PROJECT TITLE :

Napster-style Peer-To-Peer File Sharing System

Software Design Document

Name : Karthik Krishnamurthy

CWID : A20344464

Subj : CS-550 Advanced Operating Systems

Date : 09/21/2015

Project Title: Napster-Style Peer-to-Peer File Sharing System

Introduction

Peer-To-Peer (P2P) Technologies have been widely used for Content Sharing. Some of the existing examples of P2P File sharing applications are Napster, Gnutella, Free net etc. The Design of these systems is the concept of files distributed throughout Nodes. The P2P system is different from the older Client/Server Models where the files would reside on one Central Server and all the transfers would happen only between the Central Server and the Clients. In P2P File Sharing Application, the File transfer can occur between the individual Nodes/Peers.

Description

The Project is based on a Hybrid P2P model which involves a Central Index Server to obtain meta-information such as the identity of the Peer like the Peer ID and also the Peer on which the information is stored, such as the file names and the location. In this model, the Peers contact the Central Index Server to Register the Files for Sharing, Searching for other files and also to obtain the information on the files present on other Peers which are available for Download. In this model all file transfers made between peers are always done directly through a data connection over the Socket that is made between the peer sharing the file and the peer requesting for it.

Purpose

To Design and learn the internals of Napster-style Peer-to-Peer File sharing System.
And also to familiarize with the concepts of Sockets, Processes, Threads and Makefiles/Ant.

Requirements

Hardware requirements:

a) Single system for simulating the Central Index Server
b) Multiple systems to simulate the Peers.
-OR-
c) Multiple Virtual Machines to simulate the Central Index Server and the Peers.

Software requirements:

a) Machines running on Linux for both the Central Index Server as well as the Peers.
b) Java Development Kit (JDK), JRE (Java Runtime Environment) and Eclipse/Netbeans (IDE).

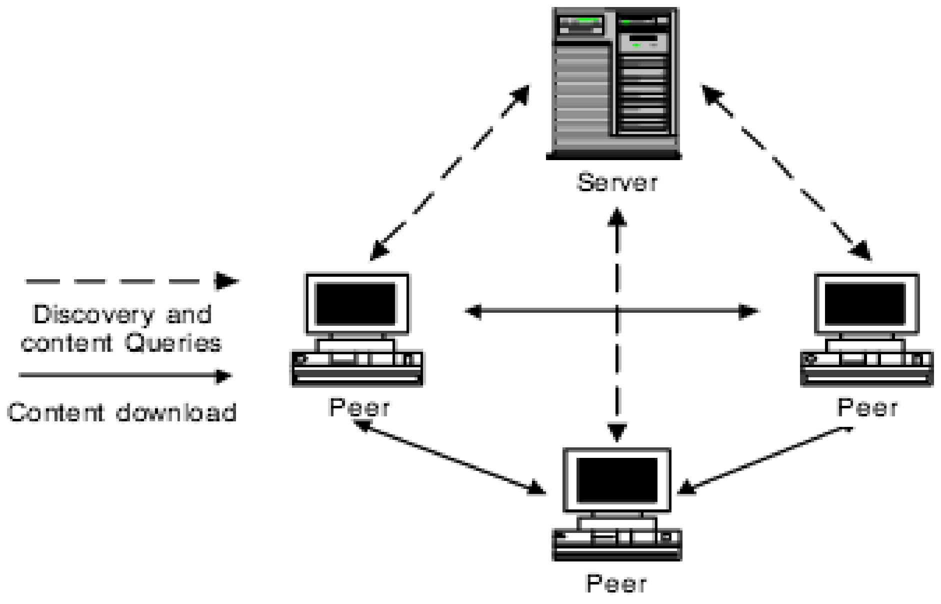
Programming Language used: Java and also Ant Script for Build Automation.

System Architecture

The System is designed as a Hybrid P2P system or a Server-mediated P2P system. This P2P architecture works just like the pure P2P architecture except that it relies on a Central Index Server for peer discovery and content lookup.

In this model, the P2P file sharing application usually notifies the Central Index Server of its existence when it Registers or tries to Search the content on the Central Index Server. The application (peer application) then uses this server to look for some particular contents such as files and it queries the Central Index Server rather than sending queries to each peer. The Central Index Server then responds with a list of the peers that possess the requested content and the peer application can contact those peers directly to retrieve/download the content. In this model, it is easier to make this solution scale better than the pure P2P model because peer discovery and content lookup only need to send a message to the Central Index Server instead of all peers.

