Advanced Operating Systems

CS550

Programming Assignment 3

Simple Decentralized Peer to Peer File sharing System

Student

Nikhil Nere

A20354957

nnere@hawk.iit.edu

Table of Contents

1. Introduction

1.1. Purpose

This document provides a detailed design of the Simple decentralized peer to peer file sharing system. It provides an overview how different components in the system interact with each other.

1.2. Problem Statement

A distributed file sharing is to be implemented which has multiple indexing servers to register the files from peers. The peers in the network can register and download files on the distributed hash table.

Decentralized Indexing Server: The decentralized indexing server will maintain an index of the files registers by peers. There will be multiple servers who will store the file information in a local hash map. When a peer will make a request for file search it will return the list of peers who have register that file on the server including the replica servers. The peer calculates hash to decide to which server to go to register the file. If any of the servers goes down then the replica will provide the data hence there will not be any loss of data.

Client: A peer will register the files on the indexing server to make them available for other peers. A peer will make a file search request to the indexing server and will get peerId of the peers having that file. It will then connect to one of the peers to get the file.

All the peers will be listening for any file download request. If a client registers a file then it keeps a file on a particular directory and the file sender part of the client will pick the file from that directory and sends when a file download requests comes.

It's a two level hashing. First at the client side to identify which server to go to this is done by calculating the hash value. Second at the sever side to store the key-value pair in the hash table.

2. Design Overview

2.1. Topology Diagram

The system has multiple servers and clients

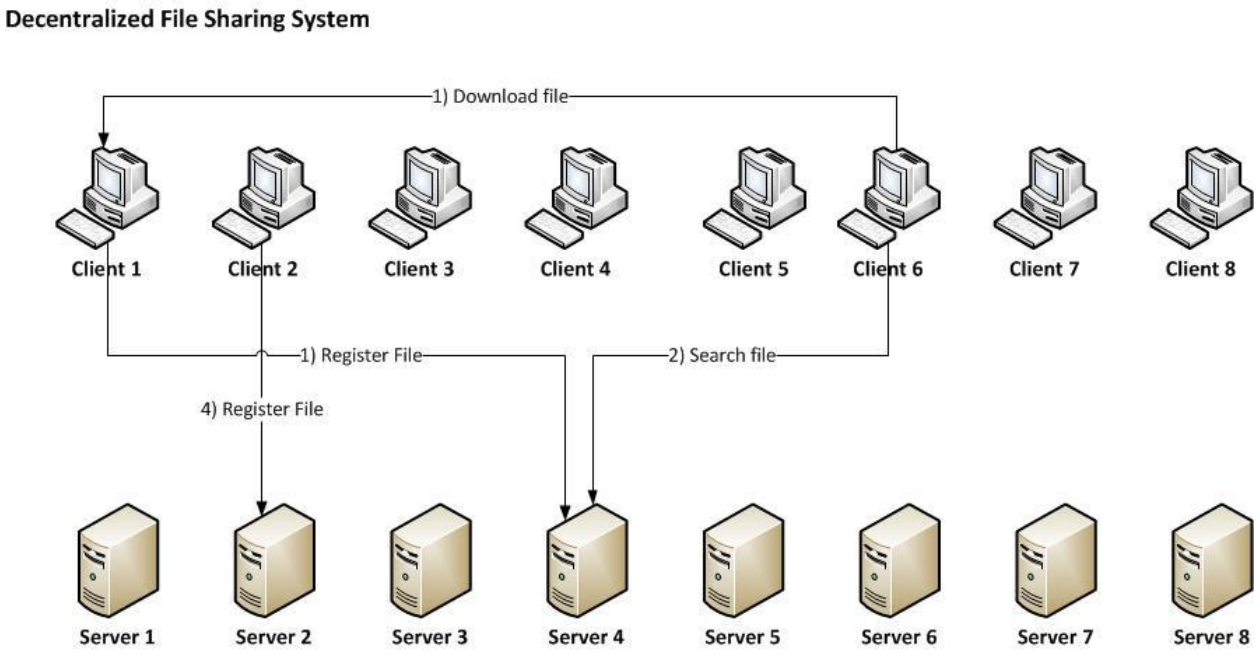

Figure-1: Decentralized File Sharing System

2.2. Detailed Design of the System

A peer in system acts as one of the indexing servers and also a part of it is listening continuously for file download requests.

