CS 550 Programming Assignment #2 Report

Haoran Wang (hwang219@hawk.iit.edu)

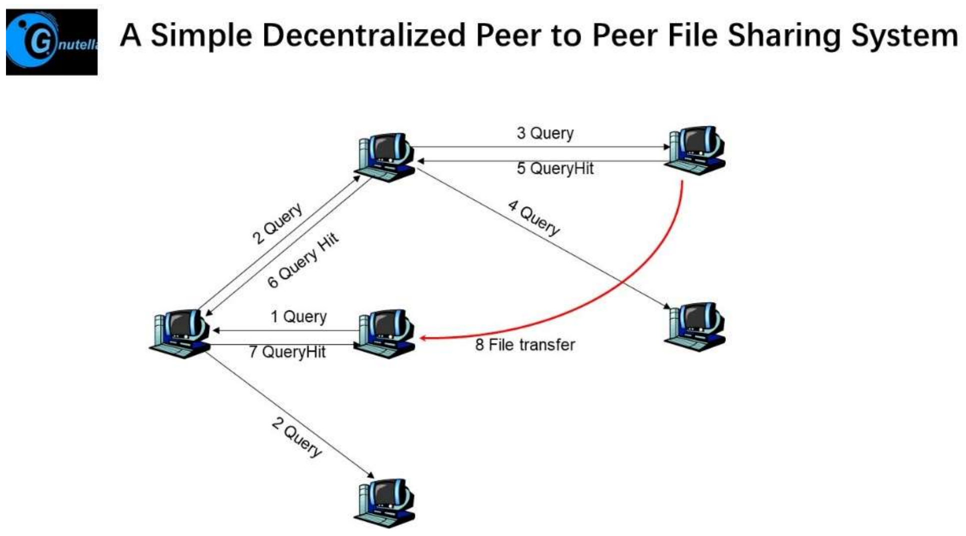
1.Goal and Requirements

1.1 Goal

- Familiarize with sockets, processes, threads, makefiles
- Learn the design and implement a pure distributed file-sharing system

1.2 Requirements

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



Base on PA1 and according to PA2 outline, in addition to the outline requirements, I implement P2P features as following:

- Implement data resilience mechanism with a system wide the replication factor, based on residual algorithm. TA deducted me 10 at PA1 because of this fuction.Support any type of files (e.g. text, binary, etc.).

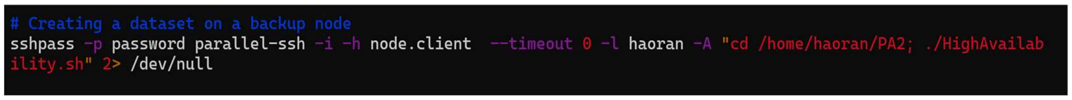
- Implement transfer throughput Small function: 9 clients concurrently issue 10K query+obtain requests each for random files from Small dataset(Bonus 2%).
- My P2P system support any type of files (e.g. text, binary, etc) up to 4GB in size
- Utilizing java multithreading pool technology, build an efficient decentralized P2P system.
- Utilizing shell script , implement development, deployment, testing automation.

1.3 Key Point

- A peer SHOULD forward incoming Query messages to all of its directly connected servents, except the one that delivered the incoming Query.
- A peer receiving a message with the same Message ID as one it has received before, MUST discard the message. It means the message has already been seen.
- A peer that receives a HitQuery message with Message ID = n, but has not seen a Query message with Message ID = n SHOULD remove the QueryHit message.
- The TTL value of a new query created by a peer SHOULD NOT be higher than 7, and MUST NOT be higher than 10. The hops value MUST be set to 0.

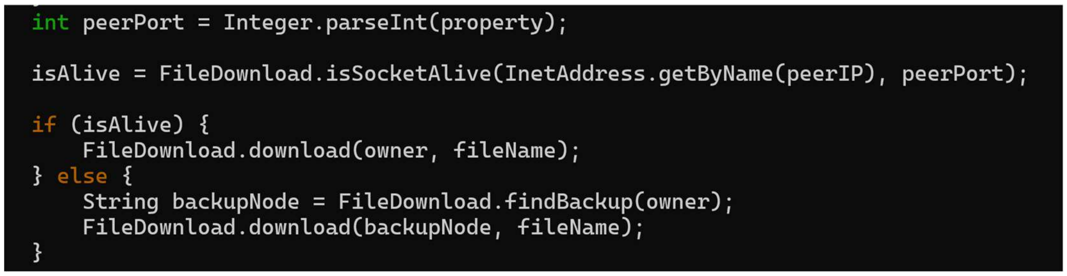
1.4 Data resilience mechanism implementation

