A Simple Peer-to-Peer File Sharing System

Program Design

Allen VanNoppen Jr.
A20415658

Introduction:

Peer-to-peer (P2P) sharing systems are one of the many great developments to come out of the connected world. Made famous (or infamous) by software companies such as LimeWire, Napster, and Bit Torrent, P2P sharing systems allow users to send and receive files to one another easily.

While those larger companies employ many different layers of software architecture to achieve scale, a simple network can be created with relative ease using RPC or in this case Java RMI. Below is an outline of the creation of the file sharing system and the improvements that can be made in the future.

Project Goals:

The goal of the project is to create a Simple P2P system composed of at least 3 peers and a central indexing server. Peers act as both clients and servers. A peer must have the ability to either send or request a file from other peers. The central indexing server acts as a directory for the files and the corresponding peer location. Peers first register their files with the indexing server and then have the ability to search that indexing server for a desired file.

The designed P2P system is centralized because of the Central Indexing server's role in directing peer traffic. Figure 1 shows the general structure and process flow of a centralized P2P system.

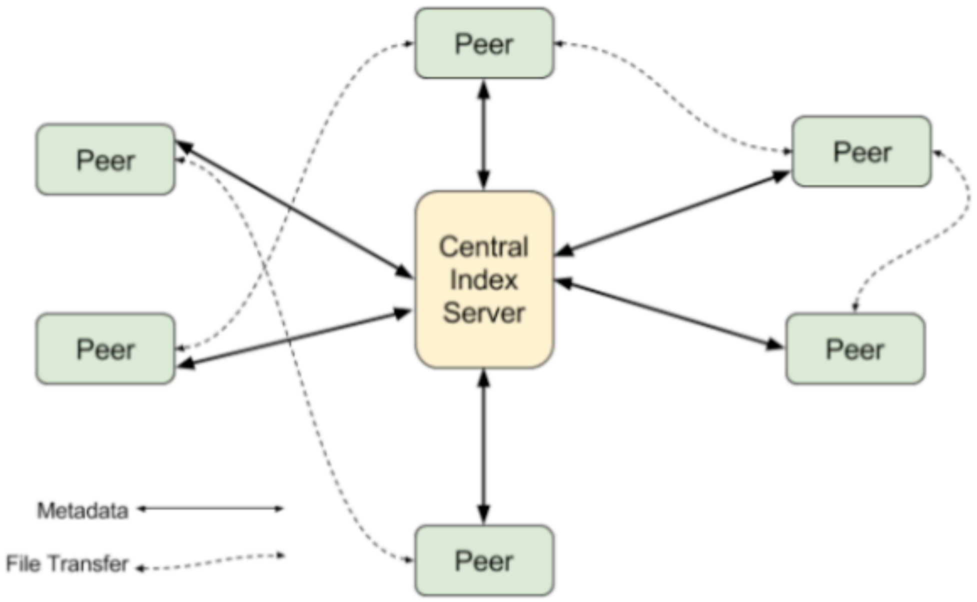

Figure 1: Diagram of a centralized P2P file sharing system [1].

A system like the one in figure 1 also requires that all tasks can be run concurrently. Multiple peers can request from the central index at the same time and peers have the ability to download and send files simultaneously.

Program Structure:

Language/environment: Java, macOS environment.

The list of files included within the P2P file sharing system with a brief summary is:

IndexServer.java
- This file contains different methods for creating the list of files and peers for the central index that can be accessed by the peers. See: peerArray(), indexUpdate(), searchIndex(), addtoIndex()

StartIndexServer.java
- Creates an instance of IndexServer (above) and bnids the local host url with the method.

IndexServerInterface.java
- Interface for IndexServer.

IndexWatch.java
- Watch service for each peer that notifies the index server of file changes thus keeping the central index server up to date of current files in a peers directory.

PeerClient.java
- Contains different methods for interacting with the central index server and downloading a file from the returned peer. See: indexRegister(), initializeLogin(), fileLookup(), sendData()

PeerClientInterface.java
- Interface for the PeerClient.

StartPeerClient.java
- Initializes the RMI connection to the server (IndexServer). The server and client URLs are defined outside the Naming.lookup function to support a more flexible method.

PeerFunction.java
- Sends a requested file to the requesting peer as bytes. See: login()

PeerFunctionInterface.java
- Interface for PeerFunction

The main functions of the program can be categorized into Index Server Registration/update, File Download, and File Search. Multiple different methods all talking to one another is required in order to accomplish these tasks. Below are the methods included in the program divided into the respective program tasks.

Index Server Registration / Update:

indexRegister()

- Updates the files on the IndexServer by calling indexUpdate on the peer's file directory.

indexUpdate()

- Updates the index with the peer and their files. If the peer has already been added then the method removes them and re-adds. This method has the opportunity for improvement.

peerArray()

- A void method that posses the list of peers and files. Used for updating the index and adding new peers to the index.

File Search

fileLookup()

- Looks up the requested file using the IndexServer method searchIndex() and returns all the peers with the file. Prompts the user to select which peer to use.

searchIndex()

- Looks in the register for any files matching the requested file and then returns the name of the peer.

File Download

initializeLogin()

- Calls the method login from peerFunction that converts the file into bytes and sends to the requesting peer.

Login()

- As stated above, converts a requested file into bytes and sends to the requesting peer. Utilizes the PeerClient method sendData().

sendData()

- Creates a new file and downloads the contents of a requested file into the newly created file.

Design Choices and Future Improvements

Java RMI was used because the author is more fluent in Java programming. Using Java requires the use of RMI (Remote Method Invocation) because Java does not support pointers.

The structure of the program was born from an online tutorial listed in the references that details the mechanics behind sending files using Java RMI.

The first build used a HashMap to store the peer data but certain methods were difficult to implement using standard HashMap manipulation. Using a Hashmap instead of an array however could yield a more efficient program.

Future improvements include a more clear user interface, implementing HashMaps instead of Arrays, allow for the downloading of larger files.and overall cleaning the codebase.

References:

[1] https://blogs.dxc.technology/2016/09/06/decentralization-the-napster-metallica-connection/

http://www.ejbtutorial.com/java-rmi/how-to-transfer-or-copy-a-file-between-computers-using-java-rmi

https://docs.oracle.com/javase/tutorial/essential/io/notification.html

https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/hello/hello-world.html