CS 550 Programming Assignment #1 Report

Haoran Wang (hwang219@hawk.iit.edu)
Bingxin Xu (bxu21@hawk.iit.edu)
Gengyu Zhang (gzhang32@hawk.iit.edu)

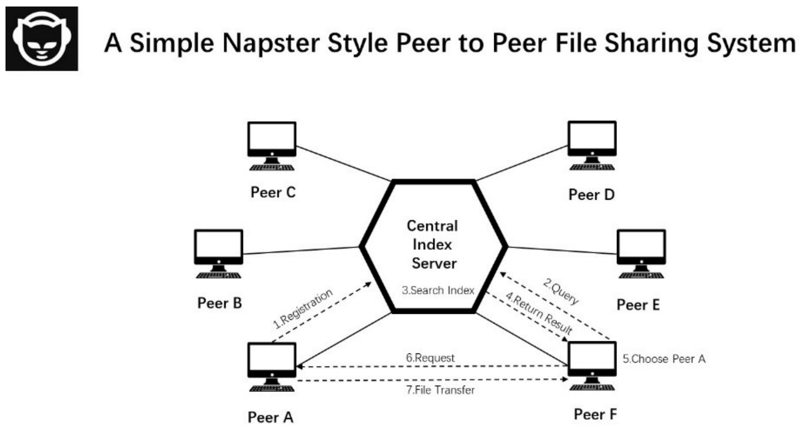
1.Goal and Requirements

1.1 Goal

- Familiarize with sockets, processes, threads, makefiles
- Learn the design and internals of a Napster-style peer-to-peer (P2P) file sharing system

1.2 Requirements

According to the outline requirements to realize the function diagram as follows:



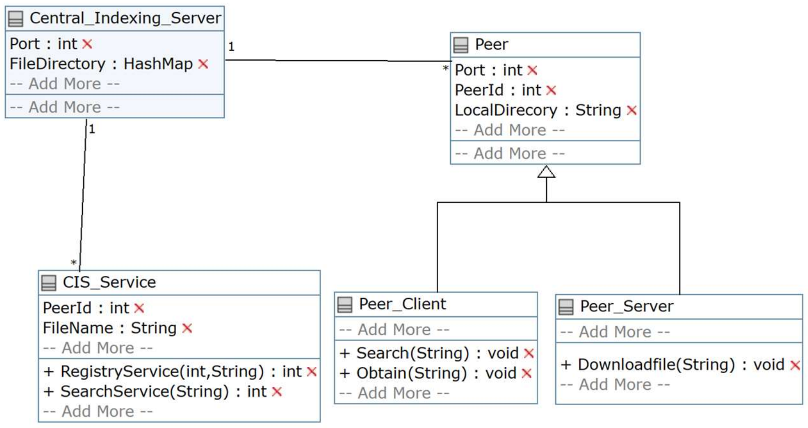
In addition to the above, the following features are included:

- Both the indexing server and a peer server should be able to accept multiple client requests at the same time
- Support any type of files (e.g. text, binary, etc.).

- Support for data resilience by allowing a configurable replication factor (system wide).

2.Solution Design

The assignment 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 Clients, we have used TCP/IP protocol using the sockets.
Based on the above requirements, we design the class diagram as follows:



Specifically, the following Java classes are included:

- **Central_Indexing_Server Class:** implements the Indexing Server startup, listen on the service port and create work thread
- **cisServices Class:** implements all the central indexing server functionality, it runs on the thread. This class provide the following interface to the peer clients:

  - registry (String peerId, String fileName) - invoked by a peer to register all its files with the indexing server

  - search(String fileName) - search the index and return all the matching peers to the requestor
- **Peer Class:** implements the peer client and peer server startup
- **peerClient Class:** implements peer as client functionality, the user specifies a file name with the indexing server using "lookup". This class provide the following interface :

  - runInteractive() - run client functionality interactively

  - runBatch(String level) - run client functionality in batch, use for evaluation and measurement the behavior of our system

  - obtain(String fileName, String ownerId, String downPath) - invoked by a peer to download a file from another peer
- **peerServer Class:** the peer waits for requests from other peers and sends the requested file when receiving a request

- **getParameter Class:** used to read parameters for system configurability

The above Java class is contained in the following file:
1. **Central_Indexing_Server.java** contains Central_Indexing_Server Class and cisServices Class
2. **Peer.java** contains Peer Class, peerClient Class and peerServer Class
3. **getParameter.java** contain getParameter Class

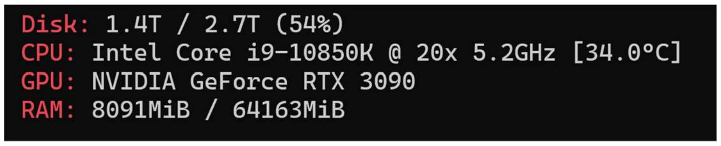
In addition to the main functional classes mentioned above, the following Supporting programs and tools are used:

- **Create_DataSet.sh:** generate test datasets for each peer automatically
- **config.properties:** Configuration files for each node, including Ip, port and shared file directory etc.
- **Makefile:** script used for automating the build process of the project
- **data_plot.py:** plot performance evaluation data in figures graphically
- **pssh:** coordinate the bootstrapping of our P2P system, and automate and conduct the performance evaluation concurrently across the P2P system.
- **node-hosts:** hosts file from which pssh read hosts names. Each line in the host file are of the form [user@]host[:port] and can include blank lines and comments lines beginning with "#".

