Name: Edwin Kaburu

Date: 11/18/2022

Lab 6: Extra Credit

Project Proposal: Simple Cloud Storage (SCS Platform)

Python Environment

For this project, I am taking inspiration from online storage platforms such as (google drive, Microsoft drive, drop-box, and others). In these platforms, you simply upload a file, and it is saved in the "cloud", with the added convenience of accessing your stored file from anywhere in the world; while at the same freeing up disk space on your local machine.

From what I have learned from distributed systems, the file being uploaded is distributed among various machines that are working with each other within a network. While ensuring reliability, robustness, extensibility and providing the invisible illusion of a single coherent system to the user. For this lab, I will attempt to build such a system.

System Composition

The entire system, SCS platform, can be viewed from 2 dimensions, we have the frontend and the backend. The front end is responsible for interfacing with the user; while the backend is what the user does not see. Only 1 node machine is required to interface (collecting commands) with the client-user.

The backend contains machines/nodes that knowns every other machine in the network. Only the leader is the one responsible for interfacing (gathering user commands) with the client-user. The leader ensures their position by occasionally sending heartbeat messages to all other nodes in the network. Any node in the network can become a leader through an election.

Election

The election process is straightforward, if for **some time a node has not received any heartbeat** message, they will **initiate an election**. The message will be sent to every node in the network; when a node **receives an election message, with a lower node's identity**, they will send an **acknowledgement message** back to the sender (with the lower id). They will proceed after sending the reply, by initiating an election. If the node **does not get any response** from its **election message**, for some time, **it has won the election**, the node will send a leader's heartbeat message to **all nodes**. If the **node gets an acknowledgement message**, they will be in a waiting-for-election-results state, **waiting for a declared winner or chosen leader of election**. (Similar to **Lab 2**). The leader on top of sending a heartbeat message to every node in the network, will also inform the client-user they are the leader (node to interface with**, send their commands to**)

File State

To make things interesting, every node in the network will have a **maximum number of files** they can store. When a file is uploaded, **the leader will hash the filename**, followed by **saving the file on hard-disk**, while also **recording a map to it as a key-value pair, (key-> hashed-filename-id, actual-path-to-file-on-disk)**, only when it has not reached its maximum-num-files-capacity. **Only the leader is responsible for hashing the filename of the file**. The leader will then send the file (along with the hash-id), to nodes in the network, and ask them to save a replica copy. When other nodes apart from the leader get the replica message, they will add the file to their hard disk space while record a map to it in their own key-value-pair chain (key-> hashed-filename-id, actual-path-to-file-on-disk). After this will send a response message back to the leader after saving the file.

To retrieve a file, a **filename is given, the filename will be hashed to key**. The key will be used to search for a file path within the dictionary and retrieve the file from disk, which will be sent back to client. The hashed key can also be used to change the state of file such as deleting. If the node does not contain the file, **it will ask other nodes to find the hash-filename-id** and send the **file back to client**. If the search turns up empty, the file does not exist in the distributed hash database (error message saying this to client).

Interfacing

The client will only send commands to the leader on what they want to do. The leader will take those commands and execute them (see previous section). **Any nodes, including the leader, that have the result can respond to the client**. The client will maintain a **listening port, to fetch responses**, and a **command request port to send commands** to the network chosen leader.

