

AOS - CS550
Programming Assignment-2
A Simple Decentralized Peer to Peer File Sharing System

TEAM:

- ❖ Vaishnavi Manjunath - A20446043
- ❖ Kavya Ravella - A20444960
- ❖ Kalpana Pratapaneni – A20448916

INTRODUCTION:

Gnutella is a decentralized peer-to-peer system. The users can share resources from their system that others can see and get. They will have access to resources shared by others on the network.

In this assignment, we implement a Gnutella peer-to-peer file sharing system. Each peer will act both as client and server. We have implemented this using java socket programming and multi-threading concepts. When we start the peer, the respective server thread is started which runs in the background. The configuration file is read to find its peer and starts the client thread for each neighbor. When the neighboring peer gets online, the connection will be established. This connection will be used to search among the network.

Server thread is used to search the file within the local directory. The result will be stored into a list. The server thread reads the config file to start the client thread to all its neighbors. Along with the file name, TTL (time-to-live) is attached. TTL prevents query messages from being forwarded many times. Server thread performs the downloading of the file from the peer which is selected by the user.

SYSTEM REQUIREMENTS:

- JDK and Java to be installed
- Server to execute the program on multiple systems

DESIGN:

Entire project is designed using Java where we have used the concepts of Socket Programming and Multi-threading. For establishing the connections between the Server and the Peers, we have used TCP/IP protocol using the sockets.

Major Components of the Project:

- Peer

How it works:

The peer acts as both client and server. We have a config file which contains the peer ID, IP address, port number and neighboring peers.

The server thread is created using this peer ID which will run the client thread in the background poll with the port number that is specified in our config file.

The peer starts the client thread for the neighbors by reading the config file and creates a list with the list of peers and their shared files.

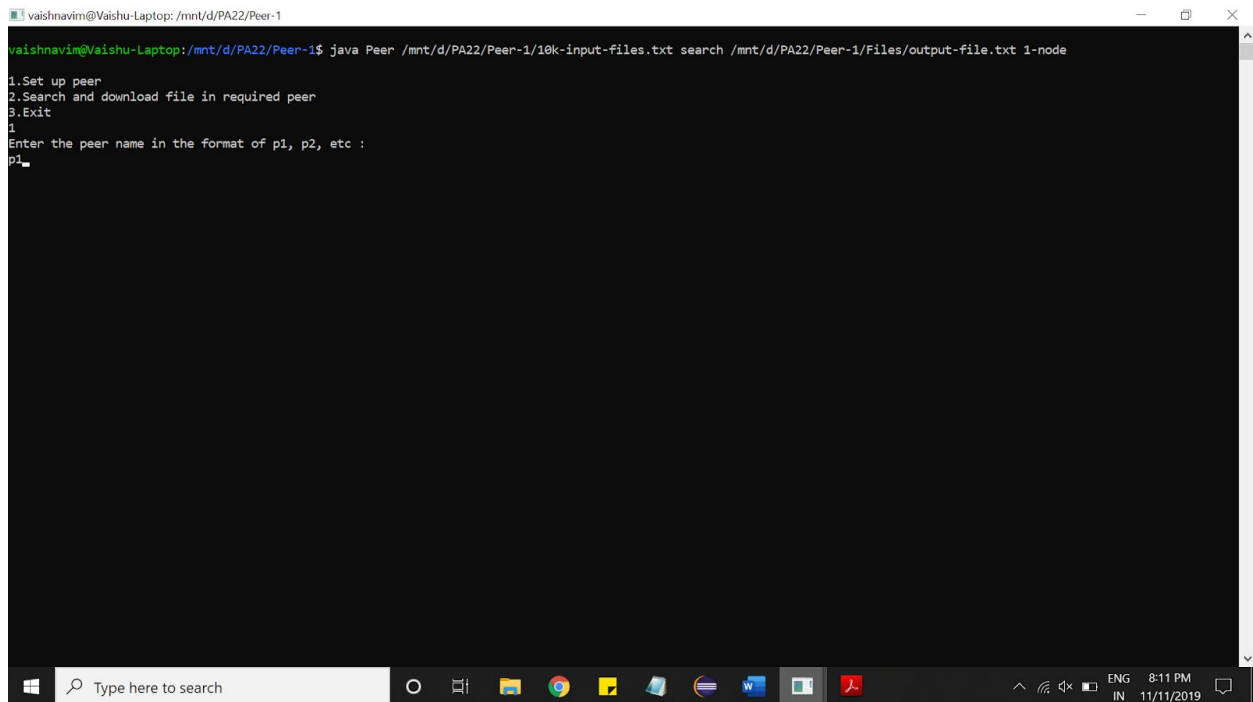
The user will be searching for the required file with the help of filename and TTL. The local shared directory is searched for the requested file and request is forwarded to its neighbors till TTL value reaches 0.

