CS6675
Advanced Internet Systems and Application Development
Assignment 2

Sathvik Karatattu Padmanabha

January 2025

Student Session: CS6675-A
GT ID: xx4032361

Topic: Programming
Problem 2, Option 1:
Hand on experience with a Structured Peer to Peer System - OpenChord

Contents

1 Introduction

1.1 Problem

I have chosen the Programming option with Hand on experience with a Structured Peer to Peer System - OpenChord. I have attempted to setup a DHT system and run experiments on lookup time for various configurations.

1.2 Open Source Code Reference and GitHub Link

I have chosen OpenChord [1], an open source DHT library for the purpose of this assignment. Most of the code was inspired from the examples in the official documentation [2]. However, I added few modifications as required by the assignment.

Github Link: https://github.com/SathvikKP/OpenChordTests

1.3 Environment and Instructions to run the code

I have attached the entire project directory when submitting the assignment. The steps to run and detailed description are provided in README.md file. We require any ubuntu / linux environment with any openjdk or jdk libraries. Copy my code to the machine and run the runme.sh script. I ran the code on my local laptop on a Windows Subsystem for Linux Environment. The submission also contains the log files of all my executions (logs_<exp_names>).

1.4 Project Directory Structure

The project is basically the OpenChord source code with additional files that I added. The 3 main files that I added are

- src/de/uniba/wiai/lspi/chord/service/impl/OpenChordTest.java –> The main code logic which either spawns a bootstrap peer (for creation of network) or client peers (which simply join existing network). Each peer then has an infinite loop where they can do the following actions:
  - insert –> insert a key value pair to the DHT
  - lookup –> lookup the value for a given key
  - runtests1 –> performs lookup many times and calculates average
  - runtests2 –> performs insertion many times and calculates average
- src/de/uniba/wiai/lspi/chord/service/impl/StringKey.java –> This contains implementation for custom data insertion into DHT
- runme.sh –> Main driver shell script that spawns several terminals and runs a peer client within each terminal.

The 2 main existing config files that I modified are

- config/chord.properties –> This contains various network properties such as number of successors per peer, thread settings etc.
- config/log4j.properties –> I modified this file so that each peer will have a separate log file. I also use the ”WARN” flag to log the messages. The default ”DEBUG” logs simply flood the log files, and hence, I used the ”WARN” flag and filtered only these to make it very convenient to analyze and experiment.

2 Experiment Details

I performed the following 3 experiments to test the lookup times for the OpenChord DHT protocol. I tested by changing the number of successors, number of peers and number of insertions. To make it easy to analyze, everytime a peer is created, I will insert a key value pair of («peer_num», ”default”) and I will try to perform lookups for all the key value pairs from one peer.

2.1 Experiment 1: Vary the number of successors for a constant number of peers and observe the average lookup time for a fixed number of lookups.

Successors mean how many replicas or how many peers have an instance of data stored in them. The idea is to see the trend of average lookup time on increasing the number of successors. I kept the number of peers to 32 (the max number of peers my laptop could support stably) and changed the number of successors from 1, 2, 4 and 8. I calculated the lookup times for peer 1 first, then repeated the experiment for the same peer. Next, I selected another random peer and repeated. The results are shown in table 1.

Table 1: Lookup Times with Different Number of Successors (32 peers)

Successors	Peer 1	Repeat Peer 1	Random Peer	Repeat Random
1	298	232	271	232
2	286	234	248	211
4	276	234	243	209
8	253	211	238	204

Note: Averaging was done over 32 lookups
Peer 1 = Average Lookup time for peer 1 (ms)
Repeat Peer 1 = Average Lookup time for the same peer 1 second time (ms)
Random Peer = Average Lookup time for a random peer (ms)
Repeat Random = Average lookup time for the same random peer second time (ms)