1. In system deployment phase, Get_Ready.sh call ./simba/HighAvailability.sh script to establish the one to one backup relation of each node and generate backup data on the corresponding node.



```
# Creating a dataset on a backup node
sshpass -p password parallel-ssh -i -h node.client  --timeout 0 -l haoran -A "cd /home/haoran/PA2; ./HighAvailab
ility.sh" 2> /dev/null
```

2.In obtain method, first check target node is alive, if the node die, get the file from backup node.

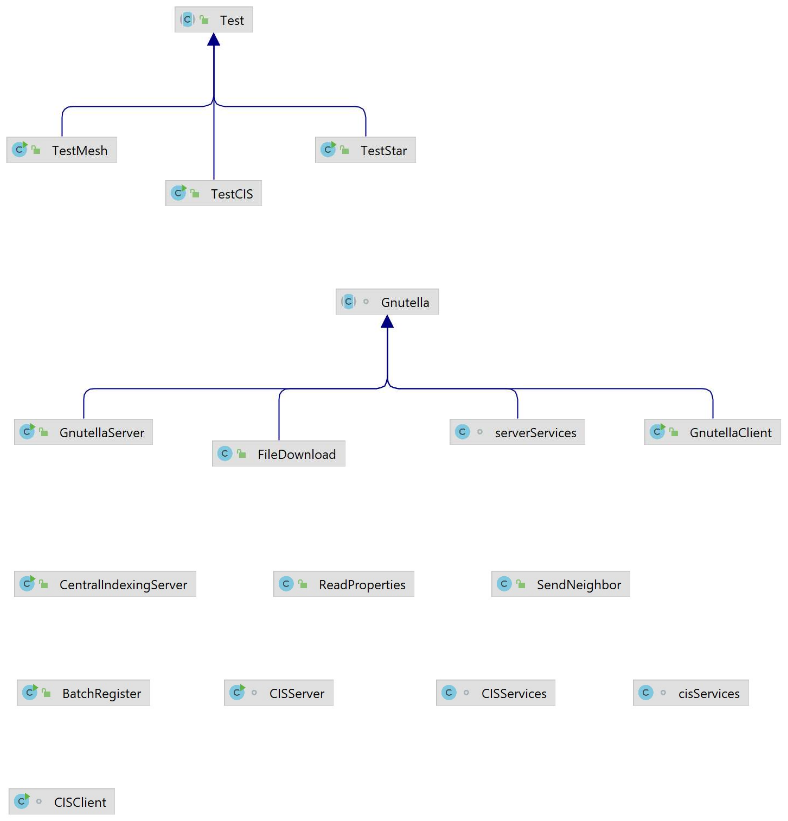


```
int peerPort = Integer.parseInt(property);

isAlive = FileDownload.isSocketAlive(InetAddress.getByName(peerIP), peerPort);

if (isAlive) {
    FileDownload.download(owner, fileName);
} else {
    String backupNode = FileDownload.findBackup(owner);
    FileDownload.download(backupNode, fileName);
}
```

