Design of P2P File System

Main idea:
Implementing Java RMI to achieve remote procedure invoke, providing remote object and service interface to client , using Socket to transfer file between two peer node. We simulate different machine by deploying different client on different directory and using different local port to communicate.

Client side:
Each client has a serversocket to listen and accept other peer nodes' requests and has a local server_stub which is a remote object, client can invoke registry() and search() through it. When client wants to retrieve file, it first call search(), marshalling parameters to remote server, when searching is done then it returns the target port of node and set up another socket to communicate with the seeder peer node. There is a background thread keep running to automatically update each node's file that registered on the server.

Server side:
We implements a concurrentMap to store the mapping from specific port that a peer node reside at to a list of files that peer node has. There is a thread-safe list to store the port num of active node(portList), when we search for a file, we traverse active port in portList and get the mapping fileList to check whether or not it contains the file. When a client tries to register, we simply add its port num and mapping from port to files.

Methods:

RetrieveFileThread
A thread for client retrieve data using socket connection. Sending filename to seeder node first, seeder writes data into socket and client writes it into local file.(downloading).

SendFile
A thread for client uploading data.

Seeder
A thread for client listening request. A serversocket keep running until receives a request and setup a new socket to upload data.

UpdateFile
A thread for client automatically update it's file info onto index server every 1 seconds.

RegistryThread
A thread for server register active peer node, store its port num and a mapping to its files.

Searchthread
A thread for server searching through the index and return the desired port nums, which is a list, for that different nodes may have same files.

Possible improvements:
1. Client updates its files every 1 second to achieve auto updating. This may cause issue in that some node may request during the update interval and client just delete that file. So we may find another way to inform index sever and update the change.
2. When different nodes have same files. We can sort the node by put the least occupied node at first and let user know which node is faster. This needs extra algorithm and data structure.
3. We cam make client code and server code isolated and let client only compile its own code instead of server-side code.
4. There might be potential concurrent problem.