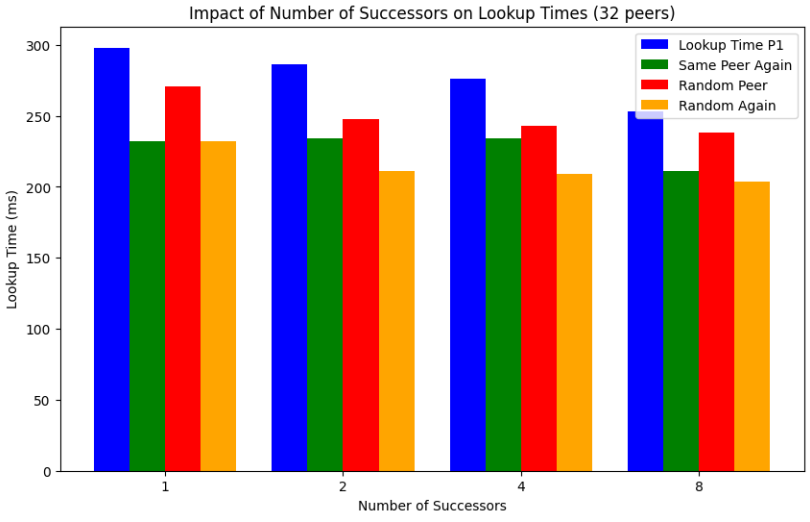

Figure 1: Impact of Number of Successors on Lookup Times (32 peers).

Analysis

First, we see that average lookup time decreases as we increase the number of successors. This is the expected behaviour. If a peer tries to retrieve data which is not found in the peer or nearby peer, the peer will have to look furthur apart in the chord ring which will increase the lookup time. Increasing the number of successors mean that there is a higher chance that the data that we are trying to retrieve is already present in one of the successors, and hence the lookup time reduces.

Second, we see that average lookup time reduces significantly when we repeat the experiment for the second time. This is due to the stabilizing effect of the chord protocol which updates the DHT after a stabilizing interval.

Third, after I performed lookup for peer 1, I observed that when I select a random peer, the lookup times are reduced for that peer also. This again is due to the stabilizing effect and the lookup times once again reduced when the lookup is repeated again.

2.2 Experiment 2: Keep the number of peers and successors for a peer as constant, but vary the number of insertions into the DHT. Calculate the average lookup time and average insertion time.

I wrote the code so that each peer automatically inserts a key value pair by default in the DHT. Now, I am adding additional key value pairs in the DHT and checking the impact of these new insertions on the lookup time. Table 2 shows the statistics that I obtained.

Table 2: Lookup Times with Extra Insertions (8 successors, 32 peers)

Extra Ins.	Total Ins.	Lookup Time	Insert Time
0	32	240	0
32	64	230	261
96	128	212	192
224	256	205	201
480	512	195	187

Extra Ins. = Number of extra insertions
Total Ins. = Total number of insertions (n)
Lookup Time = Average Lookup time (ms)
Insert Time = Average Additional Insertion Time (ms)

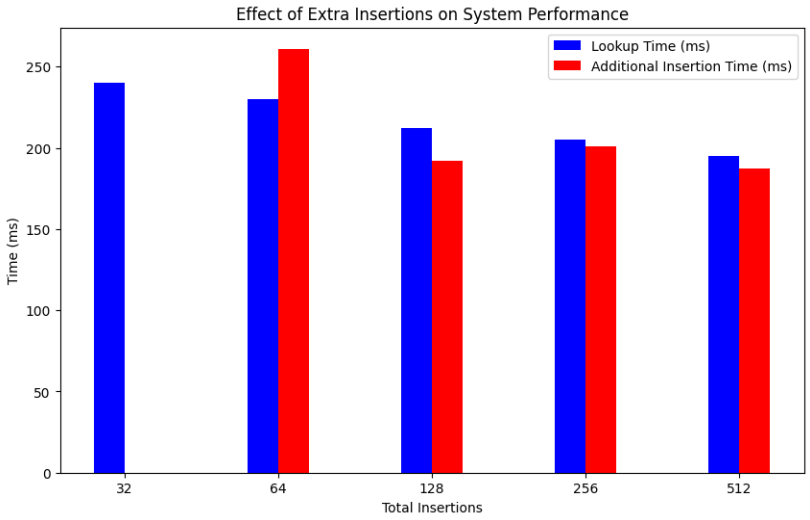

Figure 2: Effects of Extra Insertions on System Performance.

Analysis

I did not expect these results. Here, as the number of insertions increased, the average lookup time infact reduced! On furthur analysis, I realised this was due to the stabilizing effect in the OpenChord DHT. When the lookup is performed first time 32 times, the average lookup time is high as stabilizing and updating of routing tables is not properly performed initially. Next, when 32 more values are inserted and 64 lookups are performed, the lookup time for first 32 values is much faster (as they were retrived before) therefore, reducing the overall average lookup time. This explains the reduction in average lookup time as insertions increases.