2.Design & Development

2.1 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：



2.2 Development

Deployment of 1 host（simba）and 17 vms（simba01-simba17）on the Chameleon experimental platform. The host is used to manage 17 vms, which are used to run PA2 tasks, where simba17 is used for the server. The file directory and program functions on the host computer are shown below:

```
PA2.(main)
├── Get_Ready.sh(initialize system:copy and compile source code, create datasets)
├── README.md(this file)
├── again.sh(for debug,every time copy and compile source code)
├── node.all(for parallel-ssh : 17 VMs IP)
├── node.client(for parallel-ssh : exclude host simba17, which is Central Indexing Server)
├── node_1.client(for parallel-ssh : one client test)
├── node_1.server(for parallel-ssh : one client test)
├── node_9.client(for parallel-ssh : nine client test)
├── node_9.server(for parallel-ssh : nine client test)
├── p2pBenchmark.sh(benchmark automation using parallel-ssh)
├── simba(source code)
│   ├── CISClient.java(Napster-style peer client class)
│   ├── CISServer.java(Napster-style peer sever class)
│   ├── CentralIndexingServer.java(Napster-style Server)
│   ├── Create_DataSet.sh(create dataset on every node)
│   ├── FileDownload.java(obtain requests)
│   ├── Gnutella.java(Gnutella-style parent class)
│   ├── GnutellaClient.java(Gnutella-style client subclass)
│   ├── GnutellaServer.java(Gnutella-style server subclass)
│   ├── HighAvailability.sh(for data resilienc, backup Data)
│   ├── Makefile(compile automation)
│   ├── ReadProperties.java(read p2p system configuration file)
│   ├── SendNeighbor.java(broadcast message)
│   ├── Test.java((parent class of test case )
│   ├── TestCIS.java(centralized system test case)
│   ├── TestMesh.java(star topology test case)
│   ├── TestStar.java(2D-mesh topology test case)
│   └── config.properties(a system wide configuration file)
├── start-vm.sh(start all vms)
└── stop-vm.sh(stop all vms)
```

2.3 Supporting tools

- **parallel-ssh series**: bootstrapping of P2P system, as well as to automate and conduct the performance evaluation concurrently.
- **Intellij Idea**: remote Development tools, to connect to host on Chameleon with the IDE backend running there from my local machine.
- **Tmux**: a terminal multiplexe, very useful for running multiple programs with a single connection.
- **Screen**: provides the ability to launch and use multiple shell sessions from a single ssh session.

3. Deployment

Deploy 1 host and 17 VMs.
Create first vm, install jdk, parallel-ssh, sshpass etc. on host run lxc copy command to create other vms. after successes use ssh-keygen, ssh-copy-id to build passwordless environment.

```
sudo lxc launch images:ubuntu/22.04 simba01 --vm -c limits.cpu=1 -c limits.memory=6G
B
sudo lxc copy simba01 simba02 --verbose
sudo ssh-keygen -f /home/haoran/.ssh/keys/simba01
sudo ssh-copy-id -i /home/haoran/.ssh/keys/simba01 haoran@simba01
```

When the vms runs successfully, use start-vm.sh, stop-vm.sh script on host to manage vms.Use Get_Ready.sh script to distribute source code, compile, and generates 100k small files (10KB), 10 large files (100MB) and 1 large files (4GB) in a directory named shared. I also use parallel-ssh commands to monitor and kill process:

```
sshpass -p password parallel-ssh -h node.all -l haoran --par 50 -A -i --timeout 0 "l
sof -i:60001"
sshpass -p password parallel-nuke -h node.all -v -l haoran -A java
```

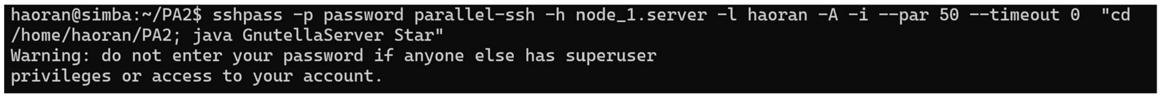
Modify the configuration information on host in the config.properties file(in /PA2/simba) before use Get_Ready.sh distribute to each node . The parameters are described as follows(not the whole, only for example):

Parameter	Meaning
Shared_Directory = shared	share file directory
Small_Files = All_Node.small	use for random sample small dataset
Large_Files = All_Node.large	use for random sample large dataset
Replication_Factor = 1	a system wide replication factor
Star_simba01_Neighbors = simba02...simba16	For star topology
Mesh_simba01_Neighbors = simba02,simba05	For 2D-mesh topology
simba01_IP = 10.108.74.142	Each client node IP
simba01_Server_Port = 60001	Each client server port
Central_Indexing_Node = simba17	central indexing server
CIS_IP = 10.108.74.213	central indexing server IP
CIS_Port = 60001	central indexing server port

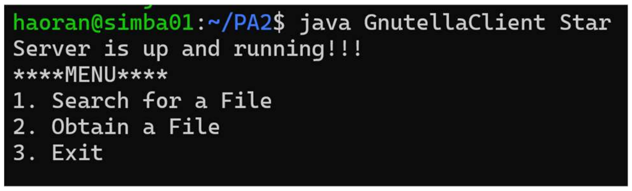
4.Run and Measurement

4.1 Run decentralized client(take star topology for example)

(1) Startup Gnutella Server on simba02…simba16:

