Design Document

A Napster-style Peer-to-Peer File Sharing System

Vivek Bajpai
CWID – A20361204

Vaibhav Uday Hongal
CWID – A20378220

CS550 Spring 2017

Table of Contents

Introduction

Peer-to-peer (P2P) file system is the distribution and sharing of digital media using peer-to-peer networking technology. P2P sharing allows users to access media such as books, music etc. from other connected computers on the network to locate the content required.

In this project, we have tried to implement a simple Napster style P2P file sharing system. Napster was introduced in 1999 as a music-focused online service that concentrated on sharing digital audio in MP3 format.

There are 2 main components in this system:

1. Indexing Server: Accepts the connection from a client and stores the information about the shared file system i.e. indexes the users and their shared content.
2. Peer: Acts as a client when registering with indexing server and as a server when sharing the files with another peer.

Process:

1. Indexing server provides a registry and a lookup interface using which client can connect to server and register itself. It also provides the information about the peers holding the requested file from a client.
2. Client initially connects to the indexing server and registers its peer id and shared file system information. Once done, client can then search for any required file and get the information about the other peers that have the file.
3. Requester client then connects with the peer that has the file and downloads the file from it.

Language and Platform

Language used: Java programming with Remote Method Invocation (RMI)

Platform: Mac OS X (Platform independent because of Java)

Design Details

We have implemented the Napster style P2P using Java Remote Method Invocation (Java RMI) API. We have used RMI over Sockets, RPCs or threads because of the following reasons:

- RMI hides much of network specific code unlike sockets
- RMI is object-oriented and hence provides code reuse
- Facilitates parallel programming as it is multi-threaded
- It is easy to write/easy to use
- RMI is part of Java's "Write Once, Run Anywhere" approach. Any RMI based system is 100% portable to any Java Virtual Machine

Source code details:

Server side:

public interface ServerInterface{} : Defines the interface that is used by the client and implemented by the server.

public class ServerInterfaceImpl{} : The ServerInterfaceImpl class implements an indexing server functionalities in P2P Napster-style File transferring application, to which the clients connect and register their filenames from directory. It provides a lookup module where in a client can search filenames, and gets in return the list of peer ids of the peers which have the file. It also provides a delete mechanism using which clients can delete the files in their shared folder and indexing server is notified of the same.

public class IndexingServer{} : The IndexingServer program declares itself as a server and provides a server for RMI invocation by the clients.

Client side:

public class FileManagerInteface{}: This interface contains the retrieve(String fileName) method which is used by the client part of the peer to retrieve a file from another peer.

public class FileManagerImpl{}: This class is an implementation of the above interface and implements the logic of the retrieve(String fileName). It opens the file and returns the buffer containing the contents of the file to be sent over network.

public class Peer{}: This class acts as the main class for a peer which performs following functions:

1. It starts the DirectoryScanner scheduled job. It does the job of registering or unregistering the files as they are added/deleted from the shared directory.
2. Interacts with the user and performs a lookup with index server to get the peers hosting a file.
3. Retrieve (download) a file from another peer sent by the index server.

public class DirectoryScanner{}: As mentioned above, this class is a scheduled Task which runs every 10 seconds. It keep a cached list of all files present in the shared directory and register/unregister the files with index server as needed.

public class PerformanceTest{}: This class is used for performance testing the system. It sends 1000 continuous requests and calculate the total time taken for servicing all the requests.

config.properties: This file includes the peer id and absolute path of the shared directory for the respective peers along with server IP address.

Improvements and Extensions

Extensions to our application:

- Include algorithms to automatically select the best peer-server based on its service history
- Fit an improved search algorithm to search for files more efficiently and accurately

However,

Napster style P2P application is dependent on a central server. If the indexing server goes down, then the entire network is malfunctioned as the clients won't be able to get the updated information about the files. A better option would be to opt for a decentralized P2P network such as Gnutella, Kazaa, Freenet etc.