DROPBOX

Yan Li (yan_li@brown.edu)

Brown University

Introduction

Dropbox is a small project I am developing not for any profits but just for fun. It basically simulates what the real Dropbox does. In addition, this project is still in progress but I'm open to share the code with you. I have already finished the basic infrastructure and the network protocol. The file could be synchronized to each side (server and client) now. This project is attractive to me because it involves many aspects: synchronization between threads, synchronization between server and client, distributed data storage, and load-balancing design.

In my Dropbox, there could be multiple users using the service in the meantime, each of which synchronizes its files with different file server that is a data storage server in my design. There is one master server who is responsible for handling connection request from clients, and also deals with the connection between file servers and master server because master server must make sure that file servers are healthy.

Each time, a client must connect to the master server firstly, then the master server receives the identification data from the client. The next step for master is to find the correct file server this client was previously attached to and ask for the file server to do identification. If the file server passes the identification, it will give a response to master and master will give the client the IP address of the file server to the client. The client then starts synchronization with the file server directly.

System Design

I learned the design from **GFS**. Here I applied some of the ideas in that paper into my project.

Master Server: responsible for building the connection between client and file server. Whenever a file server boots, the file server will connect to the master and the connection between them must be maintained.

Master needs to periodically send "Heartbeat" message to request any updates from each file server. Also, whenever a file server change its setting, it will send the changes to the master server as well.

File Server: the idea of file server is just to simulate storage servers. They could be remote servers. Each file server can synchronize with multiple clients and also keep their files in its machine. For now, for better testing, file server can accept user input to do some operations manually. File server also stores the name and password for each client that stores data in its machine, and clients' information has to be added manually.

Client: each client has a user name and password. After the user runs the client, it will automatically connect to the master server with its identification data sent. The master server then finds the file server in which stores the same identification information and gives the IP address to the client to start the synchronization process.

Protocol Design

In client side, there are two hash tables. One records all of the file in the Dropbox Directory in the last synchroization, the other keeps the file in the current synchroization. For every synchronization, client compares these two hash tables and decide the files to be deleted, added and updated. For deleted file operation, client only needs to send the relative file path, and then server will delete it. For newly created/updated file, client will send the file path and the content to the server.

The communication protocol for synchronization is like this:

Communication protocol:

FileNumber (4 bytes) -- the header of this protocol, tell the server how many file needed to be synchronized

File Operation(1 byte) -- delete, create or update, if the operation is create or update, the rest part of protocol is like this:

Data synchronization protocol:

Length of file path(4 bytes) -- tell the server how many bytes to be read in the stream to get the file path

File path (depends on the length)

File last modified time (8 bytes)

IsDir (1 byte) tell the server the file is directory or file

File size (8 bytes) tell the server how many bytes to be read in the stream to get the file content

File content (depends on the size)

If the operation is deleted, the rest part of protocol is like this:

Length of file path(4 bytes) -- tell the server how many bytes to be read in the stream to get the file path

File path (depends on the length)

So for example, if one file called "deleteme" is deleted, one file called

"createme.cpp" is created, and one file called "updateme.cpp" is modified, the protocol is:

FileNumber: 3	-- how many file needed to be synchronized
DELETE(1 byte)	-- delete, create or update
8	-- the length of file path
deleteme	-- file path
CREATE(1 byte)	-- delete, create or update
13	-- the length of file path
createme.cpp	-- file name
12:30, 2011/1/1/	-- last modified time
FALSE	-- is Dir?
10	-- file size
123456	-- the content of the file
UPDATE(1 byte)	-- delete, create or update
13	-- the length of file path
updateme.cpp	-- file name
12:30, 2012/1/1/	-- last modified time
FALSE	-- is Dir?
10	-- file size
123456	-- the content of the file

Coming features

Here I have some features in my mind that I'm extremely interested in:

1. Large file transmission and speed up
2. Multiple platforms (iOS)
3. Better crash resiliency
4. Encryption

Code Info

4000 – 5000 lines of JAVA code

Files

All of the files are written by myself!

Execution

I would suggest not execute the program because you must take care of a lot of arguments.