CS4532 Distributed File System

Jarrod Richardson - 12302939

1 Introduction

The assignment was to design and implement a distributed file system exhibiting a range of properties. The system was constructed using REST services, written in Haskell using the Servant library. These properties included:

1. A Security Service that provides authentication for clients and encrypted session for sending the encrypted messages and data during the communications.
2. A Directory Service that provides access to the files located on several File Servers to clients.
3. A Transaction Service that logs and keeps track of transactions (multiple file requests).
4. A Proxy Service that communicates with the clients and the other services and perform actions on behalf of the clients.
5. Multiple File Servers that each have a set of files.
6. A Replication Service that keeps backups of the various File Servers.

The project repository is located at: https://github.com/Coggroach/Gluon. This report file is located at: https://github.com/Coggroach/Gluon/blob/master/Report.pdf. The notable technologies used for this assignment included:

1. The Haskell Platform.
2. Haskell Stack.
3. Haskell Servant.
4. MongoDb and Haskell Connector.

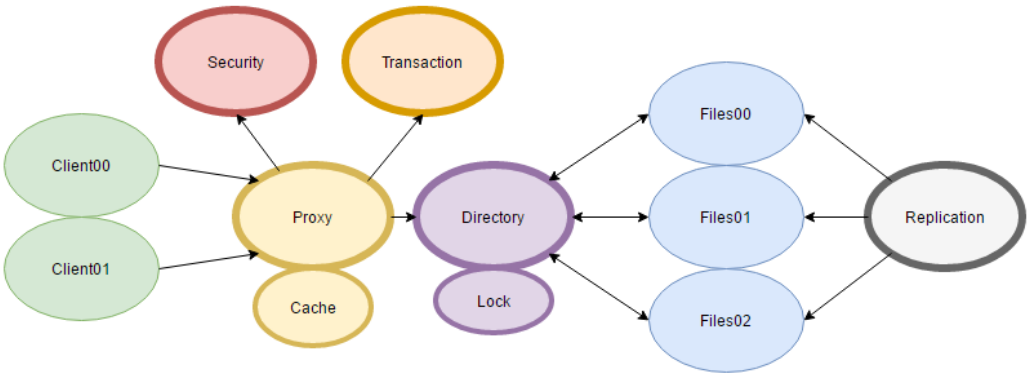

Figure 1: Relationship Diagram for the Distributed File System

1 Bold Outlines indicate local databases in MongoDb. 1way arrows indicate A calls B's restful services. 2way arrows indicate A calls B and B calls A's restful services.

1 The Project Files:

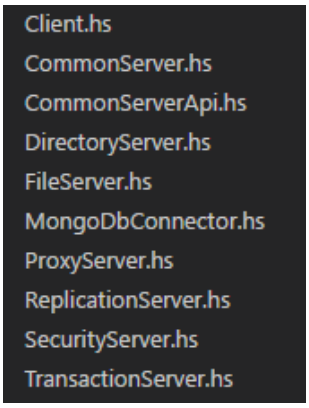

Figure 2: Haskell File Structure for the Project

2 CommonServer.hs

The Common Server is not a run time server. It contains all the common type definitions, global variables and functions that each of the other servers use. This prevents duplicate code between the other servers.

Types defined in CommonServer.hs include:

- File
- Identity - Server Identity: Address, Port, ServerType.
- ServerType - enumeration of server types: FileServer, DirectoryServer, etc.
- Client - unencrypted username and password.
- EncryptedClient - encrypted client.
- ClientRequest - request + encrypted client.
- ClientFileRequest - file + encrypted client.
- Ticket - encrypted id and timeout.
- Session - encrypted ticket and session key.
- Response - ResponseCode, Identity sender, response data.
- ResponseCode - list of codes that correlate to a response.

3 CommonServerApi.hs

The Common Server Api is the repository for all the API definitions. It also includes the calling as a client functions. The list of Apis and their routings as follows:

- FileApi - files, download, upload
- DirectoryApi - files, open, close, join
- SecurityApi - login, register, contains
- ProxyApi - login, files, open, close, begin, end
- TransactionApi - begin, end, commit

4 FileServer.hs

The File Servers have been implemented as flat files. This allows each file server connected to the service to be stored in a unique directory and provides an easy means of mapping the drives for the directory server. This keeps track of files and eliminates any concerns about directory trees for each server. These files are created by the file server and are stored locally on the disk. When a File Server is executed; by means of a 'join' function, an attempt is made to join the Directory server's list of online file servers currently active. Each File Server sends the Directory Server it's IP address and port number, which are stored in the mongoDB table. The File Server has now joined the services and its

files and services are accessible to the proxy. This will only happen if there is a positive response from the Directory Server and the Directory Server is online.

The File Server performs the following operations:

1. getFiles (files)
    - Inputs - The Ticket.
    - Outputs - An array of strings of all file names in the local directory.
    - Function - Retrieves an array string of all files in the file server.
2. downloadFile (download)
    - Inputs - The Ticket and the File Name
    - Outputs - The Corresponding File of that File Name from the local directory.
    - Function - To Decrypt the File Name and return an Encrypted File that it read locally.
3. uploadFile (upload)
    - Inputs - The Ticket and the File
    - Outputs - A Response with a Response Code.
    - Function - To Decrypt the File and save it locally, then return a Response

5 DirectoryServer.hs

The Directory server performs the function of both a hub for the users requests and as a user interface for the file server. All I-O (input-output) requests are sent to the directory server which then delegates the request to the appropriate file server. Through the use of a MongoDB Instance, this keeps track of which file servers have joined the service and what files are stored in each file server. The File servers and file mappings are stored in separate records.
When a request is received from a client to download a file, the Directory Server would need to search the file mapping in order to determine which file server the file is located on. When and if found, the Directory server will download the file to the requesting client.
When a request is received from a client to upload a file, the Directory Server will randomly choose a file server that is already connected to store the file. File mapping is inserted into the MongoDb instance for reference to the files location.