Note: I cannot guarantee that this trend will continue forever due to memory issues in the DHT. In the absence of stabilizing effect, lookup times will increase.

Next, we see that the average insertion time also slightly reduces as we increase the number of insertions in the DHT but it is not stable. I tried the experiment many times and kept getting unstable results. Hence, I feel that average insertion time is independent of total number of insertions in the DHT and depends on other parameters like number of peers, successors etc. I feel increasing the number of successors will increase the insertion time.

2.3 Experiment 3: Keep the number of successors constant, but vary the number of peers. Calculate the average lookup times.

Next, I varied the number of peers spawned and calculated the average lookup time for 'n' lookups where n equals to the number of peers. Table 3 shows the statistics that I obtained.

Table 3: Effect of Number of Peers (n) on Lookup Times, number of successors = 8

Number of Peers	Peer 1	Repeat Peer 1	Random Peer	Repeat Random
4	86	86	85	85
8	114	111	106	101
16	209	131	158	133
32	253	211	238	204
64	crashed	crashed	crashed	crashed

Note: Averaging was done over 'n' lookups where n is number of peers
Note: for n = 4, number of successors = 4
Peer 1 = Average Lookup time for peer 1 (ms)
Repeat Peer 1 = Average Lookup time for the same peer 1 second time (ms)
Random Peer = Average Lookup time for a random peer (ms)
Repeat Random = Average lookup time for the same random peer second time (ms)

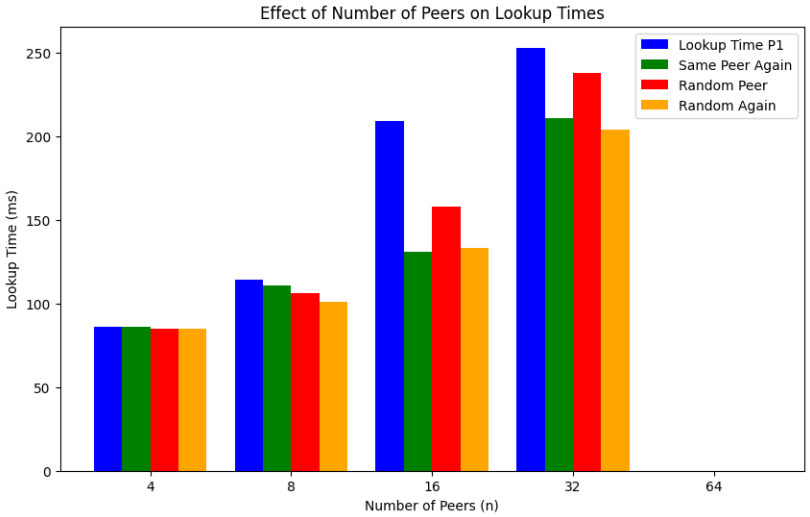

Figure 3: Effect of Number of Peers on Lookup Times

Analysis

I observed that as the number of peers was increased, the average lookup time also increased. But the increase is almost on a logarithmic scale. I increased the number of peers exponentially and the lookup time increased almost linearly. This is in accordance with the OpenChord DHT theoretical proofs which state that the lookup times will be reduced from O(N) to O(log(N)) in chord search. I performed the experiments few times to verify this, and it is nearly the same results all the time

Another interesting thing is how for small number of peers (n = 4 and n = 8), repeating the experiment did not improve the performance significantly. This was due to the number of successors equal to the number of peers. Hence, as the value to be retrieved was already present in the successors, the stabilizing effect does not improve performance upon repetition of the experiment.

My laptop was not able to handle 64 peers. It crashed the peers and I was unable to perform lookups.

3 Conclusion and Experience

3.1 Conclusion

- I learned about how structured peer to peer systems work, especially OpenChord DHT. I learned about how the lookups and insertions are performed, what successors and predecessors mean.
- I experimented and got reasonably accurate results compared to the theoretical proofs which say that the lookup times are O(log(N)). (Refer section 2.3. for results).
- I got an understanding of the lookup times, how it varies based on number of peers and successors, and how the stabilizing effect of the OpenChord protocol reduce the lookup time and increase efficiency. I also learned that repeating lookups will lead to quicker lookup times due to the updation of the DHT and due to the stabilizing effect
- I also understood how the OpenChord protocol uses different routing tables and finger tables for efficient lookups. The finger table part can be viewed only via console (simulation) mode which severely restricts other operations (see Figure 4 next page). Also see section 4 (Screenshots) which have the finger tables printed by the OpenChord software in debug mode (no API to print it manually).