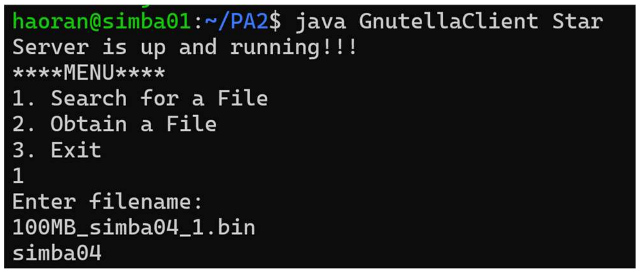

```
haoran@simba:~/PA2$ sshpass -p password parallel-ssh -h node_1.server -l haoran -A -i --par 50 --timeout 0  "cd
/home/haoran/PA2; java GnutellaServer Star"
Warning: do not enter your password if anyone else has superuser
privileges or access to your account.
```

(2) Run Gnutella Client on simba01：

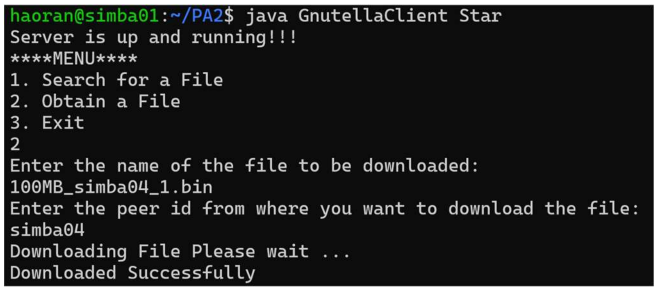

```
haoran@simba01:~/PA2$ java GnutellaClient Star
Server is up and running!!!
****MENU****
1. Search for a File
2. Obtain a File
3. Exit
```

(3) Output:

Search for a File:


```
haoran@simba01:~/PA2$ java GnutellaClient Star
Server is up and running!!!
****MENU****
1. Search for a File
2. Obtain a File
3. Exit
1
Enter filename:
100MB_simba04_1.bin
simba04
```

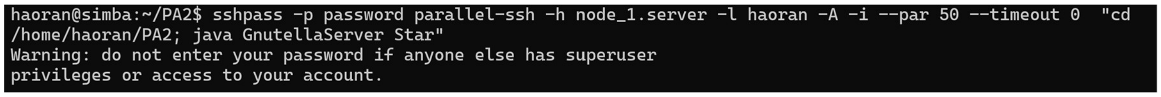
Obtain a File:

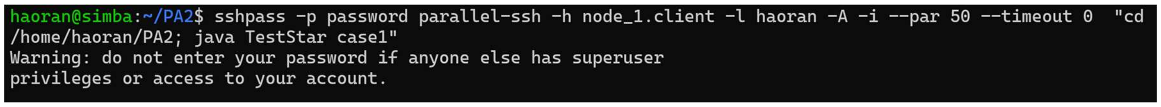

```
haoran@simba01:~/PA2$ java GnutellaClient Star
Server is up and running!!!
****MENU****
1. Search for a File
2. Obtain a File
3. Exit
2
Enter the name of the file to be downloaded:
100MB_simba04_1.bin
Enter the peer id from where you want to download the file:
simba04
Downloading File Please wait ...
Downloaded Successfully
```

4.2 Run test

(1) Interactive test (take star topology for example)

For interactive test, use command as following(take star topology for example)， first startup 15 star server(simba02-simba16) on host, then run client on simba01:


```
haoran@simba:~/PA2$ sshpass -p password parallel-ssh -h node_1.server -l haoran -A -i --par 50 --timeout 0  "cd
/home/haoran/PA2; java GnutellaServer Star"
Warning: do not enter your password if anyone else has superuser
privileges or access to your account.
```