Client Commands

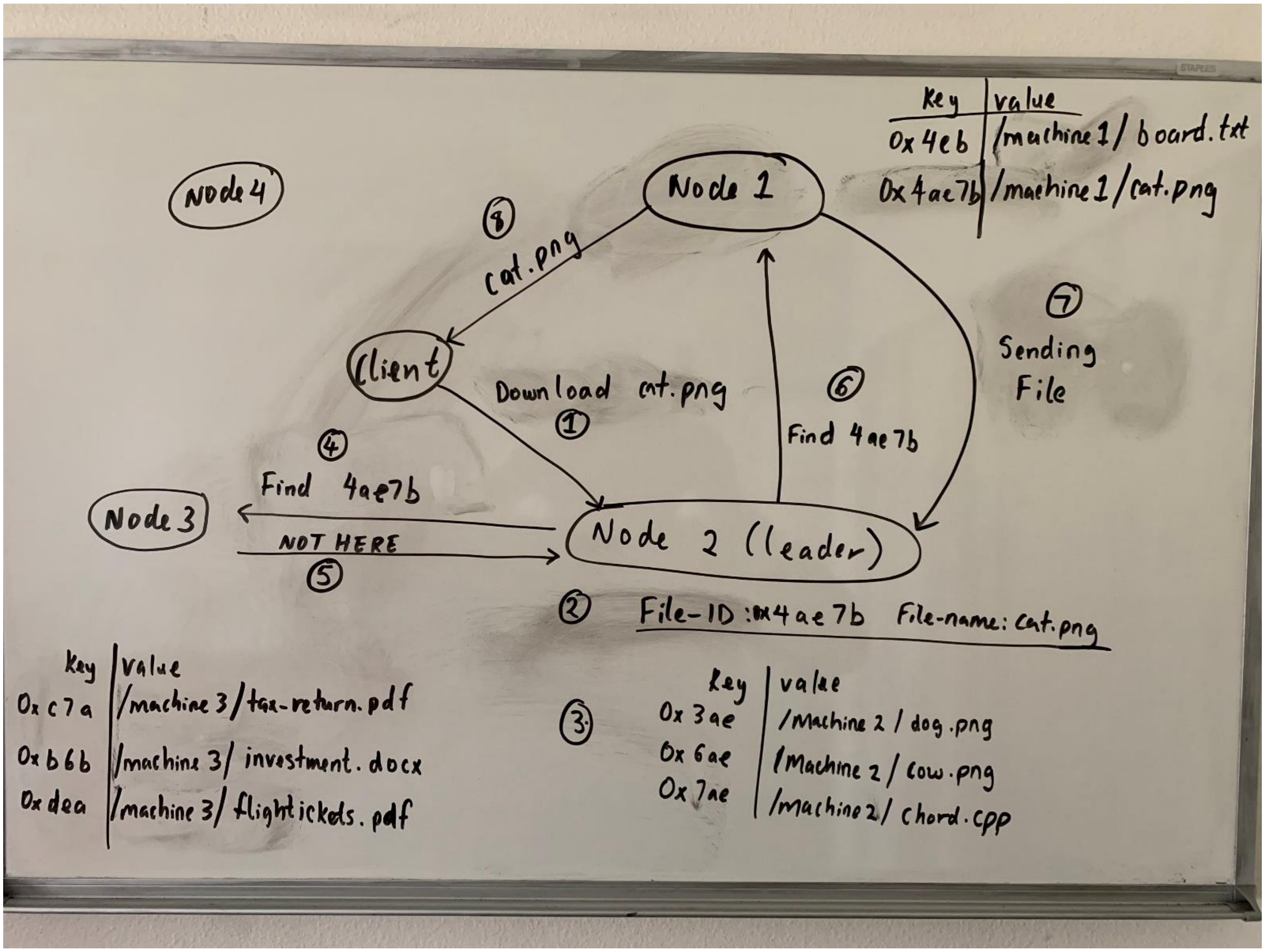
Listed below are the commands the client can request

- upload <file>
- delete <filename>
- download <filename>
- view-directory (this command will display all the files; the client has uploaded)

Below is an example of the path to download cat.png. See Image in Next Paper

1. Client Send Command to Leader
2. Leader Hashes a key-id (0x4ae7b), from the filename
3. Leader Translation Dictionary does not have file with key
4. Leader Ask Node 3 if they can send file with (0x4ae7b)
5. Node 3 Tells Leader they do not have a file with that key
6. Leader Ask Node 1 if they can send file with (0x4ae7b)
7. Node 1 Tell Leader, they have the file that matches and will send to client
8. Node 1, send cat file data to Client Listening Port

Image



Key	value
0x4eb	/machine1/board.txt
0x4ae7b	/machine1/cat.png

② File-ID :0x4ae7b File-name: cat.png

Key	Value
0x c7a	/machine 3/tax-return.pdf
0x b6b	/machine 3/ investment.docx
0x dea	/machine 3/ flightickets.pdf

③

Key	value
0x 3ae	/Machine 2 / dog.png
0x 6ae	/Machine 2 / cow.png
0x 7ae	/machine2/ chord.cpp