3.2 Experience

- Most of the OpenChord files were dated to 2004-2005 with a few files having updates to 2008. This means that it is a very old software and is prone to bugs. There is not much official reference on the web on how to setup and run things. Hence, I had to use the official guide [2] to setup and run the basic code. Most of the code is directly present in the official guide. I added just basic logic and some parts like caclulating time and memory usage.
- I also collected the memory statistics for each operation but did not include memory usage in the report as I felt it was inaccurate (You can see it in the logs/ directory of the submission - it contains the memory stats). The OpenChord software is old and my memory statistics code did not work properly. I used the Ubuntu ”top” command to monitor the memory of the java processes and they remained constant throughout the operation (25 MB per peer client + 25 MB for the terminal).
- The logging for several peers was a nightmare, OpenChord did not provide a custom log flag, so I used the ”WARN” log flag to print my required messages and filtered based on this.
- The official documentation did not provide API to print the finger tables, routing tables and entries in a peer. These can only be viewed from the an inbuilt ”console” version which spawns a console where we can simulate spawning peers. However, this does not spawn true instances of peers and it does not provide mechanism for custom data insertion and lookup time calculations. Figure 4 shows the console version which simulates a few basic commands.

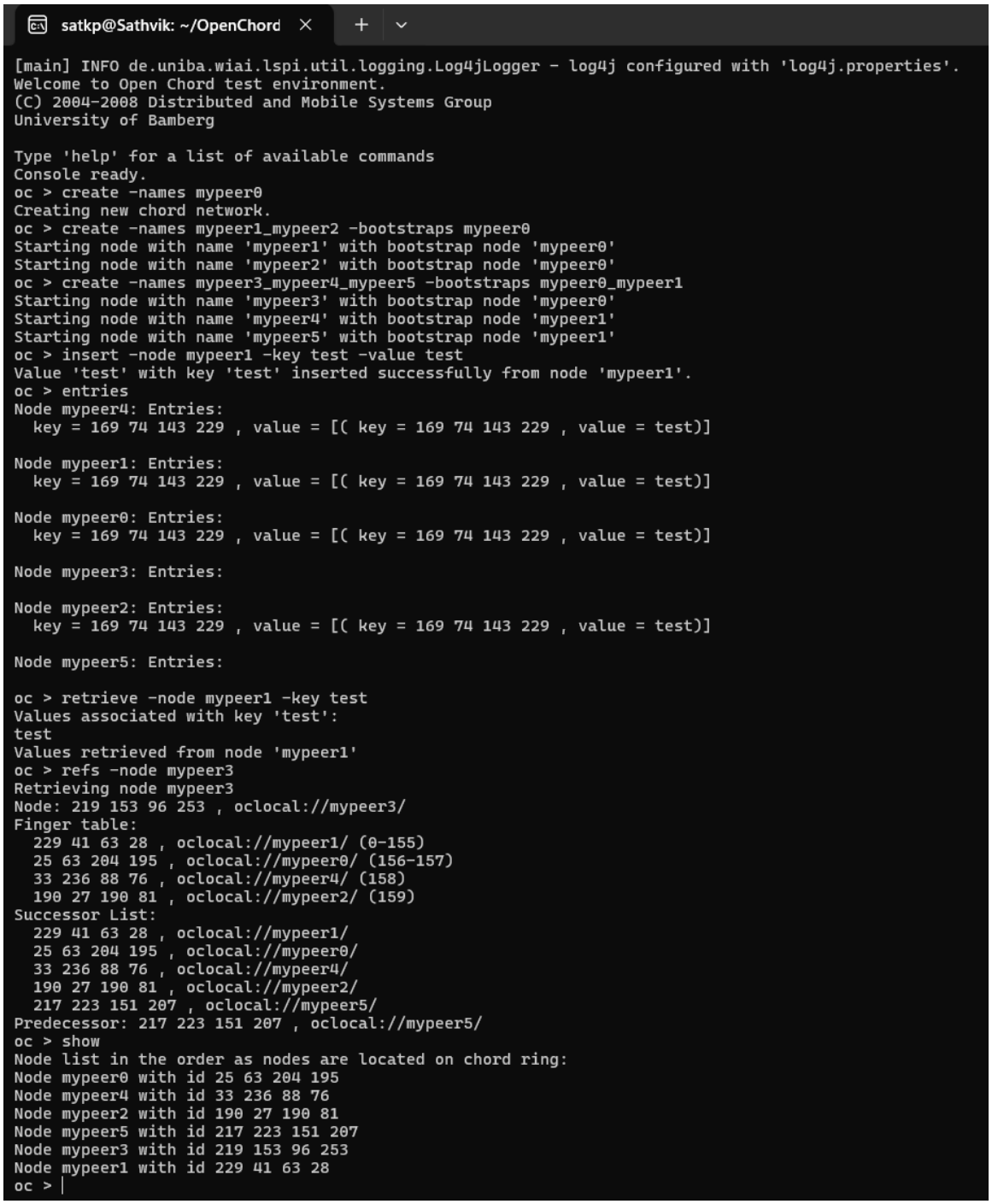

Figure 4: OpenChord Console with finger table, chord ring

4 Screenshots of Execution

```
satkp@Sathvik:~/OpenChord$ ls
Changelog.txt  README.txt  build      config      console_commands  docs            lib          logs                      runme.sh
README.md      bin         build.xml  console.sh  dist              experiment_logs  license.txt  open-chord-1.0.4.tar.gz  src
satkp@Sathvik:~/OpenChord$ ./runme.sh
Start ??? (^C to cancel)...
Bootstrap and peer nodes have been started in screen sessions. Check logs to see if any FATAL errors occurred.
screen -ls to list the screen sessions, screen -r <session_name> to attach to a session.
satkp@Sathvik:~/OpenChord$ screen -ls
There are screens on:
        20751.peer4     (02/07/25 16:51:12)     (Detached)
        20695.peer3     (02/07/25 16:51:10)     (Detached)
        20642.peer2     (02/07/25 16:51:08)     (Detached)
        20594.peer1     (02/07/25 16:51:06)     (Detached)
        20551.network   (02/07/25 16:51:02)     (Detached)
5 Sockets in /run/screen/S-satkp.
satkp@Sathvik:~/OpenChord$ screen -r peer1
[detached from 20594.peer1]
satkp@Sathvik:~/OpenChord$
```

