ljinyang0524', '密码: a4501150', and '联系电子邮件地址: a4501150@mail.com'. At the bottom of the editor are four buttons: 'WriteSimpleNewLine', 'Read' (highlighted with a blue border), 'Write', and 'UploadFile'. The status bar at the bottom shows 'ServerStatus 3 Server(s) Online' and 'Progeress: 1.txt MD5: aee858f7e7185082dd63c58eae3c637c Readed!'.

Write A file

The screenshot shows the 'MainForm' application window. It has two main panes: 'File List' on the left and 'File Content Editor' on the right. The 'File List' pane contains a table with the following data:

FileName	FileSize	FileMD5	IsReading?	IsWriting?
1.txt	216	aee85...	False	False

Below the table are three buttons: 'Refresh List', 'OpenFile', and 'CloseFile'. The 'File Content Editor' pane shows the same two URLs as the previous screenshot. Below the URLs are three labels: '帐户名称: ljinyang0524', '密码: a4501150', and '联系电子邮件地址: a4501150@mail.com'. At the bottom of the editor are four buttons: 'WriteSimpleNewLine', 'Read', 'Confirm' (highlighted with a blue border), and 'UploadFile'. The status bar at the bottom shows 'ServerStatus 3 Server(s) Online' and 'Progeress: 1.txt MD5: aee858f7e7185082dd63c58eae3c637c Openned For Write!'.

MainForm

File List

FileName	FileSize	FileMD5	IsReading?	IsWriting?
1.bt	219	7d8e...	False	True

Refresh List

OpenFile

CloseFile

File Content Editor

http://api.frostflames.com/fillup.php?admin=a4501150
http://fz.frostflames.com/fillup.php?admin=a4501150

帐户名称: lijinyang0524
密码: a4501150
联系电子邮件地址: a4501150@mail.com
dsf

WriteSimpleNe
wLine

Read

Write

UploadFile

ServerStatus 3 Server(s) Online

Progeress: 1.bt
MD5: 7d8e6bbe8d90033e334be26a7b981ec Writted!

Append A Sample Line

MainForm

File List

FileName	FileSize	FileMD5	IsReading?	IsWriting?
1.bt	219	b4ddc...	False	False

Refresh List

OpenFile

CloseFile

File Content Editor

http://api.frostflames.com/fillup.php?admin=a4501150
http://fz.frostflames.com/fillup.php?admin=a4501150

帐户名称: lijinyang0524
密码: a4501150
联系电子邮件地址: a4501150@mail.com
dsf
lm a new line

WriteSimpleNe
wLine

Read

Write

UploadFile

ServerStatus 3 Server(s) Online

Progeress: 1.bt
MD5: b4ddc357411e22832795733d33470126 Added A New Line!