The neighbor returns the peer id if it contains the requested file. List of peers containing the file will be displayed to the user.

The user selects the peer from which the file has to be downloaded. Once the peer id is selected, connection is established between them.

The options provided to our user:

1. Set up peer
2. Search and download file in required peer
3. Exit



```
vaishnavim@Vaishu-Laptop: /mnt/d/PA22/Peer-1
vaishnavim@Vaishu-Laptop:/mnt/d/PA22/Peer-1$ java Peer /mnt/d/PA22/Peer-1/10k-input-files.txt search /mnt/d/PA22/Peer-1/Files/output-file.txt 1-node
1.Set up peer
2.Search and download file in required peer
3.Exit
1
Enter the peer name in the format of p1, p2, etc :
p1
```

```
vaishnavim@Vaishu-Laptop: /mnt/d/PA22/Peer-2
vaishnavim@Vaishu-Laptop: /mnt/d/PA22/Peer-2$ java Peer
1.Set up peer
2.Search and download file in required peer
3.Exit
1
Enter the peer name in the format of p1, p2, etc :
p2_
```

The connection to multiple peers is done in a similar way.

TRADE OFFS:

Gnutella peer-to-peer model is platform independent. The client and server will form a connection regardless of the platform that is implemented.

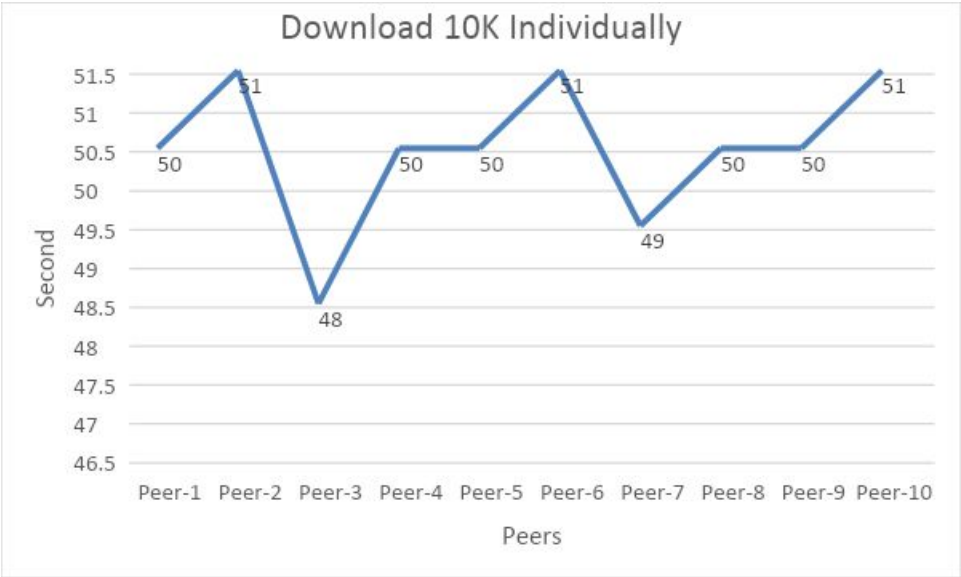
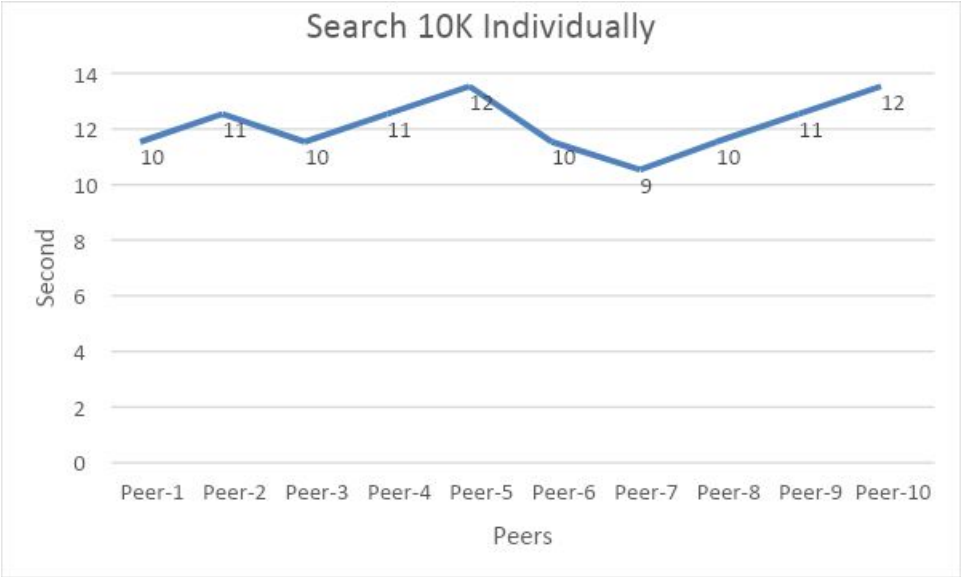
In usage when the client connects with the server, the server will continuously poll for request from the client leading to low performance. Hence there will be a delay due to extensive use of the threads as there are thousands of peers connected.

POSSIBLE IMPROVEMENTS:

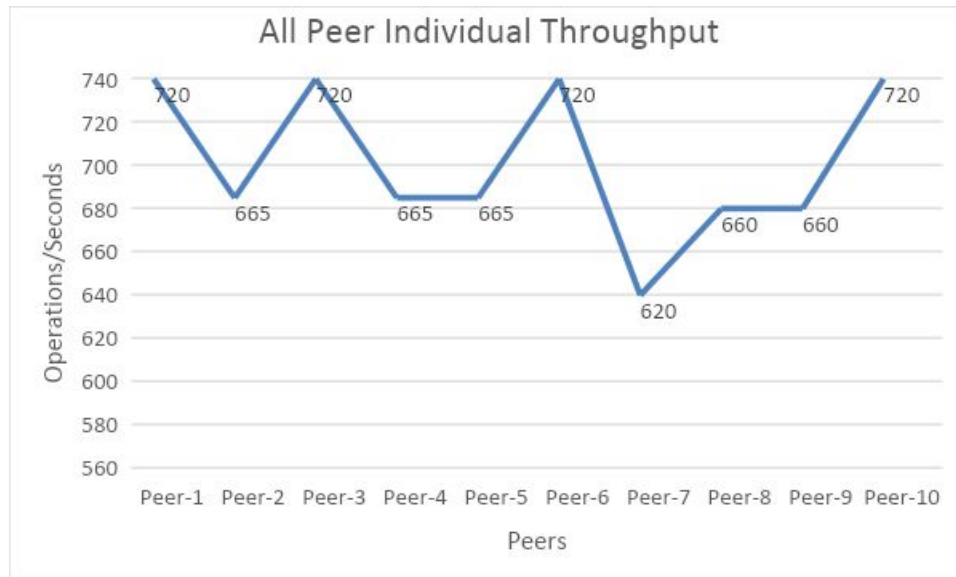
- We can improve the performance using the Avro or protocol buffers for serializing and transmitting data.
- Could develop a User Interface.