Figure 5: Project Directory with all files, runme.sh script has started with 4 peers.

```
satkp@Sathvik: ~/OpenChord
ocket=Socket[addr=localhost/127.0.0.1,port=8080,localport=59054]] to Node[id=73 253 39 51 , url=ocsocket://localhost:8080/] is not added to successor list, because it is
already contained.
818 [main] INFO class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Added reference to finger table entries 0 to 153
818 [main] DEBUG class de.uniba.wiai.lspi.chord.service.impl.SuccessorList.71 124 129 194   - Reference to new node Connection from Node[url=ocsocket://localhost:8081/, s
ocket=Socket[addr=localhost/127.0.0.1,port=8080,localport=59054]] to Node[id=73 253 39 51 , url=ocsocket://localhost:8080/] is not added to successor list, because it is
already contained.
Peer joined at ocsocket://localhost:8081/ via ocsocket://localhost:8080/
820 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Peer joined at ocsocket://localhost:8081/ via ocsocket://localhost:8080/
825 [main] DEBUG class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Closest preceding node for ID 53 106 25 43  is Connection from Node[url=ocsock
et://localhost:8081/, socket=Socket[addr=localhost/127.0.0.1,port=8080,localport=59054]] to Node[id=73 253 39 51 , url=ocsocket://localhost:8080/]
Insert time for key 1 = 93 ms
918 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Insert time for key 1 = 93 ms
Inserted (1, default)
919 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Inserted (1, default)
Memory usage at ocsocket://localhost:8081/ = 11 MB
920 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Memory usage at ocsocket://localhost:8081/ = 11 MB

[Node ocsocket://localhost:8081/] Enter command: 'insert' or 'lookup' or 'runtests1' or 'runtests2' (or 'exit' to quit):
2688 [InvocationExecution-Thread-6] DEBUG class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Entry 154 set to Unconnected SocketProxy from ocsocke
t://localhost:8081/ to ocsocket://localhost:8082/
2689 [InvocationExecution-Thread-6] DEBUG class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Entry 155 set to Unconnected SocketProxy from ocsocke
t://localhost:8081/ to ocsocket://localhost:8082/
2689 [InvocationExecution-Thread-6] DEBUG class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Entry 156 set to Unconnected SocketProxy from ocsocke
t://localhost:8081/ to ocsocket://localhost:8082/
2689 [InvocationExecution-Thread-6] INFO class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Added reference to finger table entries 0 to 156
2689 [InvocationExecution-Thread-6] INFO class de.uniba.wiai.lspi.chord.service.impl.SuccessorList.71 124 129 194   - Added new reference to end of list
2887 [InvocationExecution-Thread-6] DEBUG class de.uniba.wiai.lspi.chord.service.impl.SuccessorList.71 124 129 194   - Inserted replicas to new reference
4679 [InvocationExecution-Thread-12] DEBUG class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Entry 157 set to Unconnected SocketProxy from ocsock
et://localhost:8081/ to ocsocket://localhost:8083/
4679 [InvocationExecution-Thread-12] DEBUG class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Entry 158 set to Unconnected SocketProxy from ocsock
et://localhost:8081/ to ocsocket://localhost:8083/
4679 [InvocationExecution-Thread-12] DEBUG class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Entry 159 set to Unconnected SocketProxy from ocsock
et://localhost:8081/ to ocsocket://localhost:8083/
4679 [InvocationExecution-Thread-12] INFO class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Added reference to finger table entries 0 to 159
4679 [InvocationExecution-Thread-12] INFO class de.uniba.wiai.lspi.chord.service.impl.SuccessorList.71 124 129 194   - Added new reference to end of list
4877 [InvocationExecution-Thread-12] DEBUG class de.uniba.wiai.lspi.chord.service.impl.SuccessorList.71 124 129 194   - Inserted replicas to new reference
6678 [InvocationExecution-Thread-5] DEBUG class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Entry 154 set to Unconnected SocketProxy from ocsocke
t://localhost:8081/ to ocsocket://localhost:8084/
6678 [InvocationExecution-Thread-5] DEBUG class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Entry 155 set to Unconnected SocketProxy from ocsocke
t://localhost:8081/ to ocsocket://localhost:8084/
6678 [InvocationExecution-Thread-5] DEBUG class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Entry 156 set to Unconnected SocketProxy from ocsocke
t://localhost:8081/ to ocsocket://localhost:8084/
6678 [InvocationExecution-Thread-5] INFO class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Added reference to finger table entries 0 to 156
6678 [InvocationExecution-Thread-5] INFO class de.uniba.wiai.lspi.chord.service.impl.SuccessorList.71 124 129 194   - Added new reference at position 1
6877 [InvocationExecution-Thread-5] DEBUG class de.uniba.wiai.lspi.chord.service.impl.SuccessorList.71 124 129 194   - Inserted replicas to new reference

Unknown.
61959 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Unknown.

[Node ocsocket://localhost:8081/] Enter command: 'insert' or 'lookup' or 'runtests1' or 'runtests2' (or 'exit' to quit):
```

Figure 6: Console output for peer1 - it has joined exising network, inserted a key value pair (1, default), and its tables are updated when other peers insert values.

```
[Node ocsocket://localhost:8081/] Enter command: 'insert' or 'lookup' or 'runtests1' or 'runtests2' (or 'exit' to quit):
insert
Enter key to insert:
1
Enter value for key 1:
file1
145897 [main] DEBUG class de.uniba.wiai.lspi.chord.service.impl.FingerTable.71 124 129 194   - Closest preceding node for ID 53 106 25 43  is Connection from Node[url=ocs
ocket://localhost:8081/, socket=Socket[addr=localhost/127.0.0.1,port=8083,localport=45018]] to Node[id=254 78 41 228 , url=ocsocket://localhost:8083/]
Insert time for key 1 = 52 ms
145949 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Insert time for key 1 = 52 ms
Inserted (1, file1)
145950 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Inserted (1, file1)
Memory usage at ocsocket://localhost:8081/ = 19 MB
145950 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Memory usage at ocsocket://localhost:8081/ = 19 MB

[Node ocsocket://localhost:8081/] Enter command: 'insert' or 'lookup' or 'runtests1' or 'runtests2' (or 'exit' to quit):
```

