Design Document

The program's objective is to build a decentralized P2P file sharing system based on a single component as described in the manual.

Features:

- This system is built using Socket programming in Java language.
- To handle multiple requests I have used multithreading and exception handling features.
- To avoid data racing problem I have used thread synchronization.
- To my best efforts I have reduced code redundancy at places where I could.
- The system is compiled and tested for message passing up to 1024 Bytes and file sharing up to 1024 Megabytes.
- I have also implemented Data Resilience in the form of File Replication and Key/Value Replication.

I have created 3 files DHT.java, Client.java and Connect.java whose detail design is mentioned below:

DHT.java

This class is the main class of this system. It in turns calls other two classes mentioned above. Below are the functionalities of *DHT.java*.

- Instantiate HashMap<String, Socket> to save Socket Connections of all the other Servers
- Create a ServerSocket and start its own client (Client Thread)
- Read the config file and Connect with other Servers (Connect Thread)
- Accept connections from other server's client
- Provide Registry, Search and Obtain functionalities to incoming requests (Type2 Thread)

Client Thread (Client.java)

The Client thread will perform the following tasks:

1. Display a Menu to the user to register, search and obtain files from other servers and call the appropriate functions taking input from the user

2. Provides ***KeyPad (key)*** and ***ValPad (value)*** functions for key and value padding which in this assignment will be filename and server names.

3. Provides ***hashFunction (key)*** - This function will perform hashing on the key and will return a string array of size 2 containing names of destination server and replication server. Using these names we will get the Socket from the HashMap

4. Provides ***msgPass (String socketName, String choice, String value)*** – This function instantiate Server's DataInputStream and DataOutputStream with the Socket passed in the arguments. It then writes choice and value on the server. While reading from the server it filters based on the choice and displays the appropriate output.

5. Provides ***copyFiles (String filename, String destination)*** – This function takes care of file replication. It is called whenever a file is registered. After the file entry is replicated in the backup server, this function will copy the file from the origin server to the backup server.

6. Provides ***fileReceive (String choice, String filename, String serverName) –*** This function is used for downloading file from the server. It is called when user choose to obtain a file from other peers. It instantiate socket connection and downloads the file from the requested server passed in the arguments.

In all the above functions, to handle fault tolerance I have added logic in catch block of exception handling which will forward the user requests to replicated server. As per the logic, there are 2 replicated nodes, fourth and the eighth. For first 3 nodes and the last node, fourth server will act as backup server and for the rest eighth server will act as backup server.

Client Thread is written in a separate file named Client.java which implements Runnable interface and provides implementation of run () method.

Connect Thread (Connect.java)

The Connect thread

1. Go to sleep for 60sec until all other servers are up
2. Create a Socket Connection with the Server
3. Put the Socket in HashMap

Connect Thread is also written in a separate file name Connect.java which implements Runnable Interface and provides implementation of run () method.

Type2 Thread (Inner Class in DHT.java)

The Type2 thread will perform the following tasks:

1. Instantiate HashMap<String, String>
2. Instantiate DataInputStream, DataOutputStream for the client
3. Provide registry(filename, serverNames)
4. Provide search(file)
5. Provide Obtain (filename) - Counter part of ***obtain ()*** at Client side, it provides file sending functionality.

Thread Type2 implementation is written in a separate class named Type2 (inside class DHT) along with “Registry”, “Search” and “Obtain” functions.

Config.txt

It stores information about all the servers like name, IP address and port number. I have used comma as a delimiter. Each line read is stored in a string array and then passed as a String or used appropriately.

Program Flow:

1. In this program, on start-up I am providing the server name through command-line arguments.
2. When the program is started, its main function will be executed. It will first read the config file.
3. When reading the file, if it comes across his own name it will start his server, instantiate and start the Client Thread.
4. Else it will call the Connect Thread to connect with that Server and put the server name and Socket in HashMap.
5. The above STEPS 3 & 4 will follow until file is completely read.
6. Once all the Servers are connected it will then start accepting requests from clients.
7. After each request is accepted, it will instantiate and start a new thread of Type2 which will serve user queries.

Trade-offs:

- Since I have been using Sockets from last two assignments, it was kind of difficult to move to RMI now. But RMI is better choice to call server functions remotely from Client without extra overhead of read/write function calls of I/O.
- I trade-off more efficient hashing function to generate hash code which is independent of no. of servers with a simple hash function using mathematical mod operation.
- Also, since the system is more static all the peers need to wait until all others are up and running.

Extensions:

- As an extension for this system we can make the system more dynamic and scalable for more no. of servers.
- Currently the system supports only text and binary files, we can extend this system to share multimedia files using different file sharing protocols.
- Also, the system works on Command Prompt which can be less user friendly so we can migrate the system to a user friendly GUI using Java Frames or Applets.
- To increase downloading performance we can use threads to implement parallely in case of larger no. of operations.
- As I said in the trade-offs, we can built a more efficient hash function independent of no. of servers.
- For better performance evaluation, we can use cloud services such as amazon web services.