Performance Evaluation

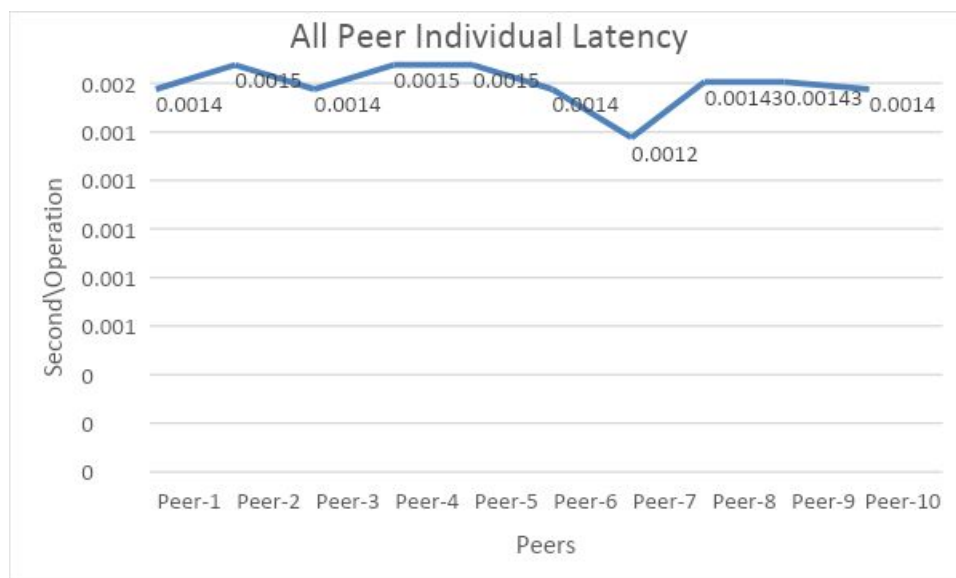
1. 10k operations from all 8 peers individually.



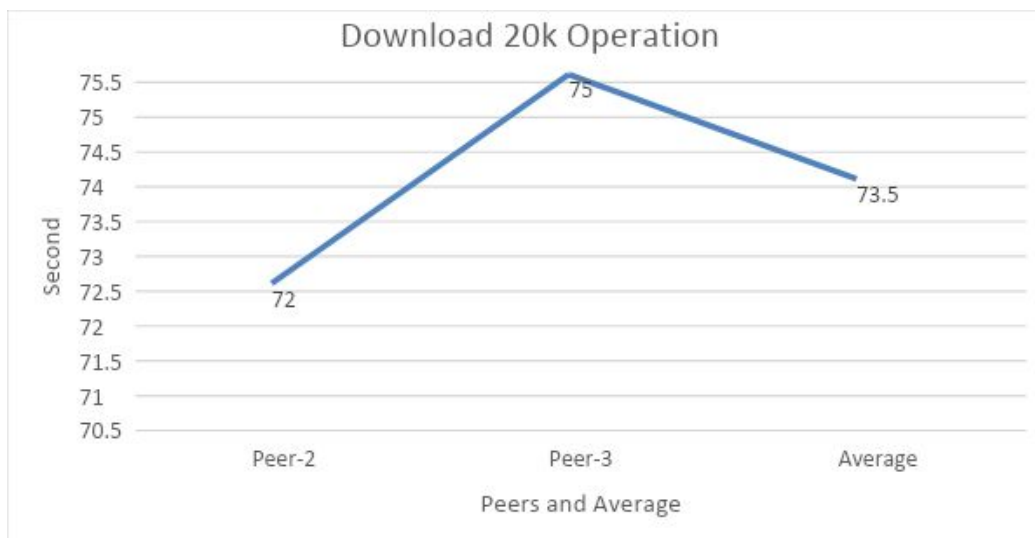
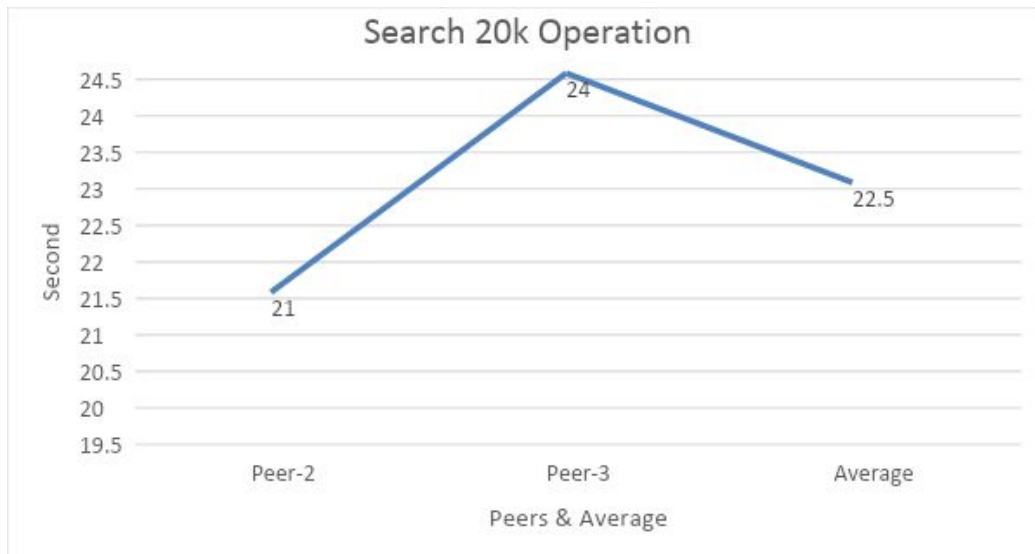
2. Throughput for individual all peer



