LionDB: A Distributed System

Andrew Henry, Dalton Oxford, Foy Watford, and Jimmy Velasco

December 12, 2021

Introduction

Our team created a distributed database system named "LionDB". This database system can accept a file from a user in the system (or a created user), and stores that file within an array of data nodes with built-in fault tolerance.

Features

Feature	**Description**
Insert User Files	User can select a file to upload from there computer and have stored on a data server
Delete User Files	User can select from a list of files to delete
Retrieve User Files	A log is created specifically for every file belonging to the user
Multiple Nodes	Application can connect and send files to a variable number of nodes. System is fault-tolerant to node failures/disconnections
Temp User Files	Files that are sent from client to master are stored in a temporary state before sent to the various data nodes
Professional GUI	Graphical user interface that is professional and easy to use for users
Password Hashing	Application contains a hashing algorithm that was provided by Professor Mackey on Moodle

Technical Specifications

Concurrency

Our database system is able to accept a variable amount of clients at the same time as well as a variable number of data storage nodes to store the user's files.

Availability

The main point of failure within our system in the master server as there is only one. The system starts up upon the master server's "start.java" class. This starts up the connection listening to the data nodes and the clients.

Fault-tolerance

LionDB's main point of failure is with the master server. The systems only weakness with the data layers is the number of physical machines to turn into the data nodes as a variable number can be added into the system.

Data Replication

Files within the data nodes are copied into each new data server that is connected to the master server in order to allow every file to remain on all servers. This mean the servers are limited to only the amount of systems and the possibility of failure to each node.

Data Structures

Use	**Data Structure**
Communication between threads	Blocking Queue
User validation	Hash Map
Thread storage	Array List

Locking

Locking mechanisms are used for user log-in and master/data nodes connection.

Discussion

After thorough testing of this application, it was discovered that there is a bug within the log files on the master server. This error does not affect functionality. Whenever a data node fails, the error will log multiple times within the log file. This is due to how we log whenever a node fails. The thread will send the error information whenever it is first disconnected, then when the server checks the nodes for failure, it will log itself again. After a team discussion, the only way around this would be to alter significant ways we coded the connection between the nodes and the master server. Just like in a business environment, logging issues like this that have no real impact and would take significant man-hours to adjust the core of the application sometimes does not have enough significance to spend ample time on.

Communication

Our team communicated primarily using a dedicated discord server. We found this to be the most easily-accessible form of communication as we all had this application because of the CS Department server. We typically met during class time on discord along with one or two other times throughout the week.

We, to some degree, used a form of agile methodology to complete the project in pseudo-sprints. We worked first on a basic outline, discussed and assigned tasks, layed out the GUI and file management system, the communication between the client-master and master-nodes, and then tested connectivity throughout.

Members

At the beginning of this project, our original intent was the spread the work as evenly as possible. We believe, as much as is possible, this was achieved throughout the course of this assignment. Everyone did their fair share and completed accurately what was expected of them.

The following is a summary of the roles that each member played inside of this assignment, while also examining and being fluent with regards to the other group members' contributions. Also included is a snippet from that person's code for an example of their work.

Andrew Henry (master server, team lead)

Andrew is one of two people (The other being Foy) who is responsible for the master server architecture. This includes mainly the log edits and authentication classes, as well as assisting Jimmy and Foy with working on connecting the Nodes and Master Servers. Andrew is the one who created the Discord server for team communication, the Google spreadsheet of the overall design architecture, and the main author on this LaTeX document.

```
    private FileWriter write;
    private final String LOG = "log.txt";

    //node connect/fail, user attempt/fail, file failure
    public LogEdit(String type, String date, String time, String node_user,
        String IP_file) throws IOException {
        write = new FileWriter(new File(LOG));
        String line = "[" + type + "]" + "\t" +
                      date + "  " +
                      time + "," +
                      node_user + "," +
                      IP_file;
        write.write(line);
    }
```

Dalton Oxford (GUI, authentication, client-master)

Dalton is the main architect on the client server. He is responsible for the GUI, client user log-in, and the connection between the client and the master server. Dalton had the original design plan for authenticating users. Dalton also created the GitHub that was initially used to share classes before the Gitea server was used to add our final code.

```
    private void logIn_usernameTextFleidKeyPressed(java.awt.event.KeyEvent evt)
        {

        try {
            int key = evt.getKeyCode();
            if(key == KeyEvent.VK_ENTER){
                logIn_logInActionFunction();
            }else {
                return ;
            }

        }catch(Exception e) {
            e.printStackTrace();
        }
    }
```

Foy Watford (master server, data server assistant)

Foy is one of two people who is responsible for the master server. His main role included working on the connection between the Data servers (the nodes) and the master server. Foy helped greatly with the initial design for the GUI application that Dalton created. Foy created the design within the master server to spread the data between X nodes.

```
    byte[] b = new byte[1024 * 8];
    int len;
```

```
3
4       // variables from the input
5       String user = splitInput[1];
6       String filename = splitInput[2];
7       String action = "UploadFile";
8
9       // writing to the output
10      String out = action + "" + user + "" + filename;
11      output.writeUTF(out);
12
13      // while loop to send the file itself to the node
14      while ((len = input.read(b)) != -1) {
15            output.write(b, 0, len);
16            output.flush();
17      }
18      System.out.println("File: " + filename + " sent to node by: " + user);
```

Jimmy Velasco (data server, fault tolerance)

Jimmy is the main architect for the Data servers (the nodes) and is responsible for all functions within the various nodes. Jimmy is the main person who worked on the File input and output of the given files, as well as greatly assisted on the master servers' connection to the Data Server. Jimmy was also responsible for ensuring the database could function in the event of node faults.

```
1       void sendFile(String filepath, DataOutputStream output) throws IOException{
2           DataInputStream input = new DataInputStream(new FileInputStream(filepath
                ));
3           byte[] b = new byte[8*1024];
4           int len;
5           while((len = input.read(b))!= -1) {
6               output.write(b, 0, len);
7           }
8       }
```

1 Resources

[1] "File Transfer via Java Sockets", https://heptadecane.medium.com/file-transfer-via-java-sockets-e8d4f30703a5

[2] Hashing algorithm, Professor Mackey, https://mackey.cs.uafs.edu/courses/

[3] "LaTex Tables", https://www.overleaf.com/learn/latex/Tables

[4] "Distributed Systems 3rd Edition", Maarten van Steen, Various chapters