Figure 7: Peer 1 reinserts key 1 with value ”file1”.

```
Peer joined at ocsocket://localhost:8083/ via ocsocket://localhost:8080/
1214 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Peer joined at ocsocket://localhost:8083/ via ocsocket://localhost:8080/
1218 [main] DEBUG class de.uniba.wiai.lspi.chord.service.impl.FingerTable.254 78 41 228   - Closest preceding node for ID 119 222 104 218  is Connection from Node[url=ocs
ocket://localhost:8083/, socket=Socket[addr=localhost/127.0.0.1,port=8081,localport=38274]] to Node[id=71 124 129 194 , url=ocsocket://localhost:8081/]
Insert time for key 3 = 198 ms
1417 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Insert time for key 3 = 198 ms
Inserted (3, default)
1417 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Inserted (3, default)
Memory usage at ocsocket://localhost:8083/ = 14 MB
1417 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Memory usage at ocsocket://localhost:8083/ = 14 MB

[Node ocsocket://localhost:8083/] Enter command: 'insert' or 'lookup' or 'runtests1' or 'runtests2' (or 'exit' to quit):
3173 [InvocationExecution-Thread-9] INFO class de.uniba.wiai.lspi.chord.service.impl.FingerTable.254 78 41 228   - Added reference to finger table entries 0 to 158
3173 [InvocationExecution-Thread-9] INFO class de.uniba.wiai.lspi.chord.service.impl.SuccessorList.254 78 41 228   - Added new reference at position 2
3331 [InvocationExecution-Thread-9] DEBUG class de.uniba.wiai.lspi.chord.service.impl.SuccessorList.254 78 41 228   - Inserted replicas to new reference

Unknown.
191627 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Unknown.

[Node ocsocket://localhost:8083/] Enter command: 'insert' or 'lookup' or 'runtests1' or 'runtests2' (or 'exit' to quit):
lookup
Enter key to lookup:
1
Lookup time for key 1 = 50 ms
199259 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Lookup time for key 1 = 50 ms
Retrieved value: default
199259 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Retrieved value: default
Retrieved value: file1
199260 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Retrieved value: file1
Memory usage at ocsocket://localhost:8083/ = 17 MB
199260 [main] WARN de.uniba.wiai.lspi.chord.service.impl.OpenChordTest  - Memory usage at ocsocket://localhost:8083/ = 17 MB

[Node ocsocket://localhost:8083/] Enter command: 'insert' or 'lookup' or 'runtests1' or 'runtests2' (or 'exit' to quit):
```

Figure 8: Peer 3 joins the network, it inserts (3, default), its tables are updated when other peers insert values, then it performs lookup for key value 1, and it obtains updated value (”file1”).