The Figure below shows the P2P architecture:



==

Protocol

This model is based on the Napster protocol. In the protocol, it is divided into Peer to Server communication and Peer-to-Peer communication. Each connection, request and response is carried over the Socket.

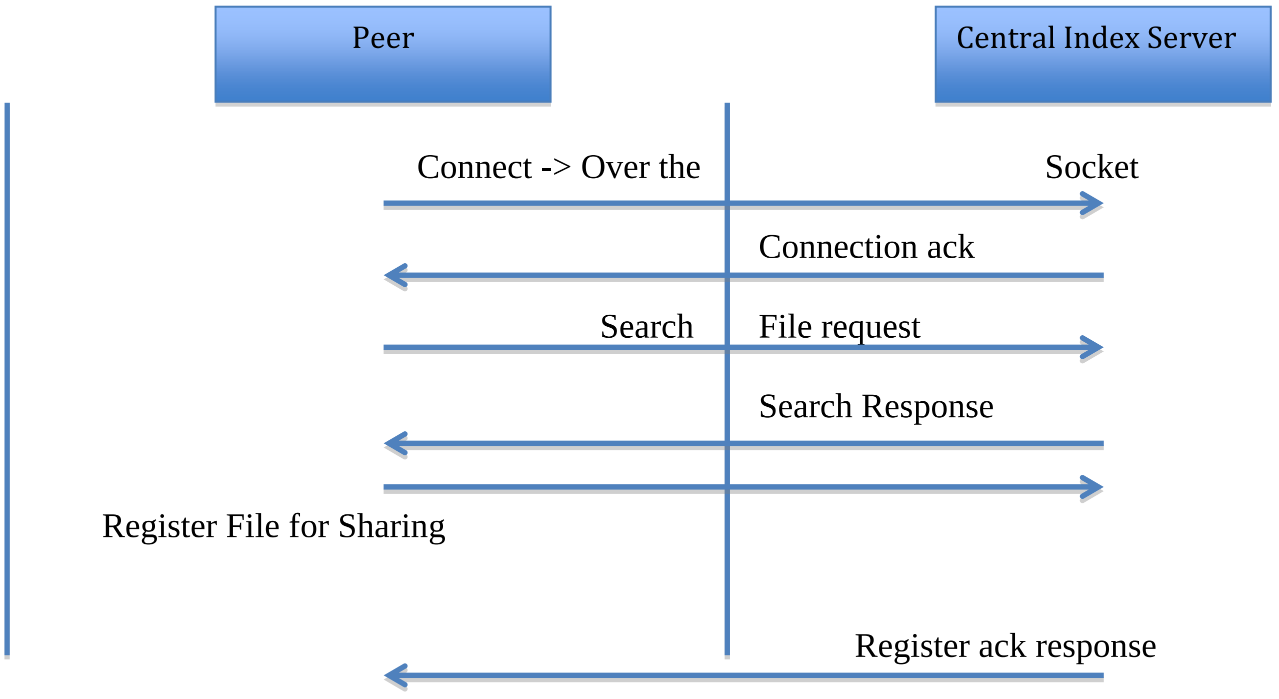


Figure: Shows the Logical flow between the Peer and the Central Index Server

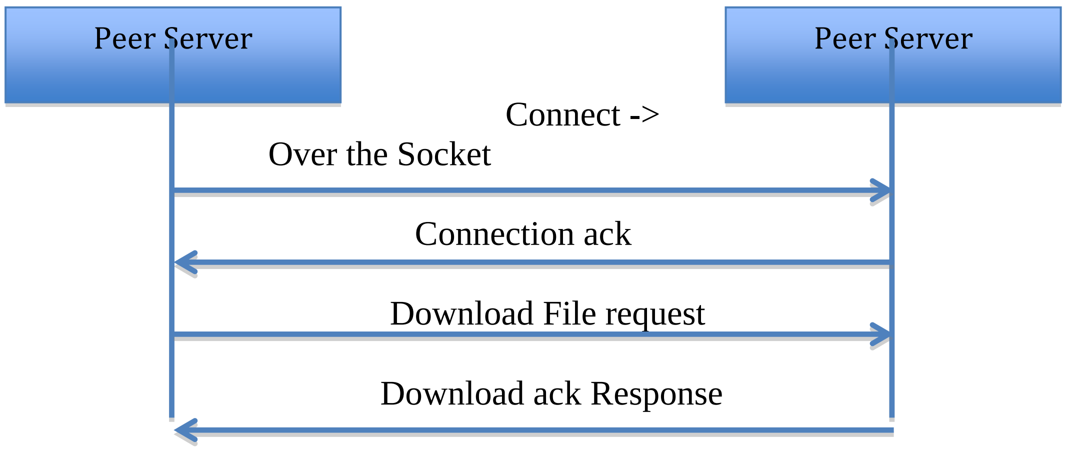


Figure: Shows the Logical flow between the two Peer Servers.

