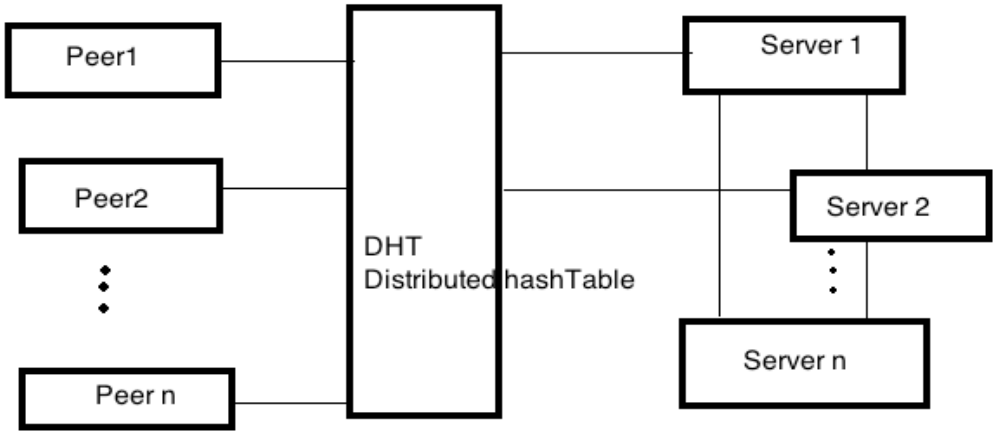
Design Document for Programming Assignment#3

Decentralized Index Server

Decentralized Index Server act as a server as well as broker for another Index Server to maintain a decentralized registry of connected peer information. Each Index Server keeps its connected peer information like peername, peerfiles list name, replication server etc. in a registry object.

Architecture Block Diagram



Maintaining Registry in Distributed Index Server

As we have multiple Index server and every index Server have its own set of peers and any peer can search for files that are stored in peers connected to other index server. To solve this problem each Index server is connected to each other through TCP socket and broadcast its list of connected or registered peer to other Index server through serialized object whenever a new peer connection is made to that index server. When any index server listen the broadcast it compares its registry with the broadcasted message and if it find any new peer listening server silently updates its registry. Using this approach we able to maintain a registry same as centralized index server though there is network latency involved but we are able to gain scalability.

Replication

Another problem is replicated file of peer to another peer, so that if primary peer fails or get disconnected other peer should be able to get data from the replicated peer server. To solve this problem we have created a property in configuration file of peer "replia.no" where any peer should specify the no of replication peer server it needs for replicating data. On registry of peer to any index server peer send no of replication needed information for Index server. Once the registration peer command comes to Index server it looks for no of replication property sent by

Peer and perform following action: If the there are no Peer conneted to index server then it does not create the replica at that time just send the broadcast message to other connected index server. Replica creation request for this Peer will be sent to other peer when any new peer is registered or when server listen to new registry broadcast message. If Index server have connected peer than it will directly send message to other peer to create replica of newly connected peer

Code for Hash Function

```
public int hash(String key) {
          int hashvalue = 0;
          if (key != null || !key.isEmpty()) {
               for (int i = 0; i < key.length(); i++) {
                    hashvalue = hashvalue + ((int)
key.charAt(i));
               }
               hashvalue = hashvalue % noOfNodes;
          }
          return hashvalue;
     }
```

In above code, we are calculating the ASCII value for each bytes and then we calculate the sum of those ASCII value after that we take mod of no of servers. So that we get randomly distributed hash function for all keys.

Decentralized Index Server API:

Decentralized index server is written by keeping in mind of scalability. To make server more scalable I have created an API for the server:

1. This API can be exposed to any application for accessing required functionality of server without worry of internal functionality.
2. We can change our server implementation any time on need basis without changing the API that is for client side.
3. As API is written for generic use so any developer can change the internal functionality without worry or hampering the outer server functionality.

Managing Peer Connections

Decentralized index server application accepts connection at any given port given there is no process running on that port by default server accepts connection at port no 9000. Every time any application makes connection to server listening port. CISServer instance acting as producer put that connection in LinkedBlockingQueue and pulled by ConnectionHandler instance acting as consumer from the queue. ConnectionHandler then gives that connection to ConnectionManager instance. ConnectionManager instance takes a Runnable instance of PeerConnection object for every connection made to the server.

Every connection is handled by a new instance of PeerConnection, which is Runnable.