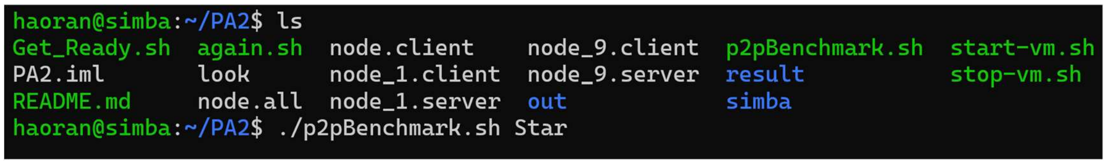


```
haoran@simba:~/PA2$ sshpass -p password parallel-ssh -h node_1.client -l haoran -A -i --par 50 --timeout 0  "cd
/home/haoran/PA2; java TestStar case1"
Warning: do not enter your password if anyone else has superuser
privileges or access to your account.
```

Case1 in the above screenshot is running Query latency experiments; Case2 refers to running Query throughput experiments; Case3 refers to running Transfer throughput Small experiments; Case4 refers to running Transfer throughput Large experiments.

(2) Batch benchmark (take star topology for example)

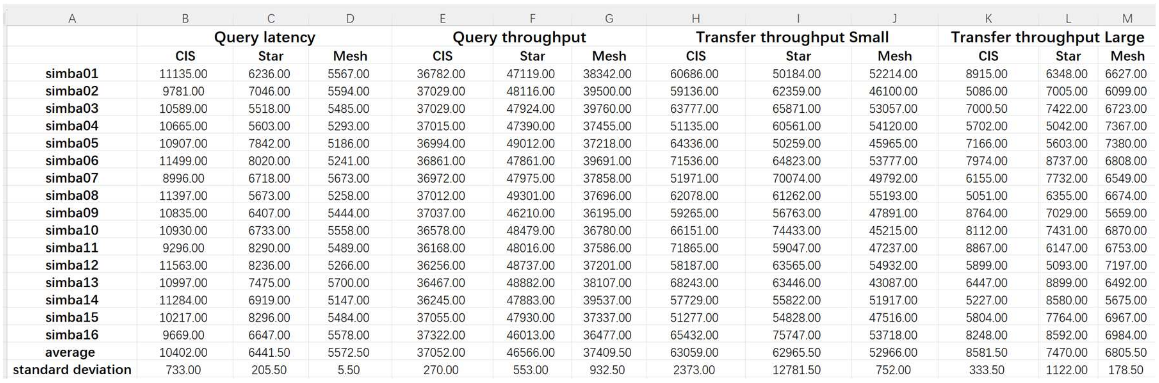
For batch benchmark run p2pBenchmark.sh on host(make it executable with chmod u+x p2pBenchmark.sh and invoke it with ./p2pBenchmark.sh) or execute shell command from script to run test one by one. Argument can be Star, Mesh, Central. Note: If no argument provide, include all.



```
haoran@simba:~/PA2$ ls
Get_Ready.sh  again.sh  node.client    node_9.client  p2pBenchmark.sh  start-vm.sh
PA2.iml       look      node_1.client  node_9.server  result           stop-vm.sh
README.md     node.all  node_1.server  out            simba
haoran@simba:~/PA2$ ./p2pBenchmark.sh Star
```