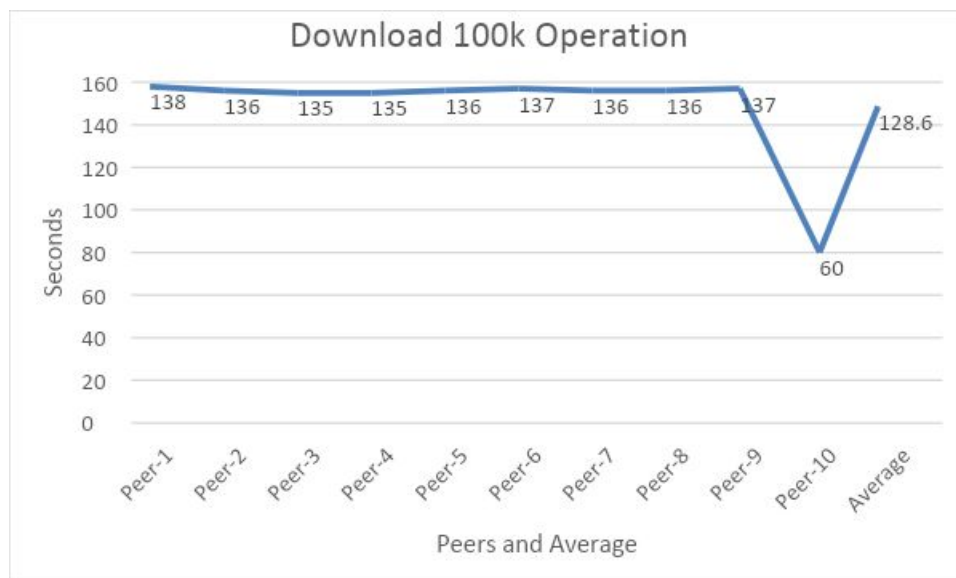
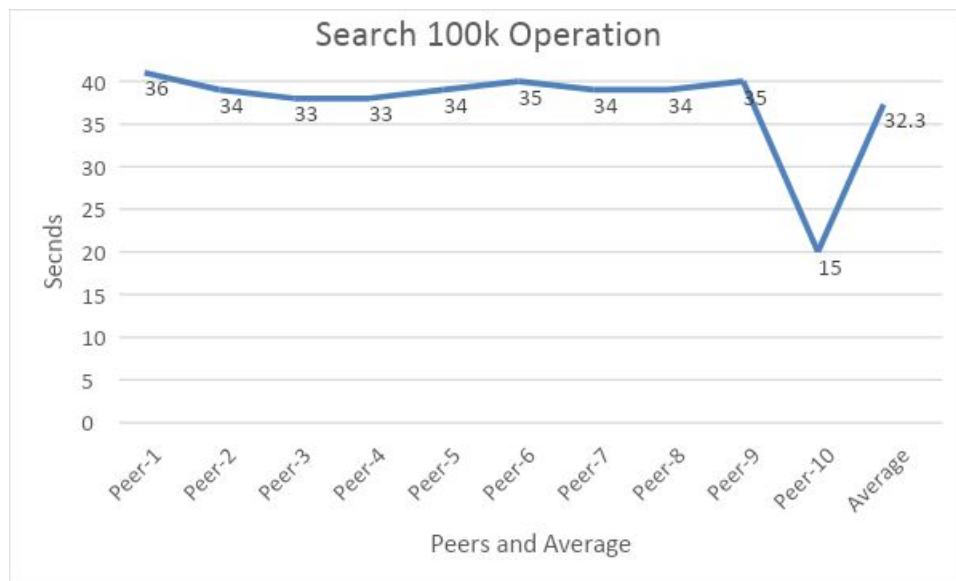
3. Latency for individual all peer



