# CS4032 - Distributed File System - Report

Killian Davitt - 13319024

February 8, 2017

## Contents



- Killian Davitt, 13319024

## 1 General Overview

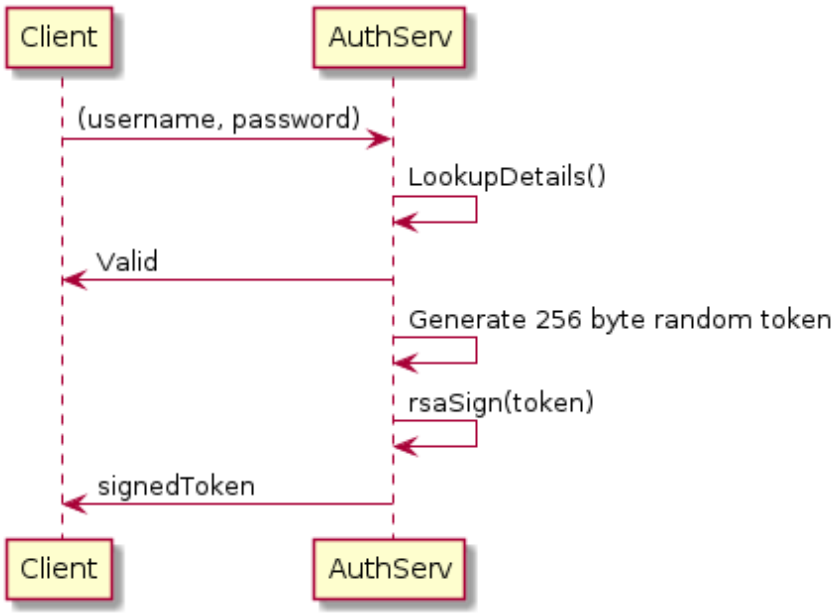
The project, written in golang consists of 5 service application as well as 1 client program. The services are intended to be run with docker and docker-compose.

If you would like to run the project, you can either install golang and all of the dependencies, or use a prepackaged image which I will provide. The images may be run using the docker-compose files in each service's directory.

The vast majority of the features described below are fully implemented in my program. The code for transactions requires a bit more work.

## 2 Authentication & Security

Security and authentication of the different services was mostly done using x509 certificates and RSA keypairs. This allowed for an incredibly robust security, but only required initially authenticating once per server. A fully configured x509 security system requires minimal effort to use after initial configuration.




The use of a signed token gives us an excellent advantage in that the auth server does not have to distribute the token amongst the other servers. If the client provides the token to any other server, they can immediately tell if it has come from the auth Server by attempting to decrypt the signature. If a decrypt is successful then the token is valid, if it does not, then it is a forged token.

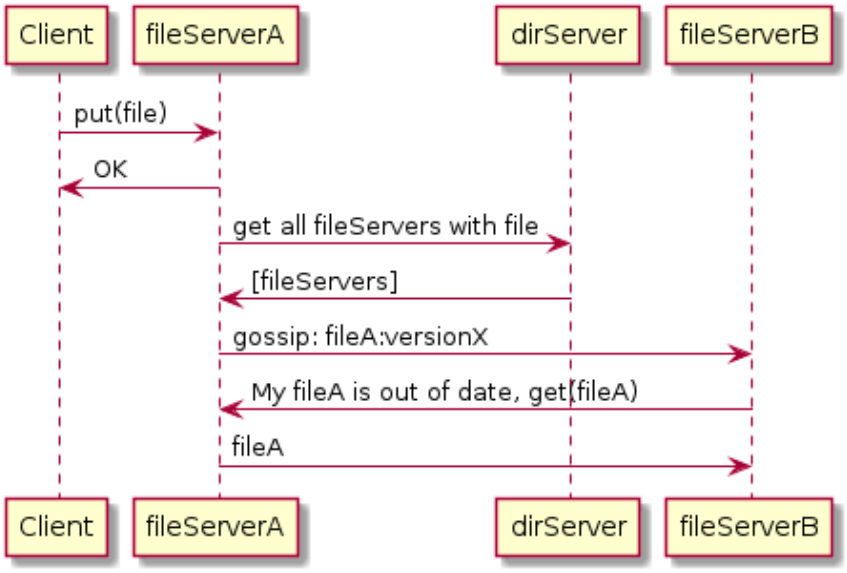
Another crucial piece of the system is that **ALL** communications are conducted over TLS (HTTPS). x509 TLS certs are already in place from the above system so they are used to conduct TLS. The auth server acts as a certificate authority. If the auth server establishes trust with a particular

server, it will sign their x509 cert as a show of trust. This means that any other server in the system can now trust that server as they have cert signed by the CA.