Design

The P2P File sharing system is designed keeping in mind the P2P architecture and its underlying protocols.
The entire design is implemented using Java and some of the abstractions used are: Sockets and Threads.
The project is built on Linux Operating system

The P2P file sharing system has two components:

1) **Central Index Server:**
   This server indexes the contents of all the peers that register with it. It also provides a search facility to the peers.

   The Central Index Server provides the following Interfaces to the Peers:

a) **Registry (peer id, filename)** – invoked by a peer to register its files with the Central Index server. The CIS then builds an Index for the peers.

b) **Search (filename)** – this procedure searches the index and then returns all the matching peers to the requestor.

2) Peer:

The peer acts as both a client and a server. As a client the user specifies the filename with the indexing server using "lookup". The Indexing server then returns a list of all other peers that hold the file. The user can then pick one such peer and the client then connects to this peer and downloads the file. As a Server, the peer waits for requests from other peers and sends the requested file when receiving a request.
The Peer Server provides the following interface to the Peer client:
a) Obtain/Download (Peer ID, filename)- invoked by a peer to download file from another peer.

Implementation

The Implementation is done using Socket Libraries for communication between the Central Index Server and the Peers as well as between the Peer Servers. The CIS and the Peer-Servers are Multi-Threaded.
The various Functional Modules are :
a) Register
b) Search
c) Download

The Central Index Server [CIS]:
When the CIS is run, Initially the Index Server will be in the Idle State.

The CIS Socket then keeps listening on a Port for incoming connection requests. Whenever a Peer makes a request on the CIS port, the CIS serves the request for the connected peer. The Registry and Search method is implemented on the Central Index Server.

a) The **<u>Register</u>** Method is implemented as follows:
When the peer invokes the register method to Register its files with the Index server, the Server has a RegisterRequestThread(); running and listening on a Port to serve the Register request.
The Peer enters the Peer ID followed by the Filename which it wants to Register.

The Peer then invokes the RegisterWithIServer() method. As soon as this is done, a Socket connection is automatically established with the Central Index Server with the CIS_ip and the port number.
The Index Server maintains an ArrayList to hold all the values of the Peer ID's and the Filenames with the corresponding Peers. This ArrayList serves as the Index for all other peers for their requests. The Peer ID and the Filename that the Peer wants to register is then stored in this ArrayList as an Index catalogue. This completes the Registration procedure.

<u>Registration Pseudocode:</u>

```
Index Server – Runs a RegisterRequestThread() and listens for active registration requests.
→ Request for Register from the Peer
→Peer enters the 4Digit Peer ID and the Filename it wants to register
   Example: 3001 testfile.txt
→ RegisterWithIServer()
→ Request for Socket with (CIS_ip, Port No)
→ Peer sends Peer ID and filename over the socket through the Output stream.
→ Server stores the Peer ID and the Filename received over the Input Stream into the Index ArrayList.
→Index Server serves Register request for the Peer.
```

<u>Code Snippet for Registration:</u>

```
public void RegisterRequestThread()
   {
             Thread rthread = new Thread (new PortListener(2001));
             rthread.setName("Listen For Register");
             rthread.start();
    }

public void RegisterWithIServer()
    {
                   //1. Creating a socket to connect to the server
                   requestSocket = new Socket(CIS_ip, 2001);
                   System.out.println("\nConnected to Register on CentralIndxServer on port \n");
```

```
                //2. To Get Input and Output streams
                out = new ObjectOutputStream(requestSocket.getOutputStream());
                out.flush();
                out.writeObject(regmessage);
                out.flush();
            }
```

