Design Document

A Simple Gnutella-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 a variety of media from other connected computers on the network to find the content required and download it to the local storage.

In this project, we have tried to implement a simple Gnutella style P2P file sharing system. Gnutella is the first decentralized P2P network, developed in early 2000. The Gnutella network is a fully distributed network unlike Napster. Here every peer acts as a client and as a server i.e. every client is a server and vice versa. To share the files on a Gnutella network, a user starts with a networked computer that runs the Gnutella client. Since the node will work both as a server and a client, it is generally referred to as "SERVENT" (SERVer and cliENT).

Process:

1. Each peer acts both as a client and as a server and maintains a list of its neighbor peers.
2. As a server, it provides a registry and a lookup interface using which another peer can connect to server and as a client, it searches for a file and sends the download request if files are available.
3. Once the network is initialized, the client searches for a file by using a *query*. This *query* will be sent to all the neighbors and each neighbor looks up for the file using local index and responds with *queryhit* message.
4. Requester client then connects with the peer server that has the file using *obtain* method and downloads the file from it.

In this project, we tested the above process for 2 different topologies – star and linear. This is achieved by changing the neighbors of the peers accordingly. The average response times are observed. This is also repeated for different system load i.e. using only one client, using 2 clients, 3 clients and so on.

To make sure duplicate query messages are not processed again and again, we maintain an assertive array which keeps record of last 50 messages processed by any peer, using their message id. Before any query is process, it is tallied against this array to make sure it is not a duplicate.

The testing was performed using 10 peers all running on separate JVMs on separate machines. We have used the t2.micro instance available in Amazon EC2 to get the 10 machines required for testing.

Language and Platform

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

Platform: Mac OS X (Platform independent because of Java). Amazon Linux was used for performance testing.

Design Details

We have implemented the Gnutella style P2P using 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 concurrency as it is multi-threaded by default
- 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:

Peer:

public class FileManagerInteface{}:

This interface acts as an endpoint to which all RMI requests are sent. The interface supports following two methods.

List<String> searchForFile(QueryMessage qmsg)

This method is called while searching for a file in the entire p2p network. The method recursively calls the same method on all its neighbors using RMI. The name of the file is passed inside the qmsg object along with msg_id, ttl etc.

byte[] retrieve(String fileName)

This method simply downloads a file from a remote peer using RMI and put it into the shared directory on the local machine.

public class FileManagerImpl{}:

The FileManagerImpl implements the methods present in *FileManagerInteface*.

public class Peer{}:

This class contains the main method for the peer and is responsible for user interaction. It asks the user for the file name which needs to be downloaded. After receiving the list of available hosts, the user is asked for the host from which the file should be downloaded. Once user enters its choice, the file is downloaded.

Public class QueryMessage:

This class acts as a wrapper for the query message being sent to the system for a file. It contains the filename to be downloaded along with message_id and TimeToLive.

public class PerformanceTest{}:

This class is used to send continuously 200 request for performance testing of the system. Nothing fancy here.

config.properties: This file includes the peer id and absolute path of the shared directory for the respective peers along the IP addresses on the neighbors. Depending of the list of neighbors for each peer, the topology is decided.

Improvements and Extensions

Extensions to our application:

- Include algorithms to automatically select the best peer-server based on its service history and closeness to the client peer.
- Support for automatic replication for files which are only present in a single peer. This way the availability for the file can be improved.
- Fit an improved search algorithm to search for files more efficiently and accurately