It should also be noted that the security and auth system is implemented using the robust libraries of golang/tls, golang/x509, openssl, etc. This ensures an incredibly high level of reliability compared with building our own crypto from the ground up.

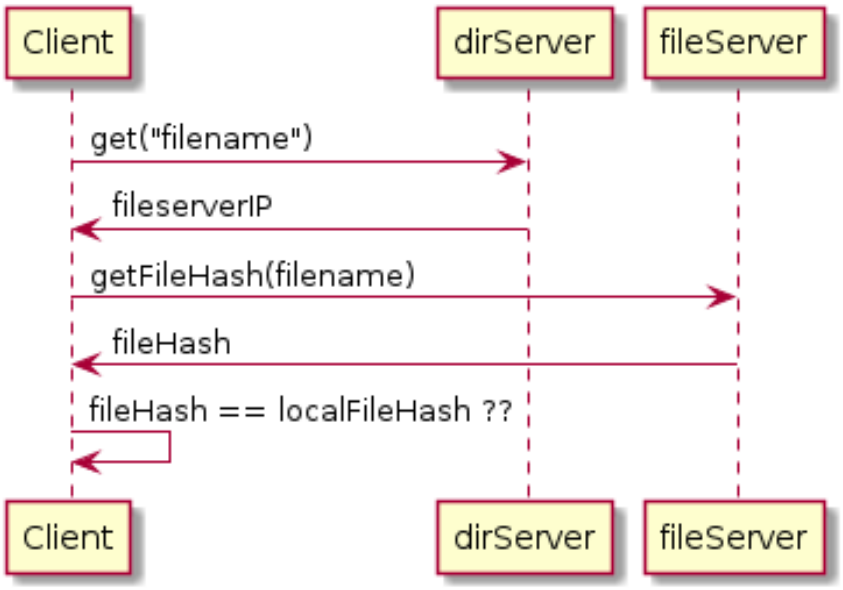
## 3 Replication

A Gossip like system was implemented. Upon a fileserver receiving a file. It asks the directory server for the list of fileserver ip addresses. Then file server then sends a request to send gossip to 3 of the file servers. If the target server responds postively, then the fileserver sends on the latest version of the file. A simple file versioning is used to ensure that the most up to date files are stored.



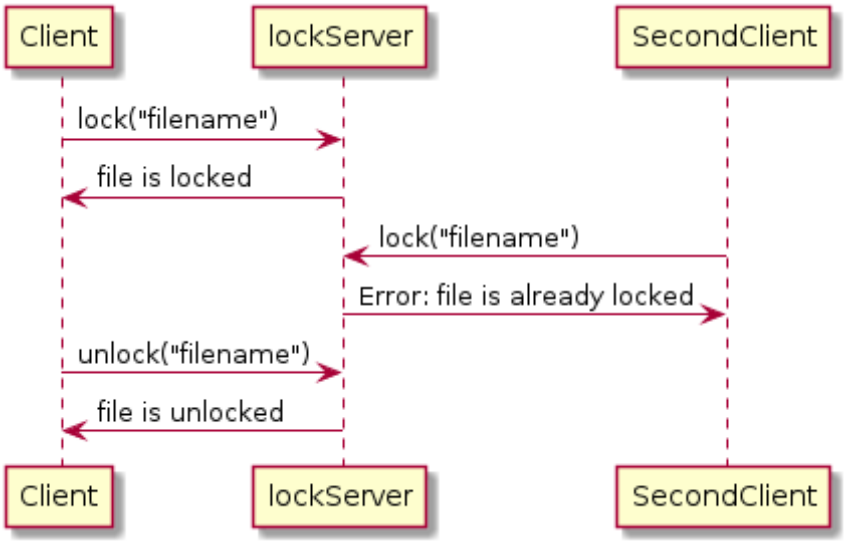

## 4 Caching

Files are cached on the client side based on the hash. A client can ask a file server for the most recent file hash. If the hash is different to the one of the file stored locally, the cache is considered to be stale and the latest file is pulled from the file server