- The system has multiple servers which will store data about the files registered by the peers in a hash table. A client in the system can register and search files on the servers.
- At the client side a hash function will decide on which server the file should be registered. Once the file is registered on one of the servers, it will be available for all the clients in the system to download.
- Any client in the system can send a search request with the filename and will get the list of peers having that file. For a search request same hash function will used to identify which server will have the file information.

Below is detailed communication between different components. Please refer to **Figure-1**.

1) Client 1 sends a register request to server 4. The decision of connecting to server 4 is taken using the hash function.
   On receiving register request server 4 will store the file information in a hash table and will return true. Now any client can search this file.
2) Client 6 sends a search request with the file name that was stored by client 1. Again the hash function will decide to which server to go to.
   Server 4 will receive the search request and will return the associated peer list from the hash table.
3) Client 6 then connects to client 1 to download the file. It gets client1's information as a result of the file search request.
4) Client 2 sends a register request with new file name. This time the hash function decides to go to server2.
   Server2 will receive the register request and will store the file information in the hash table.
   Any client in the system can now search and download the file registered by client2.

This is how the components in the system interact with each other.

All the servers in the system are always listening for the requests from clients and they can distinguish between register and search requests. Any client can register a file on the servers making it available for other clients to search and download

2.3. Replication

When a peer registers the file then based on the replication factor the file is registered on the replicas as well. The replica server is one of the indexing servers. When a replication request comes to any server it registers the file on its local hash table and comes a download request to download that file. So it keeps replica of the file making it available to other clients when the primary owner is down.

2.4. Features

- System supports file size larger than 4 GB
- System supports text as well as binary files
- This System supports Data Resilience, Fault tolerance
- Uses replication factor to control the replication
- System is implemented using Sockets and multithreading

3. Implementation

3.1. Server Component

- All the servers handle the clients independently by creating an independent thread for each client.

```
while (true){
    Socket client = indexServerSocket.accept();

    Thread hashTable = new Thread(new ServerHashTable(client, fileDir));
    hashTable.start();
}
```

- The server identifies the type of request if it is a register, search or replication request

```
if ("0".equals(reqType)){
    outToClient.writeObject(put(request));
    outToClient.flush();
}else if ("1".equals(reqType)){
    outToClient.writeObject(get(request));
    outToClient.flush();
}else if ("2".equals(reqType)){
    outToClient.writeObject(delete(request));
    outToClient.flush();
}else if ("3".equals(reqType)){
    outToClient.writeObject(replicateReq(request));
    outToClient.flush();
}
```

- The register (put) and search (get) perform the Map operations.

```
private boolean put(String request) {
    int index = request.indexOf('|');
    String key = request.substring(0, index);
    String value = request.substring(index+1, request.length());
    String peerList;
    if (distHashTable.containsKey(key)){
        peerList = distHashTable.get(key);
        String peerStr[] = value.split("#");
        peerList = peerList + "#" + peerStr[0]; //new request without replica
        distHashTable.put(key, peerList);
    }else{
        distHashTable.put(key, value);
    }
    return true;
}

private String get(String key) {
    String peerList = distHashTable.get(key);
    return peerList;
}
```

- Replication request on the server

```
if (distHashTable.containsKey(key)){
    System.out.println("File already replicated");

    String peerStr[] = value.split("#");
    String newRequest = key + "|" + peerStr[0]; // new request without replica addr
    put(newRequest);

}else{
    put(request);
    //make a replica of the file i.e. download it form the source of the file
    String peerStr[] = value.split("#");

    //fileDir is the file share folder. File will be downloaded in the share so tha
    //Note : here we are not downloading the file in downloads folder

    Thread replicator = new Thread(new FileReplicator(peerStr[0], key, fileDir));
    replicator.start();
}
```

3.2. Client Component

Client component creates connections with all the servers at start up and keeps the connections open for better performance.

