Design Document

Communication between Server and Client:

The communication between the peers and the server happens using Java Socket API. We use the technique of **ObjectInputStream** and **ObjectOutputStream** of the Socket API for communication amongst the nodes. Both the techniques are capable of transferring information using java objects only.

We'll go through the design of the system by referring each module.

1. **Network Configuration file**
   - The network configuration file provides the information of the peers in the network. It provides the program with the IP address of the peers which are there in the File Transfer System network. The name of the network configuration file must be "network.config".
   - The key NODES contains comma separated IP address of the peers in the network.
   - The key FILES_LOCATION contains the location (path) where the files are stored which will be shared by the peer.
   - If you want to use the replication feature, then define the IP addresses of the nodes who will be storing the replication data in the REPLICATION_NODES parameter in the configuration file. Separate each IP address by comma (**,**).
   - Also, mention the location where the replicated files will be stored in the REPLICA_LOCATION parameter in the configuration file.

2. **Hash function**
   - The hash function takes as input Key which is the name of the file registered by the peer and returns an integer value which denotes the ID of the node where the filename along with the peer IP address as value will be stored and accessed for retrieval/deletion.
   - So, the steps to find the node ID from the input key is –
     a) Perform hashCode of that Key. This is done by using Java's hashCode() method of Object class. The hashCode is a large integer value and it is always same for a given string. Two strings will not have same hashCode.
     b) After we get the integer hashCode, we divide it by total number of nodes present in the network (Total no. nodes can be retrieved from configuration file). In short, we perform MOD of hashCode and nodeCount.
     c) The result of this MOD operation is returned by this hash function to the calling function.

3. **Request**
   - Request class is used whenever a node wants to request something from another node/server. This class has two fields - **requestType** and **requestData**.
   - Whenever a node/server wants to request something, it creates a Request object, sets it **requestType** which is basically a command like REGISTER, UNREGISTER, LOOKUP, etc. and sets the **requestData** to appropriate data it wants to send while requesting something and sends the object using **ObjectOutputStream**.
   - A node tracks incoming Request object using **ObjectInputStream**.
   - This class provides setter/getter methods to set/get the required field.

4. **Response**

- Response class is used whenever a node wants to send response for the request received. This class has two fields – **responseData** and **responseCode**.
- Whenever a node/server wants to send a response, it creates a Response object, sets it **responseCode** (For example, 200 if request was processed successfully) and **responseData** to appropriate data as requested and sends the object using **ObjectOutputStream**.
- A node tracks incoming Request object using **ObjectInputStream**.
- This class provides setter/getter methods to set/get the required field.

5. **IPAddressValidator**
   - This class is used for validating an IP address. It uses regular expression and pattern matching to check whether the provided IP address in the configuration file is valid.

6. **NetworkUtility**
   - This class provides a method – getLocalAddress() which gives the IP Address of the Peer. This is used to check whether the selected peer (through hash function) for <filename, peer_address> pair insertion is itself or not. In case it is itself, then we don't need to create a socket connection.

7. **FileTransferSystem**
   - FileTransferSystem is the main class of the system which runs when the program is started using "make run" command.
   - This class loads the configuration file. It retrieves the IP address of all the peers which will be in network and also IP address of the peers who will be acting as replication servers. A normal peer can also act as replication peer if its IP address is mentioned in "REPLICATION_NODES" field.
   - After loading the configuration file, it starts an instance of peer client on a single separate thread and as a peer server listens to port 20000. Any peer client who wants to connect to this peer can connect using the peer's IP address and Port 20000. I have used port 20000 for my De-centralized File Transfer System.
   - This class also defines the most important data structures of this system like **hashTable**, **replicatedHashTable**, **networkMap** and **replicationNodes**.
   - **hashTable** is a concurrent hash map (Java's **ConcurrentHashMap**) which stores all the <Key, Value> i.e. <filename, peer_address> pair of its own peer. I have concurrent hash map because it is thread safe i.e. multiple threads can perform read/write operations on the **hashTable** without any kind of deadlock or inconsistency.
   - **replicatedHashTable** is also a concurrent hash map data structure which stores the **hashTable** i.e. <filename, peer_address> pairs of all the peers in the network. It maps the **hashTable** of a peer by the peer's IP address. This data structure is also thread safe.
   - **networkMap** is a hash map structure which stores the Peer ID and IP Address of all the peers in the network. I haven't used concurrent hash map in this case because multiple threads will only read this data and not modify it.
   - **replicationNodes** is a list which stores the IP Address of all the peers who are responsible for storing the **hashTable** of all the peers in the network.

8. **PeerClient**
   - The Peer Client provides following functionalities – REGISTER file, SEARCH file, UNREGISTER file, PRINT PEER SERVER LOG and EXIT. The Peer Client requires the Host Address (IP address) of the

Peer (i.e. other Peer's server) it wants to connect so. It can connect to other peer using a socket connection using that host address and port 20000.