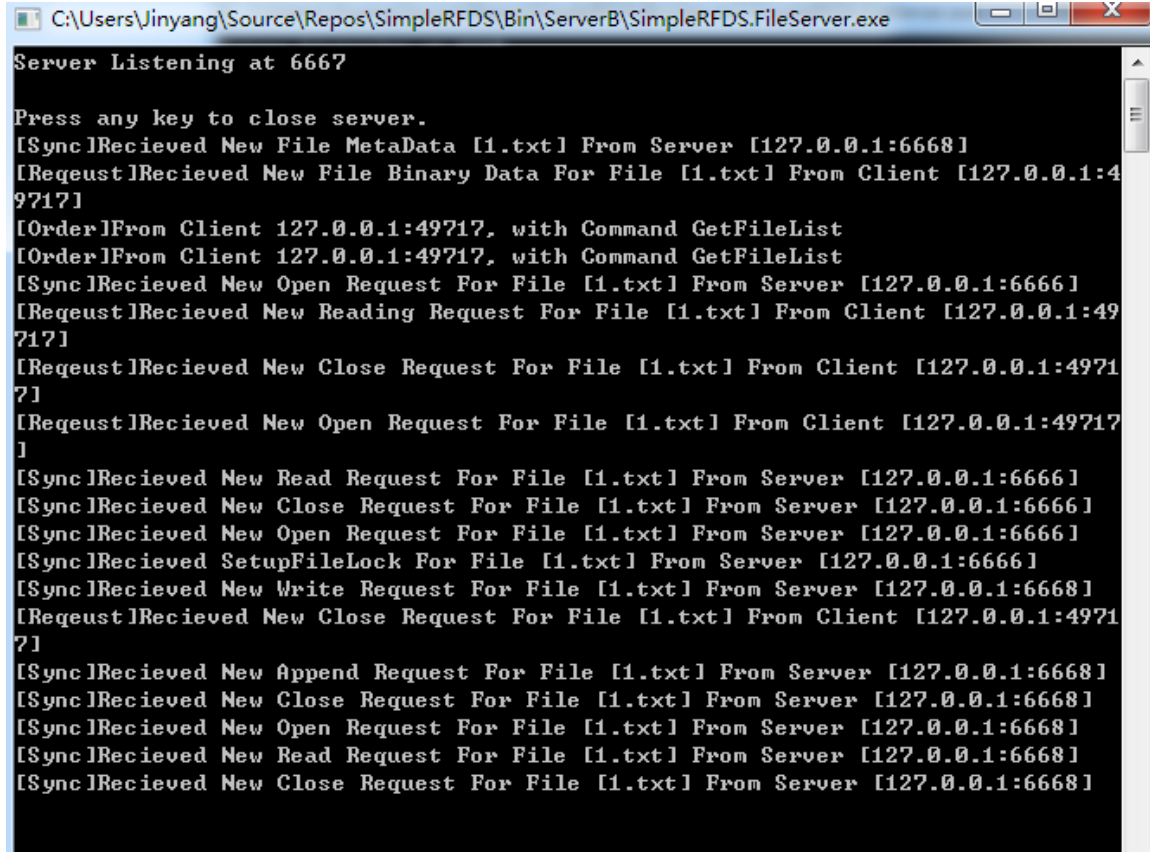
Server Logs

Server Listening At 6668

```
C:\Users\Jinyang\Source\Repos\SimpleRFDS\Bin\ServerC\SimpleRFDS.FileServer.exe
Server Listening at 6668

Press any key to close server.
[Request]Recieved New FileMetaData For File [1.txt] From Client [127.0.0.1:49718]
[Sync]Recieved New File Binary Data [1.txt] From Server [127.0.0.1:6667]
[Sync]Recieved New Open Request For File [1.txt] From Server [127.0.0.1:6666]
[Sync]Recieved New Read Request For File [1.txt] From Server [127.0.0.1:6667]
[Sync]Recieved New Close Request For File [1.txt] From Server [127.0.0.1:6667]
[Request]Recieved QueryFileLock For File [1.txt] From Client [127.0.0.1:49718]
[Sync]Recieved New Open Request For File [1.txt] From Server [127.0.0.1:6667]
[Sync]Recieved New Read Request For File [1.txt] From Server [127.0.0.1:6666]
[Sync]Recieved New Close Request For File [1.txt] From Server [127.0.0.1:6666]
[Sync]Recieved New Open Request For File [1.txt] From Server [127.0.0.1:6666]
[Sync]Recieved SetupFileLock For File [1.txt] From Server [127.0.0.1:6666]
[Request]Recieved New WriteFile Request For File [1.txt] From Client [127.0.0.1:49718]
[Order]From Client 127.0.0.1:49718, with Command GetFileList
[Sync]Recieved New Close Request For File [1.txt] From Server [127.0.0.1:6667]
[Order]From Client 127.0.0.1:49718, with Command GetFileList
[Request]Recieved New Append Request For File [1.txt] From Client [127.0.0.1:49718]
[Request]Recieved New Close Request For File [1.txt] From Client [127.0.0.1:49718]
[Request]Recieved New Open Request For File [1.txt] From Client [127.0.0.1:49718]
[Request]Recieved New Reading Request For File [1.txt] From Client [127.0.0.1:49718]
[Request]Recieved New Close Request For File [1.txt] From Client [127.0.0.1:49718]
```