<u>Index Server:</u>

```
if(port==2001)
            {
                server = new ServerSocket(2001);
                while (true) {
                connection = server.accept();
                 System.out.println("Accepted Connection From"+connection);
                 System.out.println("ConnectionReceivedFrom"
+connection.getInetAddress().getHostName()+ " For Registration");
                    ObjectInputStream in = new
ObjectInputStream(connection.getInputStream());
                    strVal = (String)in.readObject();
                    System.out.println(strVal);
                /* print substrings */
                    for(int x = 1; x < var.length ; x++){

                        //  myIndexArray[maxsize].peerid =   .;
                          begin myitem = new begin();
                          myitem.filename = var[x];
                          myitem.peerid = aInt  ;
                          myIndexArray[maxsize] = myitem;
                          maxsize++;
                    }

                    in.close();
                    connection.close();
                 }
            }
==============================================================
```

b) The **<u>Search</u>** Method is implemented as follows:
When the peer invokes the Search method to search for files with other peers on the Index server, the Server has a SearchRequestThread(); running and listening on a Port to serve the Search request.
The Peer enters the Filename to Search.

The Peer then invokes the SearchWithIServer() method. As soon as this is done, a Socket connection is automatically established with the Central Index Server with the CIS_ip and the port number.

The Index Server maintains an ArrayList to hold all the values of the Peer ID's and the Filenames with the corresponding Peers.
The Index Server then receives the Filename to search from the Input stream and then it traverses the ArrayList to find a match for the Filename searched. If the filename is present in the Index, the Server then returns the Search result with the Filename and the Peer ID with which the File is present. This completes the Search procedure.

Search Pseudocode:

```
Index Server – Runs a SearchRequestThread() and listens for active search requests.
→ Request for search from the Peer
→Peer enters the Filename it wants to search
   Example: testfile.txt
→ SearchWithIServer()
→ Request for Socket with (CIS_ip, Port No)
→ Peer sends the filename over the socket through the Output stream.
→ Server receives the Filename over the Input Stream and then traverses the Index ArrayList.
→ If a Match is found, it returns the Peer ID and the IP Address of the Peer which contains the searched File.
→ If no match is found, it returns "File Not Found"
→Index Server serves Search request for the Peer.
```

Code Snippet for Search:

```
public void SearchRequestThread()
   {
              Thread sthread = new Thread (new PortListener(2002));
              sthread.setName("Listen For Search");
              sthread.start();
    }

public void SearchWithIServer()
    {
                    System.out.println("Enter the File Name to Search");
                    Scanner in1 = new Scanner(System.in);
                    searchfilename = in1.nextLine();

                    //1. Creating a socket to connect to the Index server
                    requestSocket = new Socket(CIS_ip, 2002);
                    System.out.println("\nConnected to search on CentralIndxServer on port 
2002\n");
                    //2. To Get Input and Output streams
                    out = new ObjectOutputStream(requestSocket.getOutputStream());
                    out.flush();
                    out.writeObject(searchfilename);
                    out.flush();
                    ObjectInputStream in = new 
ObjectInputStream(requestSocket.getInputStream());
```

```
                    String strVal = (String)in.readObject();
                        System.out.println( "File:'"+searchfilename+ "' found at
peers:"+strVal+"\n");

                }
```

Index Server:

```
if(port==2002)
                {
                try {
                     server = new ServerSocket(2002);

                     while (true) {
                          connection = server.accept();
                          System.out.println("Connection Received From "
+connection.getInetAddress().getHostName()+ " For Search");
                          ObjectInputStream in = new
ObjectInputStream(connection.getInputStream());
                          strVal = (String)in.readObject();
                          String retval = "";
                     //   Peer-id's separated by space are returned for given file

                          for (int idx =0; idx < maxsize ;idx++)
                          {
                               if (myIndexArray[idx].filename.equals(strVal))
                               {
                                    retval = retval + myIndexArray[idx].peerid + " ";
                               }
                          }
                         if (retval == "")
                         {
                           retval = "File Not Found";
                         }
                         System.out.println(retval);

                         ObjectOutputStream out = new
ObjectOutputStream(connection.getOutputStream());
                         out.flush();
                         out.writeObject(retval);
                         out.flush();
                         in.close();
                         out.close();
                         connection.close();
                    }
                }
```

==

c) The **Download** Method is implemented as follows:
When the peer invokes the Download method to download the files with other peers, the Peer-Server which has the file invokes AttendFileDownloadRequest (); thread running and listening on a Port to serve the Download request.

The Peer-Client enters the Peer ID and the Filename to Download.