- A client can register a file it wants to share in the network. The peer client asks user for the name of the file it want to register. It checks if the file is present in the location specified in the configuration file and if present, it connects to the appropriate peer by hashing the filename and sends a request to that peer to store the <filename, peer_address> pair in its **hashTable**. The hash function is explained later in this document. The peer client calls **put(key, value)** function to register the file in the network for sharing. The put(key, value) function returns true if the file is successfully registered else returns false.
- A client can search for a file in the network if shared for download. The peer client asks user for the name of the file with extension (Example: data.txt), then performs hash function on the filename and connects to the appropriate peer to request the IP address of the peer which contains the file. The peer client calls **get(key)** function to get the peer IP address who is actually having the file. The get(key) function returns the peer IP address if the file is present in the network for sharing else it returns null (nothing).
- This system works on all kind of files. I have tested this system on .txt, .jpg, .docx, .zip, .pdf files.
- If the peer client is unable to connect to the respective peer (may be the peer is down), then it requests the replication nodes to get the peer IP address who is storing the file is present in the network for sharing. It calls **searchReplica(key)** in this case.
- A user can remove a file from the file sharing system i.e. unregister a file because it no longer wants other peers to download the file. The peer client asks user for name of the file with extension, performs hash function on the name of the file and connects to the appropriate peer to place a request to unregister the input filename. The peer client calls **delete(key)** function to place the unregister file request i.e. removing the <filename, peer_address> pair. The delete(key) function returns the true if the file was successfully unregistered i.e. its entry was deleted from the appropriate peer else returns false. In case the file wasn't registered for file sharing, then too it returns true.
- The peer client on initialization (on startup) retrieves its **hashTable** (if any) from the replication nodes by calling **retrieveHashTable().** Since I have implemented replication mechanism, this is necessary because if the peer shuts down for some reason and starts again, it should have its own data. If this was not implemented then, if the peer starts after shutting down, then it won't have its data while the replication nodes will have peer's data thereby creating inconsistency between the peer's hash table and replication hash table.
- The peer client on initialization (on startup) also retrieves its replication hash table from other replication nodes only if it is amongst the replication nodes in the network. If it is the only replication node in the network, then it requests all the peers to send their hash table data to the replication node. It calls **retrieveReplicationHashTable**() for this purpose.

9. PeerServer

- The Peer Server listens to the requests by other Peer Clients in the network. It serves requests like REGISTER, SEARCH, UNREGISTER, etc.
- If the request is **REGISTER** i.e. to register a file in the network for file sharing, then first it checks its **hashTable** if that filename (key) already exists. If the filename exists it sends a **Response** object with **responseCode** = 300 which means that a file with the input filename is already registered. If the peer client decides to overwrite this then it sends a **Request** object with **requestType** =

"REGISTER_FORCE". In this case, the peer server directly adds the filename along with the peer address directly to its hashtable. If present, it is overwritten. The old peer IP address for that filename is replaced by the new peer IP address. If the file is successfully registered then the peer server sends a **Response** object with **requestType** = 200.

- If the request is **LOOKUP** i.e. to search and send the peer IP address of the node who has the requested file, then the peer server checks its **hashTable** and sends a **Response** object with **requestType** = 200 and **requestData** = Value (Peer IP address) if the Value for the requested Key (filename) exists else it sends a **Response** object with **requestType** = 404 meaning file not found i.e. file not registered in the network.
- If the request is **UNREGISTER** i.e. to unregister the file from the network so that none of the peers can download the file, then the peer server deletes the <Key, Value> i.e. <filename, peer_address> pair of that filename (Key) from its **hashTable** and sends a **Response** object with **requestType** = 200.
- If it receive a **DOWNLOAD** request, then it sends the requested file from its FILES_LOCATION as defined in the configuration file.
- The requests **R_REGISTER**, **R_LOOKUP** and **R_UNREGISTER** is handled same way as requests REGISTER, LOOKUP and UNREGISTER respectively except the operations are done on **replicatedHashTable** rather than **hashTable**.
- In the **R_REGISTER** request, the peer server also requests the files from the peer who has the file so that it can store a replica of that file.
- If request = **GET_HASHTABLE**, then it sends its own **hashTable**. For response, it sends a **Response** object with **responseType** = 200 and **responseData** = **hashTable**. In case, there is no hash table for the requestor peer, then **responseData** = null.
- If request = **GET_REPLICA**, then it sends the **replicatedHashTable** as is to the peer client who has made the request. For response, it sends a **Response** object with **responseType** = 200 and **responseData** = **replicatedHashTable**.