- **Opening connections**: Connection is established with all the servers at the beginning and they are never closed.

```
for (int i = 0; i < numOfServers; i++){
    try {
        sockets.add(new Socket(serverIPs.get(i), serverPorts.get(i)));
        objOutStreams.add(new ObjectOutputStream(sockets.get(i).getOutputStream()));
        objInStreams.add(new ObjectInputStream(sockets.get(i).getInputStream()));
```

- **Hash Function:** Hash function is applied to the key. The hash calculated is the server number.

```
private int getHash(String key) {
    char ch[];
    ch = key.toCharArray();

    int sum = 0;

    for (int i = 0; i < key.length(); i++){
        sum += ch[i];
    }
    return sum % numOfServers;
}
```

- **Register file:** With file registration request file replication request is sent to the replicas.

```
public boolean put (String key, String value){
    int server = getHash(key);
    boolean res = false;
    int index = 0;

    index = server;
    for (int i = 0; i < replicationFactor; i++){
        index = (index + 1) % numOfServers;
        value = value + "#" + serverIPs.get(index) + "|" + downloaderPorts.get(index);
    }
    String req = "0|"+ "|"+key +"|"+ value;
    String replicationReq = "3|"+ "|"+key +"|"+ value;
    res = (boolean)sendRequest(req, server);

    //Replicate on replicas
    for (int i = 0; i < replicationFactor; i++){
        server = (server + 1) % numOfServers;
        sendRequest(replicationReq, server);
    }
    return res;
}
```

- **File Search:** If file search request fails then search request is sent to the replicas one by one until a valid response is returned.

```
try {
    //send request to the server
    objOutStreams.get(server).writeObject(req);
    objOutStreams.get(server).flush();

    //get response from server
    res = objInStreams.get(server).readObject();
    replicaCount = 0;

} catch (IOException e) {
    System.out.println("Server down!! calling replica");

    server = (server + 1) % numOfServers;
    replicaCount++;

    if (replicaCount<=replicationFactor){
        return sendRequest(req, server);
    }else{
        System.out.println("Primary server and all the replicas are down!!");
    }
```

- **File Server:** This is the peer acting as a file server. It accepts the file downloading requests from other peers. The file server can handle the multiple download requests simultaneously by creating an independent thread per peer.

```
while (true){
    Socket client = fileServerSocket.accept();
    Thread clientThread = new Thread(new FileSender(client, fileDir));
    clientThread.start();
}
```

4. **Future Enhancements**
    - A sophisticated hash function can be implemented to ensure the keys are evenly distributed among all the servers.
    - Currently the peer has to send the exact file name to the indexing server. The enhancement can be done in future to apply sophisticated search algorithms to find the partial matches.
    - In further enhancements the peer can be able to fetch the list of available files on the indexing server based on an alphabet or a partial match.
    - Currently peer can only register the file. In future enhancements the peer can send a request to delete the entries of the files from the index which it has previously registered
    - System can be made scalable.

5. Assumptions and Constraints

- Before running the client the port should be free on which the server will be listening. Or make sure a free port is given in the properties file.
- If running multiple servers on a single machine then give different port numbers for each instance of the server
- All the servers should be up and running at the beginning.
- Peer has to keep the files which it is sharing under one folder and the folder path should be updated in the config.properties file
- Peer should type exact name of the file for searching else the server will return message saying “File not found”.
- If a peer has registered a file on the server it has to keep listening on the port it has given in the register request. If it is down then the file will be available on the replica
- The size of the file that can be transferred is several Gbs. For the assignment the ask was to support file size of 4 Gb. The system is supporting the 4Gb file size.