5105 Project 3 Report

(By Aparna Mahadevan, Eric Kuha, and Ming-Hong Yang)

Components & Functionality

Note: the words “file server” and “client” are interchangeable
The tracker server is run first. InitServer creates a socket that is bound to the port and listens for incoming connections from servers. StartListening() then accepts connections. For each accepted connection, a TrackingServerHandler thread is started. The handler waits for each file server to send its port to the tracker, after which the tracker creates a unique socket to connect to the server. The handler thread on the tracker waits for messages from the file server. For each message received from a file server, the command is dealt with accordingly. When a client wants to find a file, it will first send the tracker the filename with “find;” appended to the front of the message. In the trackerserver handler, the “find” command calls FindFile, which is defined in func.cpp. For each client in the vector, FindFile checks if findFile (which is defined in Client.cpp) returns true. Each instance of Client.cpp has the client id, port, and ip, along with a vector storing the file list for that client. findFile looks for the file in the file list vector, and if it exists, returns 1. Then FindFile gets that client’s ip and port, and appends it to the list of servers that have the file. This server list is saved in the buffer and sent back to the file server.

When a client wants to register, ConnectClientToServer is called, which in turn calls ConnectToServer (to set up the socket communication), and UpdateList with the folder name, which is defined in func.cpp. This in turn calls GetFileList (which is defined in file.cpp) that opens the directory and returns a list of the files to UpdateList. UpdateList then sends this list of files to the tracker with “register;” appended to the front of the buffer. In the tracker handler, the “register” command saves the list of files received from the file server into a vector. It then calls updateFileList for the client to save it to the client file list.

Similarly, the file server is run and the ConnectClientToServer function is called, in which the file server connects to the tracker using the tracker’s IP and port. It sends its port to the tracker. It also starts a thread which calls StartListening() to wait for incoming connections from the tracker and other peers. After a tracker (or another peer) is connected, it starts a FileServerHandler thread which waits for messages from the tracker/peer. Similar to the tracker, for each accepted connection, a FileServerHandler thread is started. The handler thread on the server waits for messages from the tracker/peer. For each message received from a client, the command is handled accordingly.

When the client wants to download, it calls the download function defined in fileServer.cpp. This function first issues a “find” request to the tracker to find out where the file is located. Then for each server on which the file exists, it will call GetLoad to get the load of that server (on the file server handler, “get load” will return the load of the file). It also gets the latency, and then decides the server to download from. It chooses the server with the lowest latency * load. The reason for this formula (and not an addition) is that this way, the load and latency are equally weighted in the equation. As such, multiplication gives a better estimate of the load and latency. Then, dl_check is called with the file name and the ip and port of the selected server. In this function, a message with “download;” appended to the front, and the file name is sent to the file server containing the file.

On the file server handler, “download” command will call get_hash on a string containing the machine ID and the filename, which returns the checksum. The checksum is then saved to a buffer and sent to the file server along with the file. It also introduces random errors for checksum testing.

The function dl_check receives the file, recomputes the checksum, and compares it with what was received. If they match, it returns true. Otherwise, it returns false.

5105 Project 3 Report
(By Aparna Mahadevan, Eric Kuha, and Ming-Hong Yang)

Crash Recovery

<u>Tracker</u>
When the tracker goes down, all the clients block until it comes back up. In the fileServerHandler function if there is a recv error from the tracker, they sleep and try to reconnect by calling ConnectToServer in a loop.

<u>Peers</u>

How to Test It

First “make” the project.

To initialize the files for testing, there are two shell scripts. Running “sh reset.sh” will distribute the files between the servers. Running “sh reset1.sh” will put all the files in one folder so that there is only one copy of each file and they are all on the same server. The other servers are initially empty in that case.

To test the system, first run “./tracker PORT”.

Then on another terminal window, run “./client 127.0.0.1 PORT FOLDERNAME”

The client process will prompt you to enter commands.f

Currently, the latency.conf file contains the latency values for four “clients” and their folder/machineID names, alice, bob, carol, and chuck. So, an example:

```
./tracker port 8080
./client 127.0.0.1 8080 alice
./client 127.0.0.1 8080 bob
```

etc

Test cases

First, run the tracker. Then:

1) Run two-four clients (with different folders), and try doing find for each of the files in their folder.
2) Run two-four clients. From one client, try to download a specific file from the other client.
3) Run two-four clients and try to find a file a file that doesn’t exist in either of their folders. You should get an “unable to find file” message.
4) Run two-four clients, execute a command (find/download), then do a SIGINT to stop the tracker. The clients will sleep. Then rerun the tracker with the same port as before. The clients should reconnect to it. (This is slightly unreliable and may depend on your machine. It works if the tracker is able to reclaim its port number. It is not always able to do this).
5) Run two-four clients, then do a SIGINT on one of them.