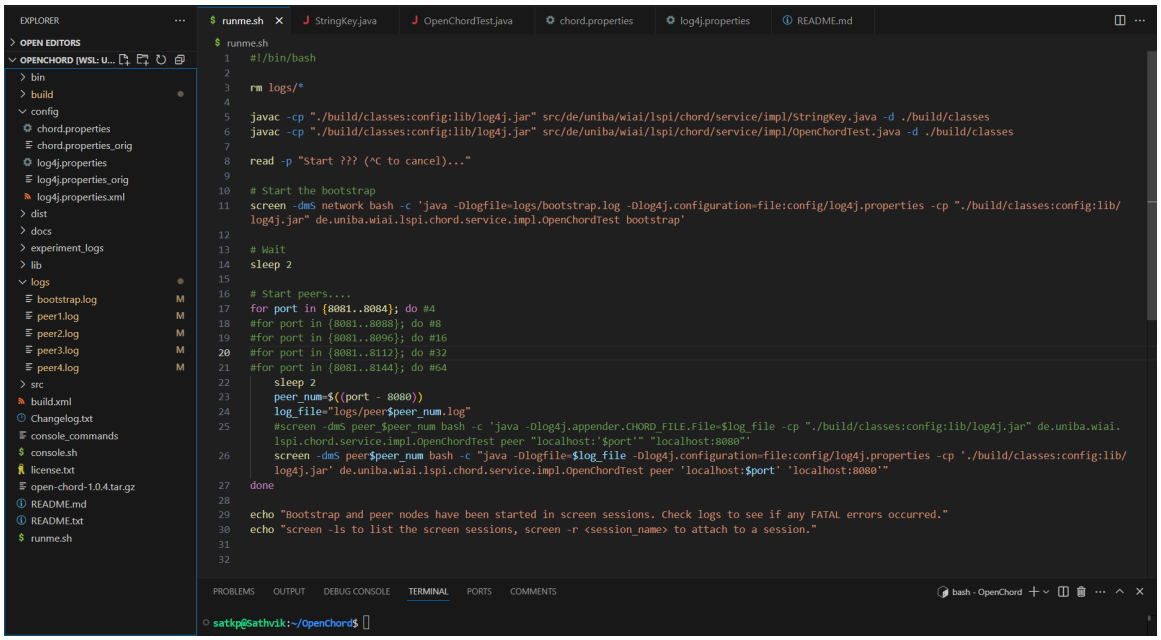

Figure 9: Project directory with runme.sh script, see the ’for’ loop which controls how many peers are spawned. Also look at the directory structure, a log file is generated separately for each peer.

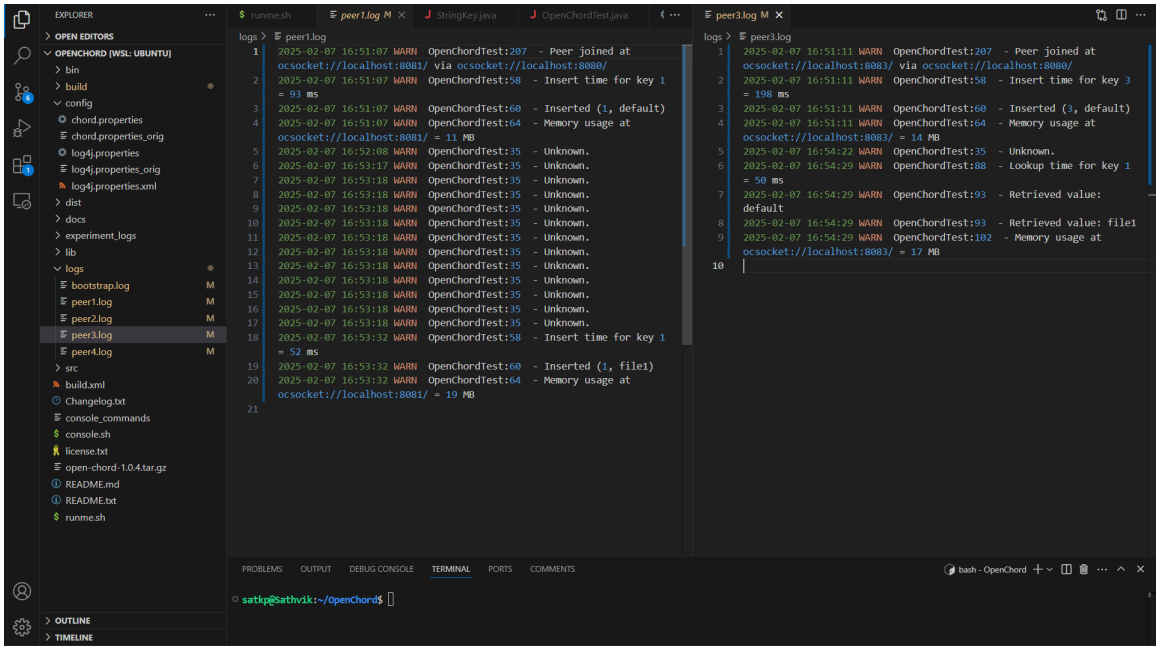

Figure 10: Comparison of logs for the experiments in previous figures. I have used the WARN flag to filter out the required logs for efficient analysis.

5 Deliverables

(a) URL to the downloaded P2P system

See section 1 (Introduction).

(b) Screen shots of your execution process.

See section 4 (Screenshots of Execution).

(c) Runtime statistics collected if any, you can use excel to plot figures, bar chart, and tables.

See section 2 (Experiment Details) .

(d) Discuss your results and experience.

See section 2 (Experiment Details), section 3 (Conclusion and Experience) and section 4 (Screenshots of Execution).

6 References

[1] https://open-chord.sourceforge.net/

[2] https://www.uni-bamberg.de/fileadmin/pi/Dateien/Forschung/open-chord_1.0.4_manual.pdf