4. Peer1 serving 20K request concurrently from Peer2 and Peer3

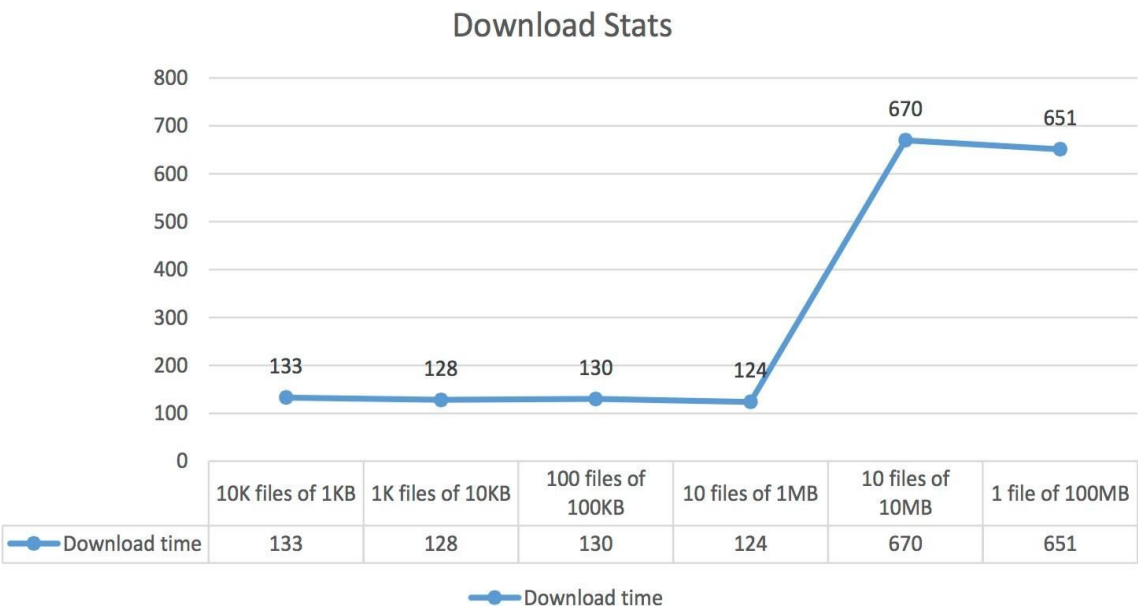


5. Peer10 serving 100K operation from Peer 1 to 9 and also requesting 10K operation, in total 1000K operations concurrently



6. Download analysis for different file sizes.

7.



7. Comparison of centralized and decentralized system for 10K operations from all 8 peers individually

