Advanced Operating System Programming Assignment 1 Report

A20491471-Sashank Lakshmana Bommadevara

A20516702 - Abdurakhmon Urazboev

A20519664-Syed Alle Mustafa

Introduction

According to the project requirements, the design, implementation, and evaluation of a peer-to-peer (P2P) file sharing system are detailed in this report. A central indexing server and several peer nodes were the components of a simplistic P2P system that was the aim of this project. All peer content that has been registered is indexed by the central indexing server, which also offers search functionality. Users can browse for files and download them from other peers due to each peer's functioning as client and server. Java programming languages were used to implement the system, which was tested in an environment running Ubuntu Linux 22.04 LTS.

System Design

Components

The P2P system consists of two fundamental parts:

1. Central Indexing Server: This server is in charge of indexing the metadata of files stored on registered peers and offering search capability. It displays the following interface: –registry(peer id, file name,...) - Enables peers to register their files with the indexing server, which creates an index for each peer.
   –search(file name) - Searches the index and displays a list of matching peers.
2. Peer: Every peer functions as a server and a client at the same time. Users can search for files using the indexing server as a client, and the server will then return a list of peers that also have the desired file. The peer serves as a server by anticipating requests from other peers and sending the requested files. The following interface is made available by the peer:
   Files can be downloaded from peers using the obtain(file name) command.

Implementation:

To implement the architecture of the system java is used on ubuntu linux 22.04. We have two main components of the system, the first one is the server containing file indexes, where the other is the client or the peers. Let's start with the server implementation first.

The Server:

The server has a main java file, which has an entry function, called “Start()”, in this function the socket for incoming requests is initialized on port 9876. After initializing the socket it calls a function called waitforconnection(), this function runs on an endless while loop. This function accepts a new incoming request, and assigns the new socket upon handshake to a new thread. This thread is of a class of which is an implementation of Runnable interface called ServerThread, this class overrides the run function, in this function, we first check whether the client

wants to register its files, or wants to search a file. If a client wants to register its files, the register keyword is followed by the filename, then the file is added to the hashmap with the client ip for indexing. If the client wants to search for a file, we search the hashmap with the filename, and return the corresponding client information. The rest of the work is done by the peer containing the file.

The Peer:

The peer has two main duties, One is to connect with the server and register all its files, where the other one is to listen for the other clients which want the files owned by it. The client, whenever initialized, registers all the files it has on the Files folder with the server. After registering, the server prompts it if it wants to search and download any file, and upon asking for the file it gets downloaded on the peer in a function called obtain(), which uses getInputStream of the socket class to download the file. All of this process is running on the main thread of the client. The client also starts a thread to listen to the requests. This thread runs on an infinite loop for listening to the requests, and whenever a new request is received, it runs a new thread to upload the given file using the getOutputStream function of the socket. It does it with a buffer size of 4096 bytes. Whenever the uploading of the file is complete, the client's thread closes that socket. And kills the thread.

Experiment Setup:

For evaluation, we deployed the following configuration:

- 2 peers
- 1 indexing server
- 3 virtual machines (VMs)

Each peer's shared directory contained the following datasets:

- 1M: 1KB text files
- 1K: 1MB text files

- 10: 1GB binary files

All files had unique names on each node, and replicated data among peers was placed in the same directory as the original data.

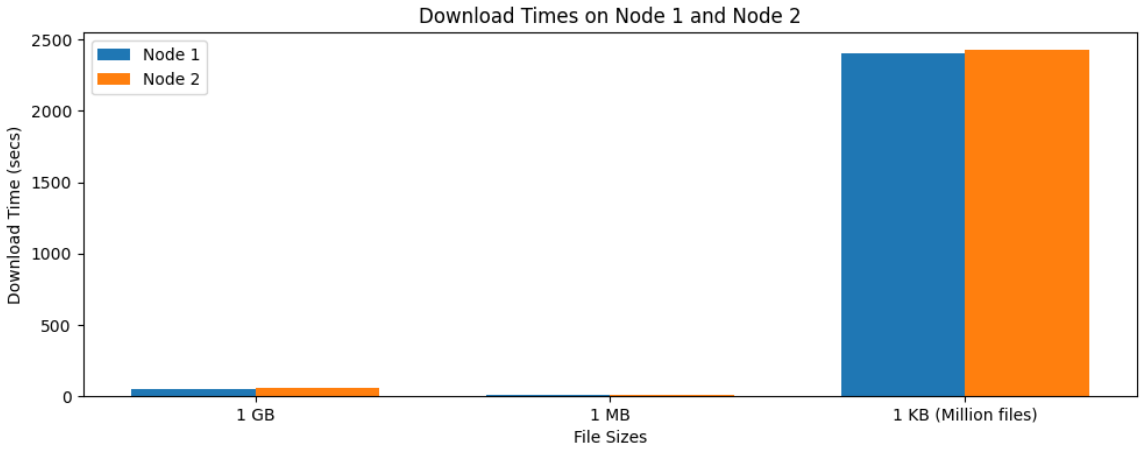
Experiment Results:

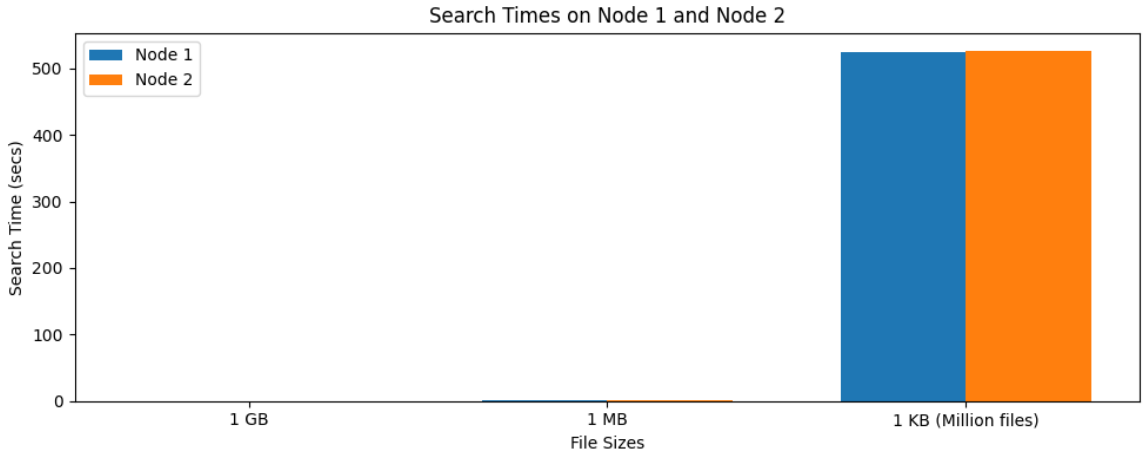
With a focus on search and download times, we ran tests to evaluate the effectiveness of our P2P file sharing system. We measured the time required for these processes using timing code and gathered data for analysis.

Download Times in secs:
10 1GB bin files : node 1 : 53.75secs , node 2: 56.23secs
1000 1Mb files : node 1: 11.13secs, node 2 : 12.16secs
1M 1Kb files : node 1: 2403.28secs, node 2 : 2427.13 secs

Search Times in secs:
10 1GB bin files : node 1: 0.009secs , node 2: 0.0012secs
1000 1MB files : node 1 : 0.78secs, node 2 : 0.79secs
1M 1KB text files : node 1 : 525.21secs, node 2: 526.87secs

```
Download Times Average (Node 1): 822.72 secs, SD: 1117.76 secs
Download Times Average (Node 2): 831.84 secs, SD: 1128.18 secs
Search Times Average (Node 1): 175.3330 secs, SD: 247.4006 secs
Search Times Average (Node 2): 175.8871 secs, SD: 248.1826 secs
```





Observations:

- Scalability up to 2 Nodes: The system demonstrates respectable scalability while increasing from 1 to 2 nodes. Even with the second node, the increase in search and transfer durations is not appreciably longer. This shows that the P2P centralized system may efficiently manage a small network with two nodes.
- File Size Impact: It seems that as file sizes grow, so do the search and transfer times. Larger files take longer to look for and transfer, so this is to be expected. The rise in timings for 1GB binary files is, however, noticeably more than for 1MB text files. This shows that the system would not scale

well for really large files, and further optimization may be required for these situations.

- Scenarios for future scaling Scaling to 1K Peers: During peak search periods, the central indexing server would likely experience an increase in load. The system's efficiency and architectural design would have an impact on search times.It would be extremely difficult to manage 1 billion peers. For a network of this size, the current architecture might not be appropriate. To achieve scalability at this level, distributed and decentralized techniques, such as Distributed Hash Tables (DHTs), might be necessary.

Conclusion

With a central indexing server and peer nodes, we have successfully developed, implemented, and assessed a straightforward P2P file sharing system. As a result of the system's effective search and download capabilities, peers can use it to share files.

The analyses provided insights into the system's performance characteristics, and the studies included measuring search and download times. This project gave me useful insight into the structure and workings of a P2P file sharing system as well as experience working with sockets, processes, threads, and makefiles.

The End