Adding a Peer Connection to Server

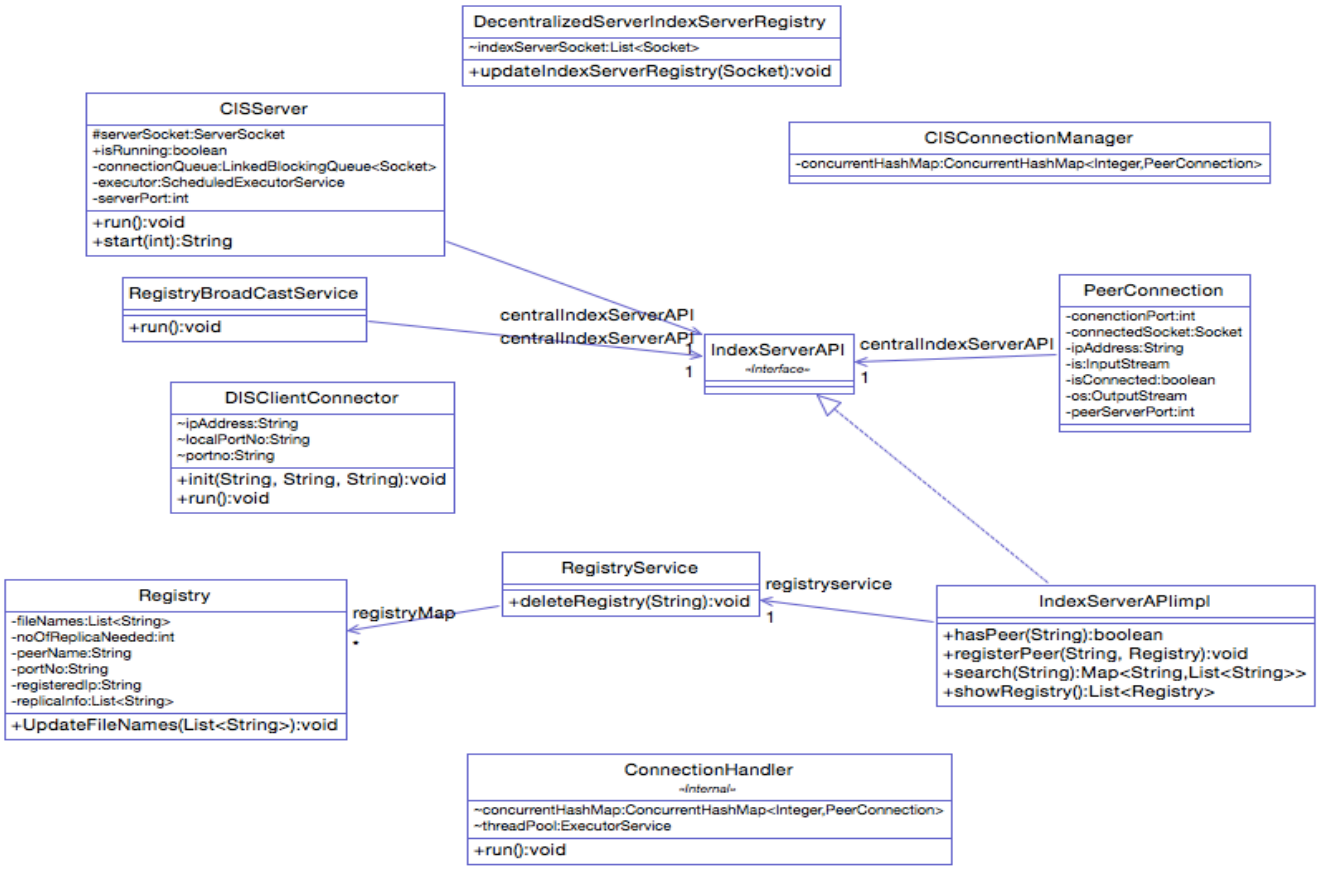
Once the connection is added to ConnectionManager. PeerConnection object send message to connected peer connection for creating a registry in server by providing Peer Name, Peer File transfer port and Peer File Names.

Serializable objects for data transfer

Decentralized index server only accepts the Command object that travel through network because Commands object as Serializable and makes the connection between peer and client more secure as any client that does not have correct command object cannot communicate with server or register with server.

If the peer connection sends the specific command for registering of peer then Decentralized index server creates a new registry for that peer and add that connection to registry service as well as allow that client to search for file names from Decentralized index server using command objects that are understood by Decentralized index server. Otherwise the connection is disconnected from server and removed from connectionManager.

