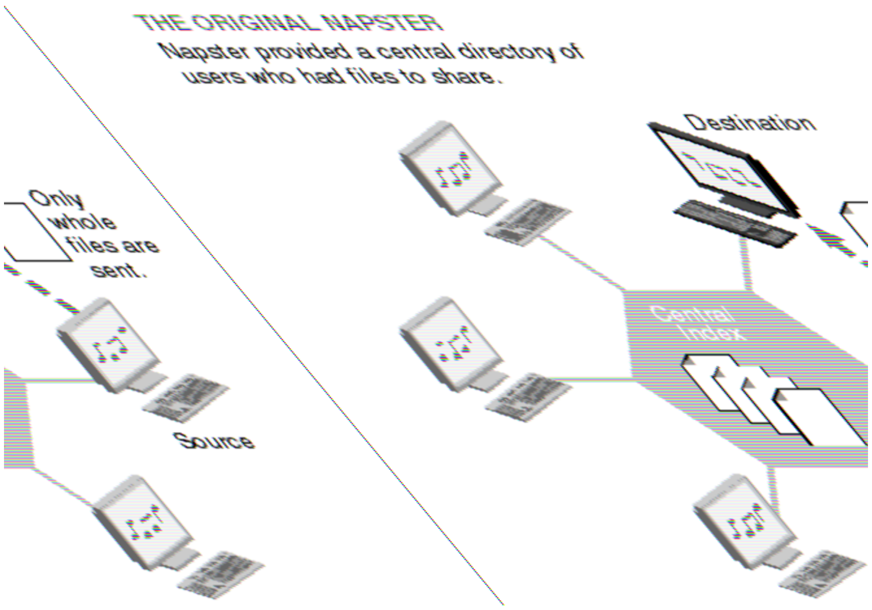
Design Document

In this project a Napster style peer-to-peer file sharing system has been implemented. A peer to peer file sharing systems allows the users to connect with other peers on the network and allows search and download of files. In these systems, the peers interact with each other to exchange data unlike the centralized system, where the central server hosts the data.

Principle:
Napster file sharing system has an index server which has the indexing of filenames and location information of peers which have the corresponding files. The peers first register the files that they want to share, on the indexing server. The registration comprises of the filename and access information of the peer hosting the file. When any peer in the P2P network requests for a particular file, the request is forwarded to the Indexing server which returns list of peers in the network which hosts the file. The peer which initiated the request then chooses a link(i.e peer) , connects to it to and downloads the file .


THE ORIGINAL NAPSTER
Napster provided a central directory of users who had files to share.
Destination
Only whole files are sent.
Central Index
Source

Advantages of Napster Peer-Peer file
- It is easy to set up a peer-to-peer system.
- The nodes can act as server as well as a client.
- Failure of one peer does not affect the other peers
- It is easily scalable

Disadvantages of Napster Peer-peer file

- There security is low, peers can download files from an untrusted source.
- They are not very useful when number of peers are less

Architectural design

The implementation is divided into two sub-category the mainly the Server side and the Peer side. The Server is known as the indexing Server and a number of peers are which register with the indexing server.

The Indexing Server

- The peers registers all files that are present within a folder (shared folder)
- Through registration the filename and peer’s access information(Ip address and port) are stored in the indexing server.
- When a peer requests for a file the indexing server returns the id’s of the peers which have the requested file.
  Note: The server does not store any data apart from indexing information of the files.
- The indexing server can handle multiple requests at the same time.
- The filename is used as a key in the indexing server.

Peers

- The Peer can act as both as a server and a client.
- As a client it requests the indexing server for a particular file
- With the information provided in the response of the indexing server it connects to the peer in the P2P network which has the desired file.
- The two peers interact with each other ,one peer acts as a server and sends the requested file while the other peer acts as a client and downloads the file.
- The peers register the files the want to share with the other peers on the network.
- Since the peers act as Servers also the their Server port is kept open so that other peers can access them.

