CS 6650 - Final Project Report

1. Project Overview

Our application aims to be a CLI distributed storage system with a simple client interface like dropbox. Clients can connect by issuing commands through the command line. We found the project very interesting and insightful. It gave us an idea of how a distributed system is designed. It served as a process to brainstorm and figure out how to make different algorithms work together in what can be close to a real world application. It prompted us to think of scenarios of failure, take into consideration edge cases, and try to solve problems we ourselves have faced in real world applications.

The application uses Thrift for RPC communication between different components and uses the Java 8 version.

2. Technical Impressions

a. Features - Client side

i. ***Upload a file***
Client connects to the server and provides the local path and filename of the file to be uploaded.

ii. ***Retrieve a file***
Client enters the name of the file that will be retrieved from the database and is created on the user's local file system. The user will get a message if the file is not available with the database.

iii. ***Delete a file***
The user enters the name of the file and gets a confirmation or an information message regarding deletion of file.

iv. ***Update a file***
The user enters the local path and filename of the file which is updated on the server.

b. **Features - Server side**

i. ***Load Balancing***
When a new file is uploaded, the metadata is checked to find the chunk server with the least load and adds the new file there to decrease the burden on each chunk server.

ii. ***Replicated Data Management***
The database is deployed on AWS, with replicated nodes. The replication is managed by AWS itself so that consistency can be maintained between the three nodes.

iii. ***Fault Tolerance***
If a node is down, replicated nodes take over until the node is back up. AWS has an automated system to get a node backup incase of failures.

iv. ***Distributed Mutual Exclusion***
We implemented this via transactions on MongoDB. If a document in MongoDB is being updated by one transaction, other transactions will have to wait and synchronize.

v. ***Distributed Consensus***
MongoDB allows modification/retrieval data only when there is a majority consensus, that means out of 3 nodes at least 2 nodes to be in agreement. The Raft protocol is used for taking consensus between servers for data update as per MongoDB documentation.

vi. ***Port-Discovery Service***
This centralized service is responsible for registering the chunk servers as well as replicas of chunk servers with the master servers. Since it is not extremely critical for the operation of the system, it has not been replicated.

vii. ***Chunk Server Replication***
The system allows multiple chunk servers to be replicated. It allows the system to be able to maintain access to parts of the data even when the main chunk holding said data is down. The use of replica nodes is handled by the master servers and is dependent on whether the replica chunks registered themselves using the port discovery service.

c. Assumptions and Design Choices

i. Use of MongoDB Atlas hosted on AWS: The use of a noSQL database allows unstructured data to be stored. It is not necessary to conform to specific data types while the data is being updated. Also, since it is hosted on AWS, it provides us with 3 replica nodes in the free tier.
It provides a single access point that can be used by the application instead of contacting each replica separately.

ii. File Size: Due to constraints in free tier of MongoDB, the file size should not be more than 16MB

iii. Centralized service - Port Discovery Service: We added this service for the convenience of starting new chunk servers on the fly without needing to provide its port number to a master server via cmd line. This is a promisingly extendable feature since we could have a server/chunk factory in the future that utilizes this API.

d. Abandoned Features

i. Paxos: Paxos was to be used to maintain the metadata of the master server after securing consensus it will then call the corresponding chunkserver based on the metadata. The algorithm implementation works end to end and can be found on the paxos branch, however due to time constraints we didn't proceed with it further.

ii. Port Discovery service replication: Due to a lack of time towards the project's end cycle, we decided to keep the port discovery service centralized.
iii. Login service with file mappings to account and share functions: we were initially planning to implement an account login service where users can store their files securely on the distributed file system. This feature fell from our rolling log due to its low priority and the implementation difficulties encountered with Paxos.

e. Future Improvements

- GUI for better usability.
- Support for bigger file size.
- AWS database deployment can be configurable related to the maximum number of requests that can be served at a time improving the scalability as well
- User authentication can also be done to restrict data access
- The system can be hosted on a cloud platform