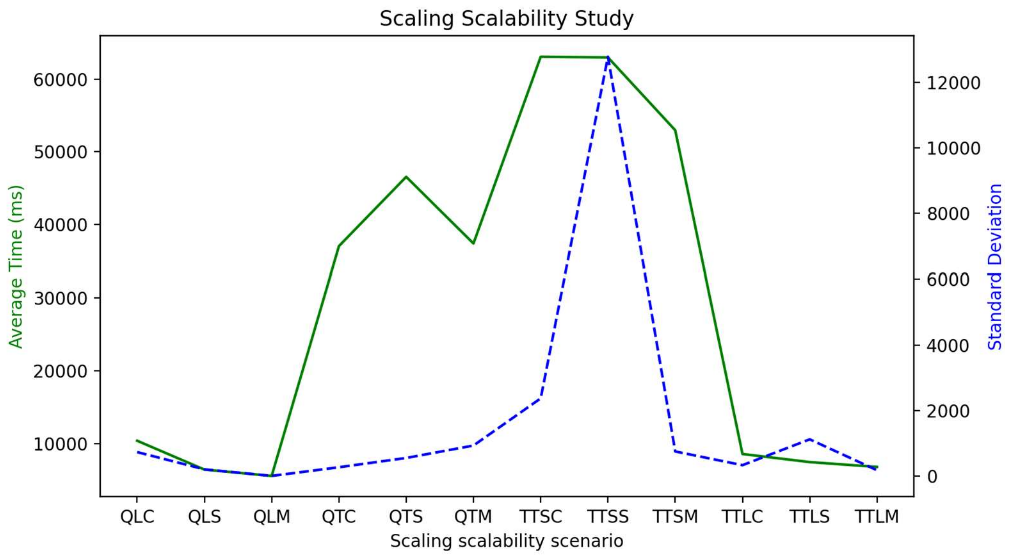
4.3 Measurement

(1) Experimental results statistics：



A	B	C	D	E	F	G	H	I	J	K	L	M
	Query latency			Query throughput			Transfer throughput Small			Transfer throughput Large		
	CIS	Star	Mesh	CIS	Star	Mesh	CIS	Star	Mesh	CIS	Star	Mesh
simba01	11135.00	6236.00	5567.00	36782.00	47119.00	38342.00	60686.00	50184.00	52214.00	8915.00	6348.00	6627.00
simba02	9781.00	7046.00	5594.00	37029.00	48116.00	39500.00	59136.00	62359.00	46100.00	5086.00	7005.00	6099.00
simba03	10589.00	5518.00	5485.00	37029.00	47924.00	39760.00	63777.00	65871.00	53057.00	7000.50	7422.00	6723.00
simba04	10665.00	5603.00	5293.00	37015.00	47390.00	37455.00	51135.00	60561.00	54120.00	5702.00	5042.00	7367.00
simba05	10907.00	7842.00	5186.00	36994.00	49012.00	37218.00	64336.00	50259.00	45965.00	7166.00	5603.00	7380.00
simba06	11499.00	8020.00	5241.00	36861.00	47861.00	39691.00	71536.00	64823.00	53777.00	7974.00	8737.00	6808.00
simba07	8996.00	6718.00	5673.00	36972.00	47975.00	37858.00	51971.00	70074.00	49792.00	6155.00	7732.00	6549.00
simba08	11397.00	5673.00	5258.00	37012.00	49301.00	37696.00	62078.00	61262.00	55193.00	5051.00	6355.00	6674.00
simba09	10835.00	6407.00	5444.00	37037.00	46210.00	36195.00	59265.00	56763.00	47891.00	8764.00	7029.00	5659.00
simba10	10930.00	6733.00	5558.00	36578.00	48479.00	36780.00	66151.00	74433.00	45215.00	8112.00	7431.00	6870.00
simba11	9296.00	8290.00	5489.00	36168.00	48016.00	37586.00	71865.00	59047.00	47237.00	8867.00	6147.00	6753.00
simba12	11563.00	8236.00	5266.00	36256.00	48737.00	37201.00	58187.00	63565.00	54932.00	5899.00	5093.00	7197.00
simba13	10997.00	7475.00	5700.00	36467.00	48882.00	38107.00	68243.00	63446.00	43087.00	6447.00	8899.00	6492.00
simba14	11284.00	6919.00	5147.00	36245.00	47883.00	39537.00	57729.00	55822.00	51917.00	5227.00	8580.00	5675.00
simba15	10217.00	8296.00	5484.00	37055.00	47930.00	37337.00	51277.00	54828.00	47516.00	5804.00	7764.00	6967.00
simba16	9669.00	6647.00	5578.00	37322.00	46013.00	36477.00	65432.00	75747.00	53718.00	8248.00	8592.00	6984.00
average	10402.00	6441.50	5572.50	37052.00	46566.00	37409.50	63059.00	62965.50	52966.00	8581.50	7470.00	6805.50
standard deviation	733.00	205.50	5.50	270.00	553.00	932.50	2373.00	12781.50	752.00	333.50	1122.00	178.50

(2) Graphical presentation of experimental results



4.4 Point-to-Point Response

(1) Q: Which topology seems to offer the lowest latency under light load (1 client)?

A: 2-D mesh topology offer the lowest latency under light load. Because in a mesh topology, nodes are interconnected allowing for direct communication between nodes. For star topology, communication between nodes depends on the central node.

(2) Q: How about query throughput under high load (9 clients)?

A: High load has increased latency. Because high load can lead to potential resource overload, and a higher risk of network congestion.

(3) Q: What do you observe on the transfer of data between small and large files?

A: Transferring small files is faster, but have a higher overhead. Large files take longer to transfer, but the overhead is lower.