CS550 Advanced Operating Systems
Programming Assignment 3
Design Document

submitted by:
Jayanth Vangari
A20337867

The assignment is on implementing a decentralized file sharing system.
Each of the peers can act as a client and a server , and as a server peer has indexing functionality, hence it is decentralized indexing. Each Peer as a server has its own internal Hash table for indexing files of other peers , with file names as keys of the hash table and peers information as values.

The network is assumed to be static. In the Initial start up, each of the peers when created reads a “config” file that contains information about the other peers in the network. The config file has information about IP address, port and unique node ids. The config file allows the peer to connect all other peers in the network during bootstrap and maintain the connections.

When a peer is created , a directory is created for it(**peer$i**) and it connects to the other peers in the network . and client registers its files at multiple servers thus distributing them, by hashing its file and registering the file at node that is represented by the value returned by the hash function In order to realise resilience, a file can also be replicated at the one of the neighbouring nodes when registering files. The replication of file at other node allows it to get access to the copy of file even if the primary node holding the file is down. Each of Servers is multithreaded to facilitate multiple peer registrations and requests.

When a peer requires a file, it looks up in the Index server at the peer containing information about the peer having the file. If it is present, the address and other information of other peer/peers list is sent back to the peer , with which it selects a peer from the list, connects to it and requests a file and downloads it.

The above scenario is implemented in JAVA .

Functionalities:

1. **Decentralized Indexing Server:** Each of the Peers having indexing capability indexes the contents of various peers in its internal Hash table, and provides the search facility to other peers.

registration : when register is invoked by a peer, the server receives the request does put operation of file names and peer information in its hash table . If multiple peers possess the same file , multiple peer locations are stored in its hash table.

search: when search function is invoked by peer, it requests the peer information containing that file. The server does a get operation of peer information by using file name key and returns the list of peers having the file.

2. **Peer:**

Peer acting as both client and server performs the following functions

1. **createpeer():** Initially, a peer is created and It attempts to connect to the all other peers

2. **register():** The peer registers the files extracting names of files contained in its directory at , along with the peer address and port,and directory path.

3. **retrieve():** Peer looks up for a file at the appropriate decentralised indexing server,sending a request containing the file name, and in return receives a list of peers having that file and their information.

4. **selectpeer():** From the list of peers received, the peer selects one of the peers to obtain the file from this method returns a type class Location which contains filename, peer address and peer port. that is passed as argument to Obtain() to make a request to other peer.

5. **Obtain():** Peer downloads the file by attempting to connect to the other peer and then sending request to the peer acting as server, In the request it sends the file that is required by it. Peer Server obtains the file name , sends the file by uploading a copy of the file from its directory.

Replication: The system ensures data resilience by replication of files across nodes . Each file has a replica present at its neighboring node . So, in case primary file is lost ,replica can be accessed.

When a peer registered its files at any of the indexing servers, it replicates the file at the node returned by the indexing server. When other peer receives the replica, it should register the file ,so any of the peers looking up for the file can know the file is at multiple peers , and can access the replica when it cannot connect to the primary file holder. The replica file location is also returned when a peer issues a retrieve operation.

Ensuring Concurrency:

Being Peer to Peer communication, ensuring concurrency in the system is obvious.

1. Concurrency is observed when multiple peers are registering and requesting to Lookup files at the decentralized Index Server.

2. When multiple peers request a same file at the other peer. In this case, the peer which now acting as the decentralized index server should handle these requests concurrenty and send the copy of files to the requesting peers.

Concurrency in my program is realised by the concept of Multi-threading in Java. Server part of the peer are implemented with multithreading.The Peer acting as server implements multithreading by ClientHandler() method. And also, The Peer server itself is created as thread to the program Peer.java . Peer server spawns a new thread for every file request obtained from other peer. So, multi-threading plays a pivotal role in handling file requests among peers.
Java and the concept of socket communication is used among peers and central serve to communicate and transfer files.

Improvements and Extensions to the Program:
When a replica file is received by any of the peers, the replica should be registered explicitly if it is not at any of the indexing, rather this process could be automated .

Scalability becomes an issue at larger scale with sychronized multi-thread. Asynchronized communication through sockets shall be preferred then.

Thread generation doesn't seem to add that much overhead here, but when there are large number of peers communicating and transferring files Thread pooling is often more convincing approach to handle multi-threading, to reduce the significant overhead of thread creation for each peer client request., that can be done by creating a pool of threads using new CachedThreadPool or newFixedThreadPool methods for creating threads and queueing the tasks giving them when thread is ready to do task.