CS550:02

Programming Assignment 1

29 January 2017

Christopher Hannon & Neil Getty

Program Design

Background

The task for this assignment was to design a peer-to-peer file sharing program similar to Napster. Specifically, a network of peers that share files and a central indexing server for finding peers with the right file. In such a program concurrency is important for both the server and the peers. Peers should be able to upload/download multiple files and the central server should be able to handle many requests at once.

This program was coded using Java, relying primarily upon the Java Remote Method Invocation (RMI) API. RMI is multi-threaded, meaning that each request on a remote object are handled in a new Java thread for concurrent processing.

Design

The program consists of two main packages: host and peer.

The *host* package defines the remote interface and implementation for the central indexing server.

Interface class *IndexInt* extends the RMI remote interface and defines the methods register, deregister and lookup. Register and deregister allow a client to modify the indexing map keying files to particular peers that have these files. Lookup returns the list of peers that have a particular file. The *ServerImpl* class implements *IndexInt* and must be run to register the remote object with the registry.

The *peer* package contains 4 classes and 1 interface.

Interface Class *PeerInt* extends the RMI remote interface and defines the method retrieve. *PeerImpl* implements *PeerInt*, each client must reference a new *PeerImpl* object to register with the RMI registry and specify the correct file directory. The retrieve method returns a byte array representation of the local file.

Client defines all the methods for a client to interact with the indexing server. When a client is started it registers all the files in a given folder with the server, and any subsequent changes to this folder using the *WatchDir* process. When a user defines a file they want to retrieve *Client* uses the lookup method to get the list of peers with the file and then uses the retrieve method to query a remote peer for the file.

ClientDriver defines the main method for the client and relies on two arguments, the id (IP address) of the client and the directory of files to register and download to. Until the application is closed, *ClientDriver* will query the user for filenames to download.

WatchDir is a separate process for monitoring the download location directory of a client and issuing registration changes with the index server.

Evaluation

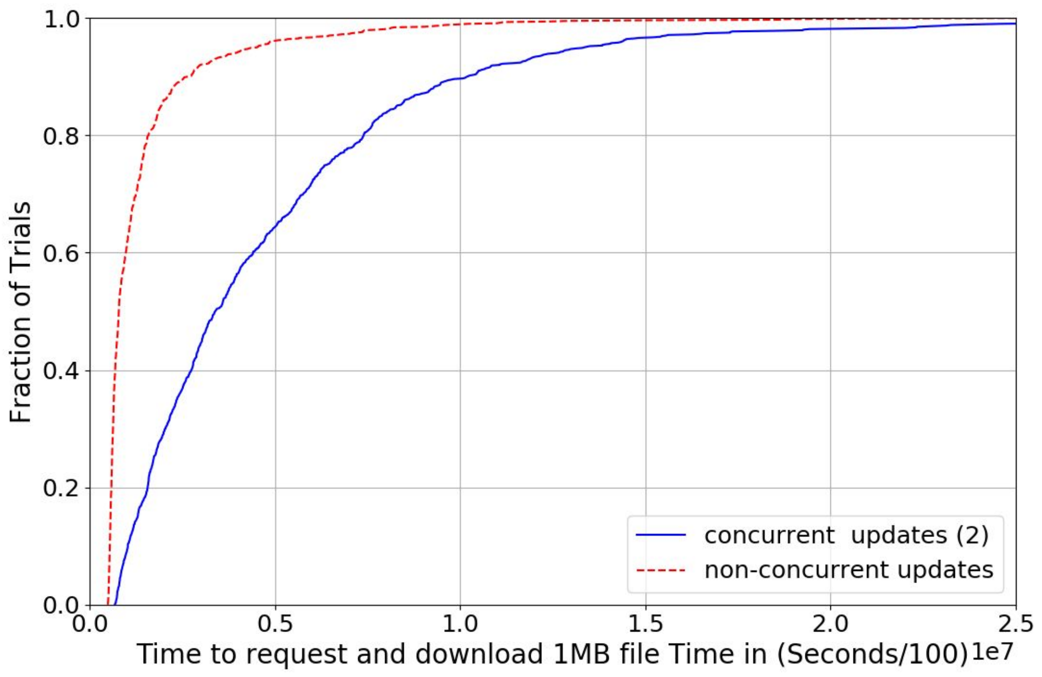
In order to evaluate the performance of the Peer to Peer file sharing system we evaluate the performance of peers calling lookup and retrieving files from peers.

Experiment Setup:

We use Mininet \cite{mininet} the network emulator to test our P2P implementation. Mininet enables us to determine the performance of multiple machines connected together by using a virtualized network and linux namespaces. We create four hosts connected together through a single virtual switch with 10gb emulated links. One host runs the Index Server and the other three hosts act as peers. Each host runs a RMI Registry process, a directory watch service and a driver function. One host has a 100mb file. Another host requests this file from the Index server and downloads it to its local directory. We repeat this process 1000 times to determine the average performance. Because our design enables our platform to support concurrent requests, we also test two hosts requesting the same 100mb file at the same time. Full detailed setup instructions are in the test case document.

Experimental Results:

Our results show that 95% or requests when only one host is requesting the file is under 5 milliseconds while under concurrent requests 95% of trials require 13 milliseconds or less with only 62% of trials under 5 milliseconds. We see that the performance deteriorates with concurrent utilization but it is not directly clear the cause for the greater delay (resource contention, scheduling, network bandwidth, etc.) without further analysis. Since the hosts automatically upload their files using a separate process this will affect the speed at which the processes will retrieve the files.



Caption: Cumulative Distribution of lookup + retrieve time in seconds / 100 of single host requesting files and multiple simultaneously. Trials were repeated 1000 times.

Improvements

There are many possible improvements to this application:

- Graphical user interface (buttons, download progress, list of files, saveable settings)
  Easily made using an IDE or libraries such as Java Swing.

- Maintaining and updating a client side searchable list of all files across all peers
  Every time the client starts the application a local file may be updated with a copy of the files keyed on the index server. This can be easily sorted and searched using partial or whole queries.

- File validity/integrity verification
  In the current version there is nothing stopping a peer from registering a different file under the same name as another. The most easy check would be file type, size or metadata. The most secure method would be to use a hash cheksum such as MD5 or SHA. This can be implemented by maintaining checksums in the central server and having the client first validate a peer has a valid file before retrieval.

- Download chunks of files from multiple peers concurrently
  As is the entire byte array representation is sent from one peer to another. It is possible to request different chunks of the file from different peers. This would spread the bandwidth across the network and allow for redundancy.

- End-to-end encryption
  It might be useful to first encrypt the file before sending and decrypt upon retrieval.

- Transfer of folders/subfolders
  The current version can only handle files in a single directory. To send all files in a directory and subsequent subfolders the retrieve method must be changed.

Conclusion

In conclusion we have developed a barebones P2P file sharing platform that supports concurrent connections through the use of Java RMI. When files are modified or deleted they are registered and deregistered to the file system automatically using the watch directory process. Peers can request the index server for a list of peers that are candidates to download from and download files directly from them.

Sources

Watch directory for changes:
https://docs.oracle.com/javase/tutorial/essential/io/notification.html
Java RMI
http://www.ejbtutorial.com/java-rmi/how-to-transfer-or-copy-a-file-between-computers-using-java-rmi
https://docs.oracle.com/javase/tutorial/rmi/
http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html
Mininet:
http://mininet.org/