Class Diagram of Decentralized Index Server



Peer Application

Peer Application is designed in such a way that it act as client for getting file names from Decentralized index server as well as work as server to other peer for downloading data. It's a multithreaded application that has one thread acting as client for Decentralized index server, another thread working as server for accepting the connection from other peers to facilitate file download functionality as well as there is another scheduler thread working ever 5 min to check the change in files in directory of peer and updating it to Decentralized index server

Peer Application acting as client

1. Making Connection with Server: When we start the peer application it creates a connection to Decentralized index server using the default IP address 127.0.0.1 and port no 9000. It can be configured to other IP Address and port no
2. Registering to Application to Server: Once Peer Application gets response form index server it ask the client to register the peer to Server by giving the peer a name and providing a directory from where peer application can share it's available file list as well download the available file form other peers.
3. File Look up in server: After the peer is registered then application ask to search any file in server and server give response with matching filename as well as location of peer IP address and port no.
4. On background there is a thread running every 5 mins for lookup of change in directory files. If there is a change in files it will send server command to update the peer files.

Peer Application working as File download Server

Every peer client has an additional thread working as server that accepts request from other peer and allows those connected peer to download the file from it directory. Port no of every peer application can be configured using properties files. Peer file download server instance can handle multiple peers those acting as client to the file download server.

Peer Application working to replicate files

Once the Request from Server comes to create a replica of peer2. Peer1 ceates a TCP connection to the peer and ask for all files to Peer2. Peer2 send all the its files present in working directory to peer1 and peer1 stores all the files of peer2 in its working directory.

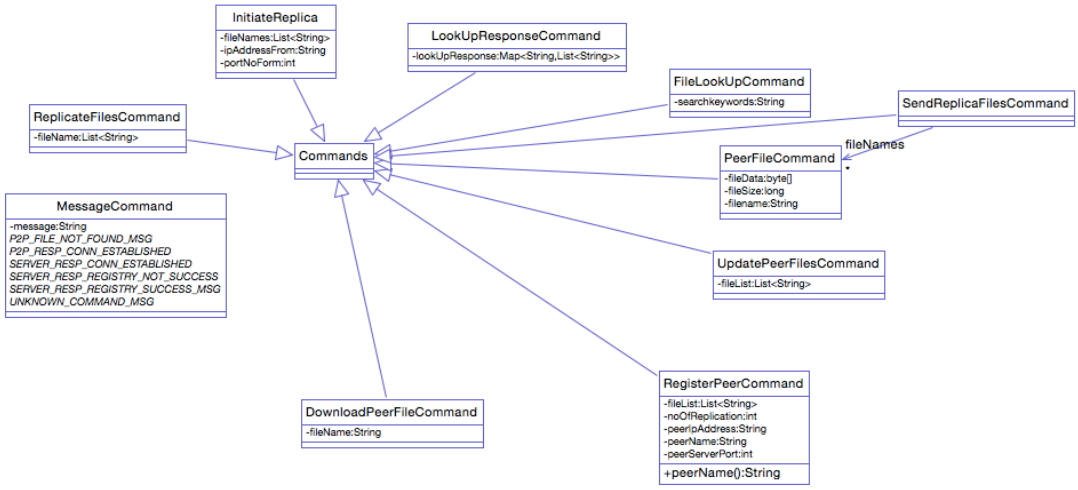
Communication Between Decentralized Index Server and Peer

To communicate between server and client we used serialized object and Enum, which is also by default serialized in java. Parent of the serialized object is named as "Command" Object and all other other serialized object child of this Command.

Benefits of Using Serialized object for communication

- Every Communication between Decentralized index server and peer is done by using Command object of Message Enum.
- Command Object makes the application more secure as anonymous peer does not having the correct commad object will not be able to send request to server.

Class Diagram of Command Objects



Language and Data structure Used

Programming Language: Java
Build Tool: Apache Ant
Network Connection: Socket
Connection Management: Using Java Concurrency package
Data Structure:

1. LinkedBlockingQueue: For Peer connection management
2. ConcurentHashmap: For keeping connection of Peer. where key is coonectedport and value is PeerConnection Object)
3. HashMap: For keeping connection successful connection of peer. Where key is combination of IP address of Peer and File transfer port and value is a custom Object that keeps record for every successfully added peer and its file list.
4. Registry Object: keeps information about connected peer and its file list