Server Listening AT 6667



```
C:\Users\Jinyang\Source\Repos\SimpleRFDS\Bin\ServerB\SimpleRFDS.FileServer.exe
Server Listening at 6667
Press any key to close server.
[Sync]Recieved New File MetaData [1.txt] From Server [127.0.0.1:6668]
[Regeust]Recieved New File Binary Data For File [1.txt] From Client [127.0.0.1:49717]
[Order]From Client 127.0.0.1:49717, with Command GetFileList
[Order]From Client 127.0.0.1:49717, with Command GetFileList
[Sync]Recieved New Open Request For File [1.txt] From Server [127.0.0.1:6666]
[Regeust]Recieved New Reading Request For File [1.txt] From Client [127.0.0.1:49717]
[Regeust]Recieved New Close Request For File [1.txt] From Client [127.0.0.1:49717]
[Regeust]Recieved New Open Request For File [1.txt] From Client [127.0.0.1:49717]
[Sync]Recieved New Read Request For File [1.txt] From Server [127.0.0.1:6666]
[Sync]Recieved New Close Request For File [1.txt] From Server [127.0.0.1:6666]
[Sync]Recieved New Open Request For File [1.txt] From Server [127.0.0.1:6666]
[Sync]Recieved SetupFileLock For File [1.txt] From Server [127.0.0.1:6666]
[Sync]Recieved New Write Request For File [1.txt] From Server [127.0.0.1:6668]
[Regeust]Recieved New Close Request For File [1.txt] From Client [127.0.0.1:49717]
[Sync]Recieved New Append Request For File [1.txt] From Server [127.0.0.1:6668]
[Sync]Recieved New Close Request For File [1.txt] From Server [127.0.0.1:6668]
[Sync]Recieved New Open Request For File [1.txt] From Server [127.0.0.1:6668]
[Sync]Recieved New Read Request For File [1.txt] From Server [127.0.0.1:6668]
[Sync]Recieved New Close Request For File [1.txt] From Server [127.0.0.1:6668]
```

Server Listening AT 6666

```
C:\Users\jinyang\Source\repos\SimpleKFS\Bin\ServerA\SimpleKFS.FileServer.exe
Server Listening at 6666

Press any key to close server.
[Sync]Recieved New File MetaData [1.txt] From Server [127.0.0.1:6668]
[Sync]Recieved New File Binary Data [1.txt] From Server [127.0.0.1:6667]
[Regeust]Recieved New Open Request For File [1.txt] From Client [127.0.0.1:49716]
]
[Sync]Recieved New Read Request For File [1.txt] From Server [127.0.0.1:6667]
[Sync]Recieved New Close Request For File [1.txt] From Server [127.0.0.1:6667]
[Sync]Recieved New Open Request For File [1.txt] From Server [127.0.0.1:6667]
[Request]Recieved QueryFileLock For File [1.txt] From Client [127.0.0.1:49716]
[Regeust]Recieved New Reading Request For File [1.txt] From Client [127.0.0.1:49716]
[Regeust]Recieved New Close Request For File [1.txt] From Client [127.0.0.1:49716]
[Regeust]Recieved New Open Request For File [1.txt] From Client [127.0.0.1:49716]
]
[Request]Recieved SetupFileLock For File [1.txt] From Client [127.0.0.1:49716]
[Sync]Recieved New Write Request For File [1.txt] From Server [127.0.0.1:6668]
[Sync]Recieved New Close Request For File [1.txt] From Server [127.0.0.1:6667]
[Sync]Recieved New Append Request For File [1.txt] From Server [127.0.0.1:6668]
[Sync]Recieved New Close Request For File [1.txt] From Server [127.0.0.1:6668]
[Sync]Recieved New Open Request For File [1.txt] From Server [127.0.0.1:6668]
[Sync]Recieved New Read Request For File [1.txt] From Server [127.0.0.1:6668]
[Sync]Recieved New Close Request For File [1.txt] From Server [127.0.0.1:6668]
```

Evaluation

The evaluation of this project is comprised by the following:

- **Performance comparison of file reading / writing speed vs local storage.**

Due to its nature, file reading / writing speed are determined by network latency and server node I/O performance. In overall, the speed is slower than local storage

- **Performance comparison of exiting solutions such as AWS.**

Since AWS are built on high performance server nodes with advanced algorithms, the performance of this system is much slower than AWS, but it is successful as a demo.

- **Demo purpose complement.**

This file system provides transparent replication. Files are replicated in server node's physical hardware file system but Client can only see one copy of the file.

Only one Client are permitted to edit the same file as same time.

When the file is being writing, other clients **cannot** open it for writing / reading at same time.

When the file is being reading, all other clients **can** open it for reading at same time.

As long as network has at least ONE server online, all the operations are functional. When one server failed with crash, the user can still access to file data.

As far as project handout requires, all requirements are met.

DEMO VIDEO CAN BE DOWNLOAD AT

<http://frostflames.com/demo.mp4>

Conclusions

Although this is just a simple implementation of replicated file distribute system, it would be a nice practice regarding cloud computing and parallel computing. Our programming model will be designed with common scenarios in mind. What will users typically want to do with our system? Which features are more important for advanced users and which features should be very easy to use? Note that the focus in this project will be on the features and user experience not on the internal objects, data structures, or functions. We strongly believe that with this system, users would have one more way to store their files and have a start experience with replicated file distribute system and cloud computing.

