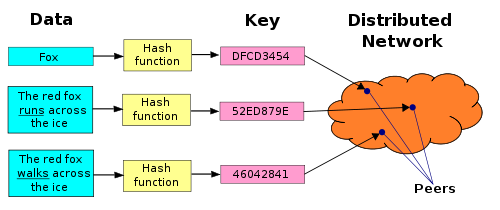
**Design Document**

In this project we are implementing a decentralized peer to peer file sharing system. It is built on the previous two assignment viz centralized indexing server with metadata about file location and second one where distributed hash table for storing the meta data was implemented.

The peer register the files they want to share with other peers to one of the servers with contains the distributed hash table. The server that the peer tries to register with applies hashCode function to the file name provided and decides on which distributed server the meta data is supposed to be stored.



**Server Side**

The Server implements the distributed hash table . When a client tries to register /deregister/download a file the server applies the hashing function to the filename to decide on which server the meta data is present/should be stored. Each of the server is connected to all the other servers which contain the distributed hashmap. When the requested data is on some other server , the server contacted by the client passes messages to the other server and obtains the information and returns the data to the client. Data resilience has been implemented to prevent the loss of meta data when a server goes down. The number of replicas can be configured (here the simple logic of storing replica of metadata of the server onto a server with the next id). The clients are aware of the replica server , hence when the primary server is down the request is redirected to the secondary sever.

**Client Side**

The client connects to any one of the servers and registers/deregisters/requests for a file. For downloading ,the location of peer which has the file is obtained and then the peer that requested for the file connects to peer hosting the file it and downloads the file. In case the client does not want to share a file it had already registered it can deregister the file. File replication has been implemented to prevent loss of files if the peer hosting a shared file goes down. Each peer during registration of a file to the distributed server sends the data to its replica peer. When registering the file the peer registers both itself and its replica peer.

Server Side Implementation

DistributedServer.java

This class triggers the server , connects with other distributed servers , opens an port and keeps it in listen state for client requests. Each time a clients connects to it creates a new thread to process the client’s request.

DistributeServerOperate

The class is used by the server to interact with other distributed servers. When the requested data is not in its own hashtable the server uses class this to communicate with other servers and retrieve metadata and it also responds to requests sent by other servers.

ServiceClient

This class is called by the DistributedServer each time the client connects to the server.It implemets multi threading on the server side. The hashfunction is called in this class , it is used to determine on which server metadata on requested file is present or should be stored.

Utility Class

This class has generic utility functions. It is also has the connectToServers() function in which is used by the server to connect to all other servers before it begins to accept client requests

Client Side Implementation

PeerClient

This class triggers the client application and provide the user an interface to send requests to the server. It also creates a new thread when another peer sends a request to it, thus implementing multi threading on the client side.

PeerServer

This class is used by peer to interact with other peers to send and receive files.

Use of property file

To prevent hardcoding of values within the code , PeerProperty.properties file and Serverproperty.properties file is used for running the server side and peer side code.

The Property file has parameters such as server ip, port , shared file location, download location on peer.

Improvements

* The number of distributed servers can be made dynamic

This would need detection that the server has left the network and altering the number of servers in the hashing function.

* Number of replicas can be made configurable

The code where data resilience is implemented has to be put into a loop so as to connect and send data to the replicas. A little more complex implementation than the current one would be making the peer/server closer as replicas thereby reducing network overhead.

* Better algorithms can be explored to implement the hashing function to ensure equal distribution of the metadata on the different servers