Implementation of the file sharing system

The file distribution system is implemented by the following classes.
Server side- IndexServer.java , IndexRegisterOperate.java
Peer- Peer.java, PeerServer.java

IndexServer.java
The class implements a multithreaded indexing Server. It has a Server socket open on a port and it is on listen mode, when a client sends a socket request to the port it accepts and creates a socket connection . After the socket connection to the peer is made then it transfers the control to the IndexRegisterOperateClass with the new thread.

IndexRegisterServer.java
The IndexRegisterServer has functions which accept the registrations from the peers and responds to requests about file location from peers.
On receiving the request for registration of files to the index server from the peer, all files names in the particular shared folder location of peer is received by it and it maps the file name to the peer's access information(i.e Ip address and port). LinkedHashSet is used to store the Ip address and port of peers which have the file , this prevents duplicate values of peer information for the same file when registration is done periodically.

The list of files to be registered is sent by the peer, the filenames is taken in an array and iterated to map the file names to the peer id. HashMap is used by the index server to store the filename and map it to the peers which have the particular file. The HashMap is updated during each registration process invoked by the peer. The **filename is used as the key** to store and retrieve values of peers.

Data Structure to store file name and information of peers having the file.
HashMap<String filename>, LinkedHashSet <String Ip and ports of peers>

HashMap

Peer.java
This file contains code which initiates two threads for the peer , one to act as a client and another to act as a server. For running the peer this file should be executed.
startClient() starts the client thread
startServer() starts the server thread

startClient() method allows the peer to choose whether it wants to register its files to the index server or wishes to download a file and gives an option to exit the client mode.
For registration, the method sends the list of file names in its shared folder to the server.
For downloading of a file the method sends a request to the index server with the filename it wants to download on receiving response from the server with list of peers having the file and

startServer() starts a thread for the peer to act as a server by opening one of its port and keeping the port in listen mode. It creates a new thread for each new request it receives for file download and passes the control to PeerServer.java class. Thus making the peer a multi-threaded server.

PeerServer.java
It has the run method of the of the threads initiated in the startServer() method.
This class file implements the server code of the peer in the run method. It accepts requests for files from other peers in the network and and sends the requested file data. It reads the file in its location through BufferedInputStream and writes it to the BufferedOutputStream and closes the socket connection once the file transfer is done. **The project supports transfer of .txt, .bin, .jpeg , .pdf files**

Use of property file
To prevent hardcoding of values within the code , PeerProperty.properties file and Serverproperty.propertie 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.

ClassDiagram:

Indexing Server:

Peer

Challenges

Heap space error when executing on VM due to to shortage of memory, it was overcome by assigning more memory to the vm.

Prevention of duplicate values of peers corresponding to a file particular in the indexing servers , LinkHashSet is used to avoid this. Also the data was getting erased when different clients tried to register , the HashMap used to store file name and peer id's(i.e Ip and Port) was made a static variable.

Multithreading on the peer when it acts as a Sever, i.e thread within a thread . The first thread is to run the server code on the peer, this thread id then multithreaded to make the peer a

Testing on geographically distant peers would provide more realistic results during performance testing. Since all the systems under test where in the same geographic location.

Testing on multithreading and concurrency was not full fledged due to limited of resources.

Some Tradeoffs

- Sharing of an entire folder by peer instead of each file at a time. This is done so the the files are registered in batches. So if the peer has to register only one file then only that file must be present in the shared file location.
- The folder location is not stored on the index server only its Ip and port .The location is of the folder is in the property file, this is for the sake of simplicity. The folder location can be added to the hashmap on server along with ip of the peer.

Enhancements which can be done on the project:

- Deletion of peer id on index server when a particular file is deleted from it the shared location folder
- Secondary index server which is updated periodically , in case of any failure in connecting the primary indexing server , the peers are redirected to the secondary indexing server.
- To increase security , connection to other peers can be done through ssh , so that security increases as authentication is required
- Duplication of data, each peer has a copy of its data on another peer , hence data resilience can be achieved.
- Checks if the peer already has the file somewhere locally before downloading
- What if peer requests for a file that it has already registered with its own ip
- Deletion of peer registering corresponding to a file when the file is deleted from the peer
- Persistence of the registrations on the server even after it has been closed.