10. ReplicationService

- The ReplicationService class provides data replication services used by the Peer Server.
- If it receives a REGISTER <filename, peer_address> request, then it sends a R_REGISTER <filename, peer_address> request to all the peers contained in the **replicationNodes**. So, it basically tells all the replication nodes to store a replica of this <filename, peer_address> pair.
- If it receives a UNREGISTER <filename> request, then it sends a R_UNREGISTER <filename> request to all the peers contained in the **replicationNodes**. So, it basically tells all the replication nodes to delete the <filename, peer_address> pair with the specified filename.
- If it receives a REPLICATE request, then it retrieves the hash table i.e. data from all the other peers and also retrieves all the files which have been registered by all the peers in the network.

11. Data Replication

- You might have got a basic idea about data (hash table) replication implemented by this DHT system.
- When the Peer Server receives a REGISTER <filename, peer_address> request, after serving the request it calls the **ReplicationService** on a different thread with REGISTER <filename, peer_address> request. The **ReplicationService** sends a R_REGISTER <filename, peer_address>

request to all the replication nodes (found through **replicationNodes** in configuration file). The Peer Server of all the replication nodes serves the R_REGISTER <filename, peer_address> request by adding the <filename, peer_address> pair in its **replicatedHashTable** structure. In this way, whenever a peer in the network registers a file in the network for sharing, it is also stored in all the replication nodes.

- When the Peer Server receives a UNREGISTER <filename> request, after serving the request it calls the **ReplicationService** on a different thread with UNREGISTER <filename> request. The **ReplicationService** sends a R_UNREGISTER <filename> request to all the replication nodes (found through **replicationNodes** in configuration file). The Peer Server of all the replication nodes serves the R_UNREGISTER <filename> request by removing/deleting the <filename, peer_address> pair having the specified filename from its **replicatedHashTable** structure. In this way, whenever a peer in the network unregisters a file from the network, its record is also deleted from all the replication nodes.
- As mentioned earlier, if any peer client in the network is not able to connect to another peer for LOOKUP or DOWNLOAD operation, then it calls searchReplica(key) which in turn sends a R_LOOKUP <filename> request to all the peers in **replicationNodes**. In this scenario, the Peer Servers, serve the R_LOOKUP <filename> request by retrieving the peer IP address of the node having the requested file from its **replicatedHashTable**. In this way, the location of the file is retrieved from the replication nodes in case the original nodes are not active. Similarly, if the peer is not active, then its files are served by the replication nodes.
- Since this system does not load the peers in the network dynamically (i.e. a peer isn't added after all the peers in the configuration file are initialized), if a node decides to act as replication node after it is initialized, it won't be able to do so. There is one way to do this – the node (peer) will restart itself after adding itself in configuration file. This will work because, after starting, the peer will contact other replication nodes to get its data and also to retrieve the replication data. But, again this won't work in case this is the first replication peer which is made the replication node after starting the program. In this case, it will lose its data.

12. LogUtility

- This class provides functionality of creating and modifying log files which is used by the Peer Server.

<u>Suggested improvements in the current system</u>:

- The code of the programs can be improved so as to work efficiently. In a network of 5-10 nodes, the difference may not be visible but in a distributed network of more than 10 nodes the difference may be important.
- A better Hash Table structure can be implemented for faster operations.
- Data fields in the Request and Response class can be increased so that multiple kind of data can be sent in a single request/response.
- This system doesn't do any kind of error checking at application level after the file transfer is done. The transferred file may become corrupt under certain conditions. (The files transfer is perfect in the current system. Just that it's better to add an extra error checking mechanism.)

  This can be done using checksum mechanism. Just before sending the file, the node will send a Response object containing details about file including the checksum. After the file transfer is complete, the peer can compare the received checksum with the downloaded file's checksum. If they are same, then file was transferred correctly else not.

- No authentication is done when the peer connects to the server in this system. Authentication mechanism so that the Indexing Server is able to authenticate the peer and accept only authorized connections. Also, while serving the file transfer request, the peer should check if the requestor (peer) is authorized and authenticated with the Indexing Server.

  This can be done using simple user id and password mechanism. The peer will need to have a User ID and Password to connect to the Indexing Server. For better security, instead of passing raw passwords trough the network, the peer will calculate hash of the password and send the hash value (i.e. use SHA algorithm). Also, the indexing server will maintain a list of connected and authenticated users. So, before initiating file transfer request, the peer server will check whether the peer is connected and authenticated with the Indexing Server. If yes then transfer file else don't.
- The code of the programs can be improved so as to work efficiently. In a network of 10-20 nodes, the difference may not be visible but in a distributed network of more than 100 nodes the difference may be important.
- Data replication can be made more efficient by using various parameters like peer's network speed, reliability, how often the peer remains connected/disconnected. These parameters can be used and algorithm can be devised which selects a better replication node amongst the connected peers.