CS550 Advanced Operating Systems
Programming Assignment 3
<u>Design</u>

submitted by:
Chiranjeevi Ankamredy
A20359837

The assignment is on implementing Decentralized Indexing Server (DIS) and performing Register and search operations on the Indexing server, Download the file from Peer Server and Multiple clients accessing the DIS..

We Assume that servers and network is static. There are 8 Clients, each client will act as client and server, and 8 Indexing servers .Each Indexing 0server has its own internal Hash table which stores file names as key and File information of Strings. Each of the 8 peers does the following operations on Indexing Server:

1. Register(File)
2. Search(File)

When Client gets file information details from Indexing server. Now Peer client connects to Peer server to do following Operation.

3. Download(File)

Each peer hashes the key and then invokes a particular operation on the server that represents the hashed value.

To provide handling of concurrent requests, servers are implemented in multi-threading in java.

The above scenario is implemented in JAVA using the concept of sockets, Multithreading for communication.

<u>Functionalities:</u>

1.IndexingServer

Here, we have a config file containing IP Address and Port of 8 peers . When Indexing server is initialized, it contains a server port which is given as a Static The Server receives a message from peers to do either register or Search operations on its hash table and does the appropriate operation on index server. It is multi-threaded to take up multiple peers requests and perform all the two operations.

2. PeerServer

Here, we have a config file containing IP Address and Port of 8 peers since each peer acting as a client and server. When a server is initialized, it contains a server port which is given as a Static The Server receives a message from peer clients to do upload(Replica file) or Download file operations and does the appropriate operation on server. It is multi-threaded to take up multiple peers requests and perform all the two operations. The Server also ensures resilience of files ,uploaded from neighbor nodes.

3. PeerClient

The client also reads the config file and connects to all the servers during initialization, and maintains the connections until the user chooses to exit. Here, user will give Hash key and connetct to the Server. This reduces the overhead of having to connect every time the client invokes an operation on one of the 8 servers. The peers acting as clients do the following operations

<u>a.Hash(key):</u> The hash function returns a randomized hash value for any key. This (hash_value)%8 gives a value 0 to 7 , each value representing one of the 8 servers given in Config File. We refer to this value as node.

1. **<u>Register(File,Fileinfo):</u>** The Client doing Register(File,Fileinfo) hashes the key and sends a message that invokes the 'put(key, value)' operation on the indexing server that represents the node. It returns 'Sucess' on successful REGISTER operation on the Server and 'Failed' if the Registration is failed.

2. **<u>Search(File):</u>** The Client doing Search(File) hashes the key and sends a message that invokes the 'get(key)' operation on the server that represents the node. If the key exits at the server, then the client receives the File information, else it receives the message "Invalid Key".

3. **<u>Download(File):</u>** The Client doing select the File information from search results to Download(File) and sends a message that invokes the 'Download` message to the Peer server that represents the node. It returns 'Downloaded' on successful Downloading the File.

MESSAGE_FORMAT:

Client and Indexing server interact through messages for Each operation. To invoke an operation client sends a message to indexing server. The message is a string containing information about what operation to invoke at the server. The message that Client sends to the indexing server is a string of the format " KEY {VALUE}". The message contains value if it is a register operation.

The message size varies with the size of key. VALUE is a string of 1000 bytes.

OPERATION_ID can be 1, 2 or 3.

1 for Register (Filename, Fileinfo) : "c1 KEY VALUE" (if it is to invoke put (KEY, VALUE) operation on indexing Server)

2 for Search(File name) : " 2 KEY" (when invoking get (KEY) operation on indexing Server)

3 for Download (File) : Sends the Message" Download" to the peer Server.

The Server receives the message and interprets it and performs the operation based on OPERATION_ID.

After sending the message to peer server, if the Peer Client interprets the message as Register operation, it forwards the message to its neighbor node for replicating the File, so the neighboring node knows it is from the Peer and it stores the replica of File and register in its hash table.

Multi-Threading and Concurrency: Servers are expected to handle operations request from multiple clients at a time. So, to ensure concurrency on the server side multi-threading is used. The server creates a new thread for each client connection. If server fails to do Client Operations ,It will Raise the exceptions. And Peer server are to handle upload and Download operations from multiple Clients at a time. So, to ensure concurrency on the server side multi-threading is used. The server creates a new thread for each client connection.

The communication is synchronized between the clients and servers. The client invokes a certain operation and waits for the server to respond to the message.

Improvements and Extensions to the program:

Multi-threading at peer server and Indexing Server is implemented by generating a new thread for a client connection, since there are only 8 peers to considered, thread generation doesn't seem to add that much overhead, but when there are large number of peers communicating and Thread pooling is often more convincing approach to handle multi-threading, to reduce the significant overhead of thread creation for each client connection, Here, we haven't create locks on process because since we have only 8 servers, that can be done by creating a pool of threads using **new CachedThreadPool** method for creating threads and also Event driven programming can give a better performance at a larger scale of peers.

To ensure further data resilience, the replication can be done across several nodes rather than a single next node.