Performance analysis:

For the analysis of how the program worked, some performance and load testing was done. Registration of a large number of files by peers concurrently was tested. And the average time taken by the server to process the registrations is observed.
Lookup of a large number of files is done through concurrent peers and the response time of the server

To test the register scenario

PeerRegisterTest.java file is used .It creates a huge number of files say 1000 on the peer(The number can be varied by editing the "FileCreationStartNuminName" and the "FileCreationEndNuminName in the PeerProperty.properties file (Note when file chooser pop up occurs twice when choosing file on running this test script).Files created by the test script has the following naming convention : test**($num)**.txt

Files required to run this test :

Snippet of Index server output for load registering

```
key, test1959.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
key, test1696.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
key, test1370.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
```

```
key, test1216.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
key, test1287.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
key, test1787.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
key, test1578.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
key, test1091.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
key, test1666.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
key, test1742.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
key, test1356.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
key, test1819.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
key, test1624.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
key, test1221.txt value [192.168.2.7|8567|/home/vthangap/test_mine/]
```

To test the lookup scenario

PeerForLookUpTest.java is the file that is used. It looks up for a large number of files by sending search requests to the file. This test is usually preceeded by the reigister test, so that a huge number of files are created and registered to the server.
The 'start' and 'end' variable in the PeerForLookUpTest.java file can be modified to look up for a range of files. Eg start= 1048 , end=1058 the the look is done for the filenames test1048.txt, test1049.txt,test1049.txt…….. upto test1058.txt

Snippet of lookup test peer side output:

```
Location of test1253.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
Location of test1254.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
Location of test1255.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
Location of test1256.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
Location of test1257.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
Location of test1258.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
Location of test1259.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
Location of test1260.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
Location of test1261.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
Location of test1262.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
Location of test1263.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
Location of test1264.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
Location of test1265.txt is 192.168.2.7|8567|/home/vthangap/test_mine/
```

Class file:PeerRegisterTest

Average time to register with different number of clients run concurrently

Number of clients(peers)	No. of files registered to index server	Average time to register one file(ns)
1	1000	9270
2	1000	69038
3	1000	1998675

Observation

- As the number of clients connected to the server the average time taken per register increases
- The server response time increases when many clients are registering with it concurrently.
- When the number of files to be registered increases then the response time also increases

- The processing speed of peers vary .

Class file: PeerForLookUpTest
Average time to lookup with different number of clients run concurrently

Number of clients(peers)	No. of files looked up in index server	Average time to look up one file(ms)
1	1000	1768
2	1000	25578
3	1000	94890

Observation
As the number of peers increases the response time of the index server increases.
Also when the number of entries in the index server is more the processing time for look up is more.

Sample Output
For registering files
Server side:

```
property file chosen is /home/vthangap/workspace/IndexServer/src/ServerProperty.properties
The value of serverPort is 7896
1userChoice!
```

```
192.168.2.7
The port number that the client has sent to server for future request8567
into get file function
2
test1.txt
test2.txt
end of get file function
key, test1.txt value [192.168.2.7:8567]
key, test2.txt value [192.168.2.7:8567]
```

Peer Side:

```
property file chosen is /home/vthangap/workspace/PeerCode/src/PeerProperty.properties
hello welcome to Registry server
Please enter 1. to register the peer 2.to download a file 3. To quit
1
You want to register the peer ...please wait
File test1.txt
File test2.txt
Ip address of the client to be registered /192.168.2.7
registration complete
response time for the request!!!351902670
```

For download of file

Server side:

```
2 userChoice!
test2.txt
```

Client side

```
hello welcome to Registry server
Please enter 1. to register the peer 2.to download a file 3. To quit
2
Please enter the file name you want to download
test2.txt
192.168.2.7:8567
Into download function
Enter index number the link from where you want to download the file
Index:0. 192.168.2.7:8567
0
IP chosen 192.168.2.7
Port 8567
true
starting download of file test2.txt
File test2.txt 37 download complete
```