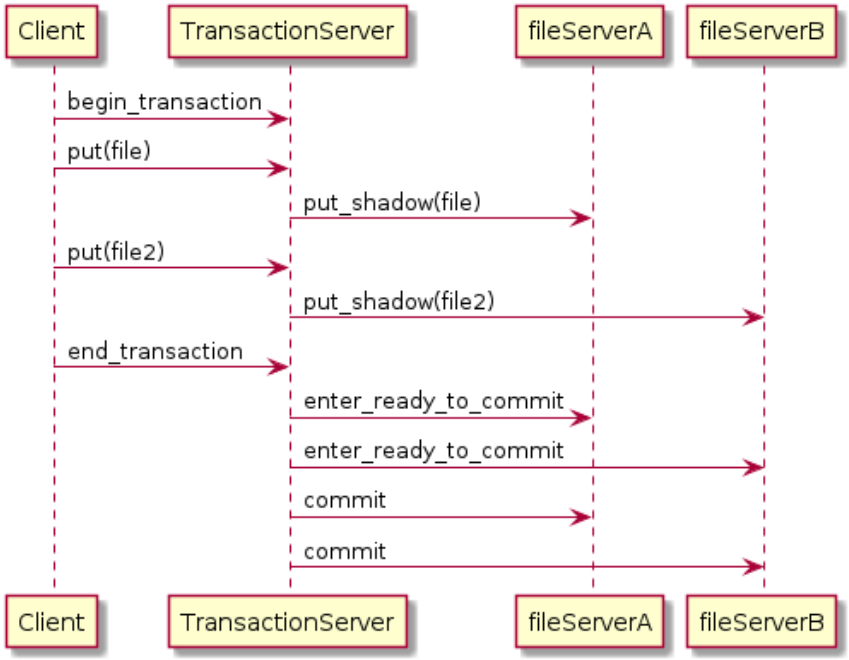

## 5 Locking

An advisory locking system was implemented using the lock service application. The lock service accepts requests from clients to lock a file. The lock service then records locally that the file was locked by a user a. If another client b attempts to lock the file the lock service will see that the file is already locked and will inform the user of such.




## 6 Transactions

Transactions are implemented in a fashion very similar to what was described in the assignment specification. A transaction server is hosted and any client looking to begin a transaction will ask the transaction server to begin a transaction. If there is no ongoing transaction, the transaction server will issue a transaction id to the client and the client will present this id in all further communication with the transaction server.




Any write operations that the client then wishes to make will be directed at the transaction server. The transaction server will attempt to complete these operations with a given fileserver although the fileserver will make these alterations 'in shadow' i.e. the fileserver will make a note that it has been asked to make the change, but it will not commit the change to the physical filesystem.

When a client has finished making operations, it will tell the transaction server to end the transaction. The transaction server will then ask all the involved fileservers to enter the 'Ready to Commit' state.

# 7 Client Proxy

The client proxy, or client library is contained in the directory utils/client. It implements get, put, list, lock and unlock methods. The get, put and list methods contact the directory server directly, a response is returned from the directory server. In the case of get and put, the response from the directory server contains the ip of a file server where the client is to complete its task.

### 7.1 Put file

After contacting the directory server to put the file, either the file already exists or it does not. If the file already exists, the the ip of the fileserver which already contains the file. Otherwise the directory server will assign a random fileserver from it's list of fileservers from which to give the client.

After the client has the ip of the fileserver it needs to contact, it simply calls the $\text{put}_{\text{file}}$ endpoint on the fileserver and the fileserver saves the provided file to disk.

### 7.2 Get file

Get file is similar to put file, however if the directory server does not have a record of the file it will return error to the client. If the directory server does return the ip of a fileserver the client contacts that fileserver with the $\text{get}_{\text{file}}$ endpoint and the fileserver returns the file to the client

### 7.3 List files

To list files the client contacts the directory server and asks for a list of files. The directory server returns the list and the client displays it.

## 8 Client example

The client example is a simple implementation of the above client proxy. It implements an 'sftp' style command line interface and provides all of the methods of the client proxy directly to the user.