3.Development and Deployment

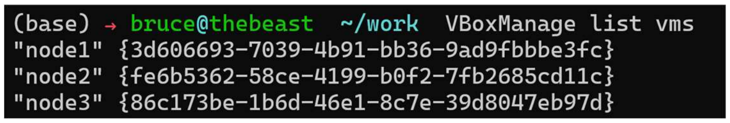
3.1 Hardware

Host:



```
Disk: 1.4T / 2.7T (54%)
CPU: Intel Core i9-10850K @ 20x 5.2GHz [34.0°C]
GPU: NVIDIA GeForce RTX 3090
RAM: 8091MiB / 64163MiB
```

Guest VM:



```
(base) → bruce@thebeast  ~/work  VBoxManage list vms
"node1" {3d606693-7039-4b91-bb36-9ad9fbbbe3fc}
"node2" {fe6b5362-58ce-4199-b0f2-7fb2685cd11c}
"node3" {86c173be-1b6d-46e1-8c7e-39d8047eb97d}
```

3.2 Software

os: Ubuntu 22.04 jammy

jdk: openjdk version "11.0.20.1" 2023-08-24

shell: bash 5.1.16

pssh: 2.3.4

3.3 Build

For compilation:

```
make
```

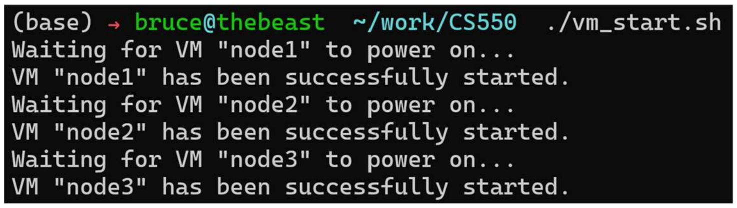
For clean up the compilation environment:

```
make clean
```

3.4 Deployment

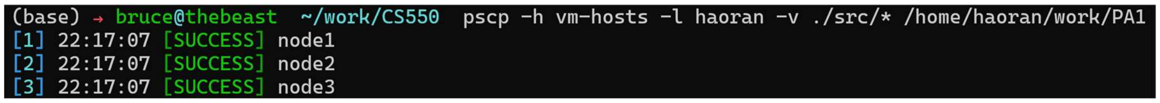
(1) Deploy 2 peers and 1 indexing server over 3 VMs.

(2) Startup VM:



```
(base) → bruce@thebeast  ~/work/CS550  ./vm_start.sh
Waiting for VM "node1" to power on...
VM "node1" has been successfully started.
Waiting for VM "node2" to power on...
VM "node2" has been successfully started.
Waiting for VM "node3" to power on...
VM "node3" has been successfully started.
```

(3) Put Source Code to VM:



```
(base) → bruce@thebeast  ~/work/CS550  pscp -h vm-hosts -l haoran -v ./src/* /home/haoran/work/PA1
[1] 22:17:07 [SUCCESS] node1
[2] 22:17:07 [SUCCESS] node2
[3] 22:17:07 [SUCCESS] node3
```

(4) On each peer node, run Create_DataSet.sh, using peerid as a script parameter.

```
Create_DataSet.sh 1
```

When the script runs successfully, it generates 10k small files (1KB), 1K medium files (1MB) and 8 large files (1GB) in a directory named shared.

(5) Modify the configuration information for each node in the config.properties file. The parameters are described as follows:

Parameter	Meaning
Peer_Node	this peer node number
File_Node	peer node number other than this node
Central_Indexing_Server_Ip	IP of Idexing Server
Central_Indexing_Server_Port	server port number of Idexing Server
Peer1_Ip	IP of Peer node 1
Peer1_Client_Port	port number of peer 1 as client
Peer1_Server_Port	port number of peer 1 as sever
Peer1_Shared_Directory	share file directory of peer 1
Peer2_Ip	IP of Peer node 2
Peer2_Client_Port	port number of peer 2 as client
Peer2_Server_Port	port number of peer 2 as sever
Peer2_Shared_Directory	share file directory of peer 2

4.Run and Measurement

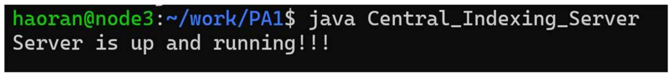
4.1 Run

(1) Startup Indexing Server:

Input:

```
java Central_Indexing_Server
```

Output:



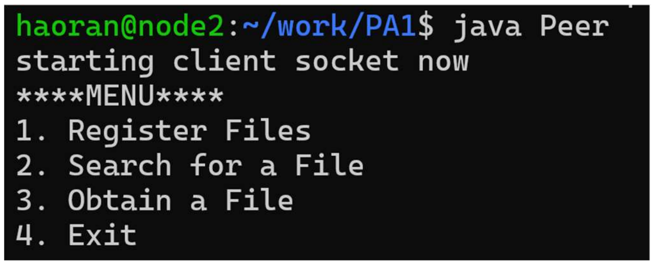
```
haoran@node3:~/work/PA1$ java Central_Indexing_Server
Server is up and running!!!
```

(2) Run peer interactively：

Input:

```
java Peer
```

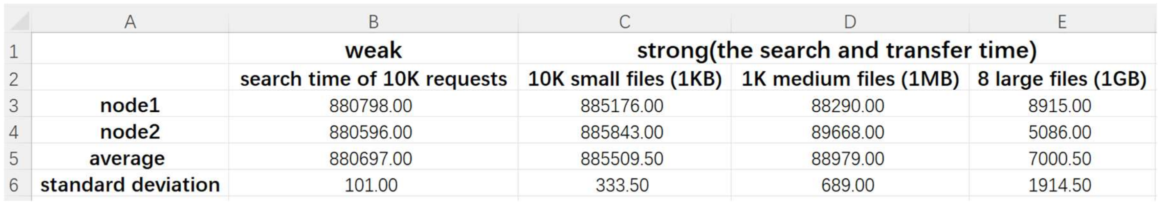
Output:



```
haoran@node2:~/work/PA1$ java Peer
starting client socket now
****MENU****
1. Register Files
2. Search for a File
3. Obtain a File
4. Exit
```

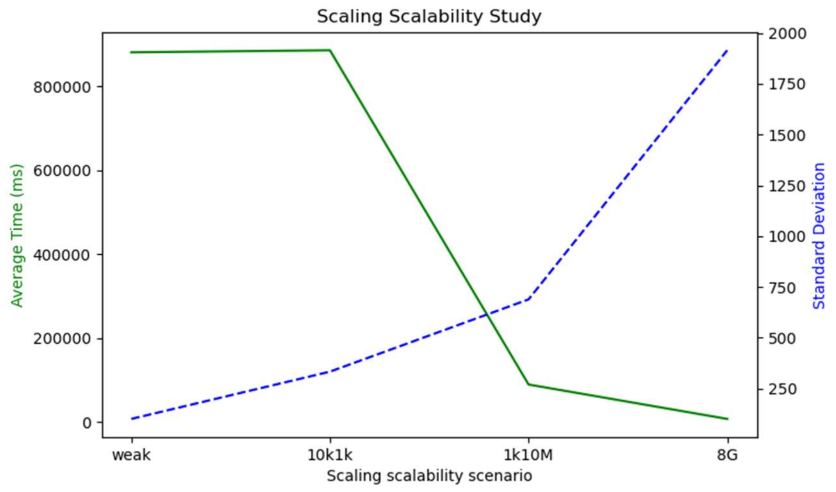
4.2 Measurement

(1) Experimental results statistics:



	A	B	C	D	E
1		weak	strong(the search and transfer time)		
2		search time of 10K requests	10K small files (1KB)	1K medium files (1MB)	8 large files (1GB)
3	node1	880798.00	885176.00	88290.00	8915.00
4	node2	880596.00	885843.00	89668.00	5086.00
5	average	880697.00	885509.50	88979.00	7000.50
6	standard deviation	101.00	333.50	689.00	1914.50

(2) Graphical presentation of experimental results



4.3 Point-to-Point Response

(1) Q: Can you deduce that your P2P centralized system is scalable up to 2 nodes?

A: With 2 nodes, the indexing server can distribute the processing load more effectively. This can lead to faster indexing times and improved query response times. It will also increase the capacity and high availability of the centralized system.

(2) Q: Does it scale well for some file sizes, but not for others?

A: Considering the indexing and querying operations time, the effect is obvious for a large number of small files, but not for a small number of large files.

(3) Q: Based on the data you have so far, what would happen if you had 1K peers with small, medium, and large files?

A: From the results of the above experiments, we can see the following facts: First, a large number of concurrent searches on the CIS performance impact, but for different sizes of files serach time does not change much. Second, transfer takes more time than search, and compared to small size files, big size files have a greater impact on bandwidth, resulting in greater variation in the transfer time of each node.

(4) Q: What would happen if you had 1 billion peers?

A: If central indexing server had 1 billion peers, the system will face many challenges, such as high resource requirements, network congestion, indexing and querying operations latency issues and indexing overhead etc.