5.1 Locking

Locking is performed on the directory server at the File Mapping level. Files can be "Locked", "Unlocked", or "Read Only". Once a file is requested to be opened it enters the "Read Only" state from the "Unlocked" state. Closing the file will move the state of the file into the "Unlocked" state again. Ideally the directory server would "Unlock" a file after a Client's Token has marked them as expired/timed out.

The Directory Server performs the following operations:

1. getFiles (files)
    - Inputs - The Ticket.
    - Outputs - An array of strings of all file names of all the file servers.
    - Function - Retrieves an array string of all files from all the file server. This is read from the local mongoDb database called "FileMappingDb".
2. openFile (open)
    - Inputs - The Ticket and the File Name
    - Outputs - The corresponding File of that File Name
    - Function - To query the local mongoDb database "FileMappingDb" for the matching file server containing the File of File Name and download it from the corresponding file server.
3. closeFile (close)
    - Inputs - The Ticket and the File
    - Outputs - A Response with a Response Code.
    - Function - To query the local mongoDb database "FileMappingDb" for the matching file server containing the File of File Name and upload it to the corresponding file server.
4. joinServer (join)
    - Inputs - An Identity of a file server.
    - Outputs - A Response with a Response Code.
    - Function - To insert a new file server entry into the mongoDb database "FileServerDb"

6 SecurityServer.hs

The Security Server's function is to provide a login and authentication check on a Client. A new Client can choose to register. An exisitng Client will login with their encrypted password. The Security server will decrypt and validate

the user to ensure their existence.
A sessionKey is randonly generated for a valid Client which will act as an encryption key for all messages the Client sends to the Servers. A Client is given half an hour to complete their tasks before a session timeout is performed.
The encryption method used was a simple XOR function. This provides a simple method for symmetrical key encryption. Athough not the most secure method, it was felt given more time, a more secure method could be implemented, but was not needed based on the scope of the project.
A secret server key was used instead of an individual one also because of time frame and simplicity, however, once again, individual keys could easily be used.

The Security Server performs the following operations:

1. login (login)
    - Inputs - The Encrypted Client.
    - Outputs - The Session of that Client.
    - Function - To check whether the client is registered and if so return a new Session otherwise a failure Session.
2. register (register)
    - Inputs - The Client.
    - Outputs - A Response with a Response Code.
    - Function - To register a new client with the security server.
3. contains (contains)
    - Inputs - A Client's Name
    - Outputs - A Response with a Response Code.
    - Function - To check whether a client with the given name is registered on the security server. Returns a "SecurityClientNotRegistered" or a "SecurityClientRegistered" Response.

7 TransactionServer.hs

The Transaction server is used to update existing files. The 'live' file is not actually updated until the user chooses to commit the transaction. All changes are kept on a temporary file and once the commit is executed, the updates occurs and the temporary file is deleted.

8 ProxyServer.hs

The Proxy Server is a client side program which is executed by any user wishing to gain access to the distributed file system. It issues instructions to the client to perform the desired tasks and provides a text-bases interface to the distributed file server.
The Proxy Server gives the client the choice of logging in or registering. Authentication is carried out. In the event of a new account, once approval of the new account is given, the user will have to login. Having logged in, the cache is initiated and the user can continue.

The Proxy Server performs the following operations:

1. loginClient (login)
    - Inputs - A Client Request.
    - Outputs - A Response with a Response Code.
    - Function - Checks to see if the Client is registered on the security server, if not registers and logs in.
2. getFiles (files)
    - Inputs - A Client Request.
    - Outputs - An array of strings of all file names of from the directory server.
    - Function - Downloads the array of strings of the file names from the directory server.
3. openFile (open)
    - Inputs - A Client Request.
    - Outputs - The File requested.
    - Function - Downloads the requested file from the directory server.
4. closeFile (close)
    - Inputs - A Client File Request.
    - Outputs - A Response with a Response Code.
    - Function - Uploads the attached file to the directory server.

9 The Client

To Communicate with the Proxy Server as a Client, the client needs to send a post request with the Client Request JSON object as a ReqBody. First the Client should attempt to login to the system using "login" from the ProxyApi. The Client then should download the list of files "files" from the ProxyApi. The client is then free to "open" and "close" an files it wishes whilst it is logged in.

10 MongoDbConnector.hs

The MongoDbConnector contains the global database information and functions used to help query the databases. These functions include:

- connectToDatabase
- drainCursor

11 Main.hs

The main file takes an argument that will start up the relevant server. The start command is as follows: "stack exec Gluon-exe 'ServerType'". 'ServerType' is defined as follows.

- FileServer0 - Start a File Server 0
- FileServer1 - Start a File Server 1
- FileServer2 - Start a File Server 2
- DirectoryServer - Start the Directory Server
- SecurityServer - Start the Security Server
- ProxyServer - Start the Proxy Server

The order in which they need to be started:

1. SecurityServer
2. DirectoryServer
3. ProxyServer
4. FileServer[0,1,2]