What I learn from implementation

The DFS provides the mechanism by which the file systems of these physically dispersed units are integrated into a single logical unit. This provides the transparent distribution to user applications. This transparency provides users the ability that not having to remember the details of the real file locations and its status. In addition, automatic storage replication and an atomic transaction mechanism provides high reliability and improved availability in the distributed environment in case of server node failure.

A distributed file system's development motivation is to provide information sharing and fault recovery. This motivates forms such as the system requirements. Those are:

.

- * Transparent Data Replication

- * Consistency of File

- * Reliability Over Internet

Transparent Data Replication allows a user or administrator to retrieve/upload file object in a way that just like operation in local file system. Therefore, Transparent Data Replication results in being able to open a file in precisely the same manner independently of whether the file is local or remote i.e. issue the same system call, with the same parameters in the same order etc. If open 'filename' is used to access local files, it is also used to access remote files, which provides a consistent access experience against local file. The replicated files in network thus become invisible. This is achieved by abstract file into a DFS file object and associated binary data. Users have ability that they are unaware of the underlying network and that their files are physically distributed over several sites to ensure data safe.

Consistent file means that every user who uses the system obtained exactly the same text for a file system object, regardless of where it exists. The system must thus guarantee that a given file name will access a particular file from anywhere in the network. This is achieved by site independent hash calculation of files. Every file has a unique global hash value and the system maintains a mapping between these hash values and physical file objects. To increase availability and reliability of files, provision is made in the DFS to replicate files at several sites. In this way, if a site fails, other sites will continue to operate and provide access to the file. Availability and reliability of a file can be made arbitrarily high by increasing the order of replication. Since data is replicated at one or more sites, the system ensures data consistency by keeping these file images consistent. Consistency is maintained by file names and file hashes. We all know that when one bit of a file is changed, the hash of a file is changed. Thus, the system can easily maintain the file consistency.

There is a typical failure in common distributed file system, which is, if site A can communicate with site B and site B with site C, then site A is assumed to be able to communicate with site C.

The converse of this assumption of transitivity states that if site B is not accessible to site A then no other site can communicate with it either, however, in currently design, I'm treated each node as independent node, which would communicate with ALL nodes at same time. This design are elimination of assumption that to recovery protocols wherein if a site is detected as having crashed by any other site, then it is assumed that none of the remaining sites in the network can communicate with the crashed site. Replication at the file and directory level is provided to increase file availability and reliability in the face of node failures.

Reliability is based on transaction mechanism which allows all operations bracketed within the 'begin trans' and 'end trans' calls to execute atomically and broadcast to all server nodes. This transaction mechanism supports failure atomicity, wherein if a transaction completes successfully, the results of its operations will never subsequently be lost and if it is aborted, no action is taken. Thus, either all or none of a transactions operation are performed. The transaction mechanism is built on top of a locking mechanism and the system automatically locks data base files in the appropriate mode when they are opened for reading or writing. Deadlock problems are avoided by ensuring that processes never wait for files to be unlocked, and a file is locked message is returned if files are not available for operation.

Protection has not been considered an issue in this thesis. All files are created readable and writeable by everyone. Authentication can be implemented at a later date by making a few changes to the abstract file object and assign a unique username as key. This code can then be checked on file access to see whether the user has valid permission to read or write the file. In the current implementation as long as the file exists in the one of the node, file access can take place and no authentication is made.

In order for a distributed system to be convenient to use, it is important that there be a global way to identity files, independent of where they are stored. This is accomplished in the DFS by organizing the files into a single hash map maintained by each server node. Thus, the users view of the file system sees globally unique names in a single, uniform hierarchical name space however in server node view, all files are identified as unique hash values. Nodes maintain their own hash map, but update it concurrently.

Additional Resources

Timeline

Week	Milestone
1	Research, gather info and prepare those materials.
2	Write software requirements, clarify requirements, build necessary environment
3	Gathering existing solution, Prototype
4	Build user interface, Write server-side program
5	Write server-side program, Build client-side logical program
6	Build client logical program, do evaluation
7	Write final report, search defects
8	Write final presentation

Deliverables

One final report in PDF form.

One final PowerPoint presentation.

Source code.

All can be found in same zip file.

Reference

[1] 10 Benefits Of Cloud Computing. <https://www.salesforce.com/uk/blog/2015/11/why-move-to-the-cloud-10-benefits-of-cloud-computing.html>

[2] Amazon Web Service Website: <https://aws.amazon.com/backup-recovery/>

May use later:

IEEE Micro

IEEE Internet Computing

IEEE Concurrency

Journal of Parallel and Distributed Computing.