The Peer then invokes the DownloadFromPeerServer () method. As soon as this is done, a Socket connection is established with the Peer-Server which now acts as the Server to serve the Download request.
The Peer-Server then receives the Filename to be downloaded from the Input stream and then it then opens a Filereader object to check for the file and read the contents and then uses a writeObject to store the contents from the output stream and finally on getting the writeobject with the Filename, it then writes to the file from the Input stream. This completes the Download procedure.

Download Pseudocode:

```
Peer-Server – Runs a AttendFileDownloadRequest () and listens for active download requests.
→ Request for download from the Peer
→Peer enters the Peer ID, IP Address of the Peer holding the file and the Filename it wants to download.
   Example: Peer ID: 3001
            Peer IP Address: 10.0.0.13
            Filename: testfile.txt
→ DownloadFromPeerServer ()
→ Request for Socket with (Peer_Ip_Addr, Peer ID1)
→ Peer sends the filename over the socket through the Output stream.
→ Peer-Server receives the Filename over the Input Stream and then readObject to read the contents of the file and sends it over the Output Stream.
→ The WriteToFile Object then writes the contents to the file, on the requestor Peer
→Peer-Server serves Download request for the Peer.
```

Code Snippet for Download:

```
public void AttendFileDownloadRequest(int peerid)
    {
            Thread dthread = new Thread (new PortListenerSend(peerid));
            dthread.setName("AttendFileDownloadRequest");
            dthread.start();
    }
```

```
public void writetoFile(String s)
   {

                  //To Append String s to Existing File
                  String fname = searchfilename;
                  FileWriter fw = new FileWriter(fname,true);
                  fw.write(s);
                  fw.close();
   }
public void DownloadFromPeerServer()
    {

               System.out.println("Enter Peer id:");
               Scanner in1 = new Scanner(System.in);
               String peerid = in1.nextLine();

               System.out.println("Enter the File Name to be Downloaded:");
               searchfilename = in1.nextLine();

                 int peerid1 = Integer.parseInt(peerid);
                 //1. Creating a socket to connect to the Index server
                        requestSocket = new Socket(ipadrs, peerid1);
                        System.out.println("Connected to peerid "+peerid1);
                        //2. To Get Input and Output streams
                        out = new ObjectOutputStream(requestSocket.getOutputStream());
                        out.flush();
                        out.writeObject(searchfilename);
                        out.flush();
                        ObjectInputStream in = new
ObjectInputStream(requestSocket.getInputStream());
                        String strVal = (String)in.readObject();
                        System.out.println( searchfilename+": Downloaded\n");
                        writetoFile(strVal);
                  }
===================================================================
```

Pros and Cons of the P2P Model

Pros

- Less Network Resource consumption.
- High Scalability

Cons

- Central Index Server dependent.
- Less fault tolerant.

===

Traceability Matrix

P2P Requirement	Module	Functional Module
Requirements	Central Index Server	Registry
		Search
	Peer-Server	Download

===

Tradeoffs

- ➢ Used ArrayList instead of HashMap.

The ArrayList has O(n) performance for every search, so for n searches its performance is O(n^2).

The HashMap has O(1) performance for every search (on average), so for n searches its performance will be O(n).

For random operations of content retrieval, a HashMap is better. For retrieving items in an Order, ArrayList is better. So, there is a tradeoff in the speed in this model due to the use of ArrayList instead of HashMap.

- ➢ Files of large size say few 100MB's cannot be transmitted/downloaded.

- ➢ Automatic Registration of all the files in the folder at one shot is not implemented.

Instead, Multiple files can be registered simultaneously by the Peer, by specifying the Peer ID and the Filenames, it wants to register.
Example: Enter the 4Digit ID and the Filename(s) to Register:
3001 A.txt B.txt C.txt D.txt E.txt ……………. And so on…..

===

Possible Improvements/Extensions

→ Automatic Registration of all the files in the Peer Folder.
→ Support for Large files
Example: Files over a few 100 MB's
→Performance improvement by using different data structures like HashMap instead of ArrayList

→ Support for Data resilience by allowing a data replication factor.
→ User Interface

===

References:

http://arxiv.org/ftp/cs/papers/0402/0402018.pdf

http://computer.howstuffworks.com/file-sharing1.htm

http://www.ics.uci.edu/~bhore/papers/Hollyshare.pdf

https://en.wikipedia.org/wiki/Peer-to-peer