

CS4103-Practical 2

Introduction – System Functionalities

Coordinator

- Ping all server nodes present in P (text file) and check their status every 20 seconds.
- Form and maintain a ring-shaped network with live server nodes in the order they are listed in P.
- Create a token and propagate it across the ring network to access a shared resource Q.

Server Nodes

- Manage 2 (max) Client node connections where they can send and receive posts from other users connected to different servers in the network.
- Check coordinator ID and if necessary call for and decide elections (single or multiple) to assign a new coordinator.
- Only allowed to communicate with successor in the ring and coordinator.
- Access Q exclusively when holding the token:
 - Pull/push a single client request from/to share resource Q.
 - Pass token when finished or if not required (i.e. clients not connected to server).
- Logging: Save communication information (sent and received messages) into a log text file.

Extensions

- Leaving (Partial)
- Multicast messages (Full)
- Fault tolerance (Partial)

Note: For testing P is locally hosted in the resources folder and shared resource Q is implemented as a server with a queue manager hosted in the local machine.

Design and Implementation in Java

Coordinator Functions

The coordinator functionality was implemented by creating a Coordinator class which was responsible for forming and maintaining the ring-shaped network whilst simultaneously servicing client and server node requests. When the node becomes a coordinator, it reads the resource P (stored locally as a text file) and stores the nodes' server location to later open a Java Socket. These are stored as an ArrayList to prevent any numerical ordering of the network. The coordinator pings all the nodes and compares the current live nodes with the previous cycle's pinging results. It then sends an INFORM message to set the nodes' successor endpoint and sends a TOKEN message. However, if the nodes have changed (joined or left), a new ring is always formed, and a new token is always issued. This can result in two tokens circulating around on network since the reforming process occurs regardless of the location of the token. This was addressed by implementing a randomised token number, unique to each ring formation which is sent when INFORM is sent. Nodes that receive a token message with the old token number would conclude a new token was issued by the coordinator and would not propagate the old token across anymore or attempt to access Q. An example of this is shown below:

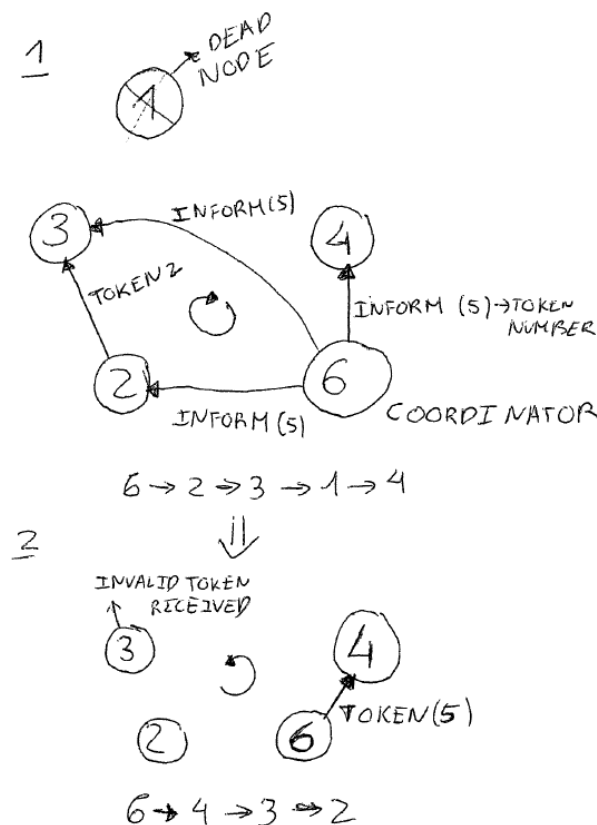


Figure 1. This figure illustrates the process of invalidating a token where node 1 is dead causing reforming of the network. Order of the ring also changed.

This helps ensure mutual access exclusion to Q whenever a new ring is formed. However, resource starvation can occur in nodes that preceded the coordinator, as they might not receive a valid token if the ring keeps changing. To prevent this, the order of the ring is reversed when the new token is issued as shown above in Figure 1.

Leaving nodes (Partially)

Non-coordinator nodes that fail are detected by the coordinator in the next Liveness check (every 20 seconds). This is possible since the ServerSockets run on a separate thread using Java Timer and the coordinator's routine will run concurrently. The same process accounts for nodes in resource P becoming live after a network is formed. The coordinator will then reform the ring-based network and issue a new token, whilst maintaining mutual exclusion to Q. However, support for leaving or joining the network via command requests sent to the coordinator were not implemented (P updating).

Server Nodes, Elections and New Coordinators

Server nodes use Java's ServerSockets and an Executor Thread Pool to implement a multithreaded server. All server nodes can call for leader elections via an ELECTION message when they receive the TOKEN and verify that their ID number is greater than the coordinator's. This is to prevent elections from being called before the ring is fully formed. The server nodes will then attach their IDs and circle the election messages and resolve the elections that they started by checking the first ID in the ELECTION message. They'll decide on the next coordinator and inform the network via a COORDINATOR message. The new coordinator will assume then assume the role.

Logging

All nodes are capable of logging received and transmitted into a text file created at bootup and stored locally.

Mutual Exclusion and Resource Management

Q was treated as a database server and hence it required a ServerSocket to communicate with the server nodes. It was responsible for managing client messages using LinkedLists to ensure the ordering of the message was kept (FIFO) and kept a message queue for each client. A HashMap with the receiving username set as the key and the number of messages available was added to provide faster lookup speeds, as the PULL request will always be issued by the server node if a client is connected to it. This minimises the time the token is held by the node when checking for new messages.

A structure called MessageClass was used to decompose the message and stored the information in a more usable format. Three commands were supported, ALL (Multicasting), ADD (Add post) and PULL (Read post) for return communications from clients to server to Q.

Client Read/Write and multicasting

ADD commands issued by the client and received by the server node would be stored in a local queue and eventually sent to Q once a token was received. Messages received when issuing the PULL command to Q are sent to the client immediately as clients will wait until the server sends a message.

A separate queue for multicasting was added for storing multicast messages and given a higher priority in comparison to normal addressed messages. All PULL requests check this queue before checking the Client specific message queue. Q will remove the message from the multicast queue only when the issuing node sends a PULL request.

Coordinator Failure

To add fault tolerance to coordinator failure, a state of “interim” coordinator was introduced. The coordinator’s predecessor (C) could detect if the coordinator node was dead when attempting to pass it the TOKEN. Then C would assume the role of coordinator on a temporary basis and behave as the coordinator, reform the network and issue a token that once received by C, would send an ELECTION message instead of passing the TOKEN to nominate a new coordinator. Since the COORDINATION message was not issued when C became coordinator, the nodes will not call for elections when receiving the token which prevents multiple elections being held. The network will eventually reform with a valid coordinator as shown below:

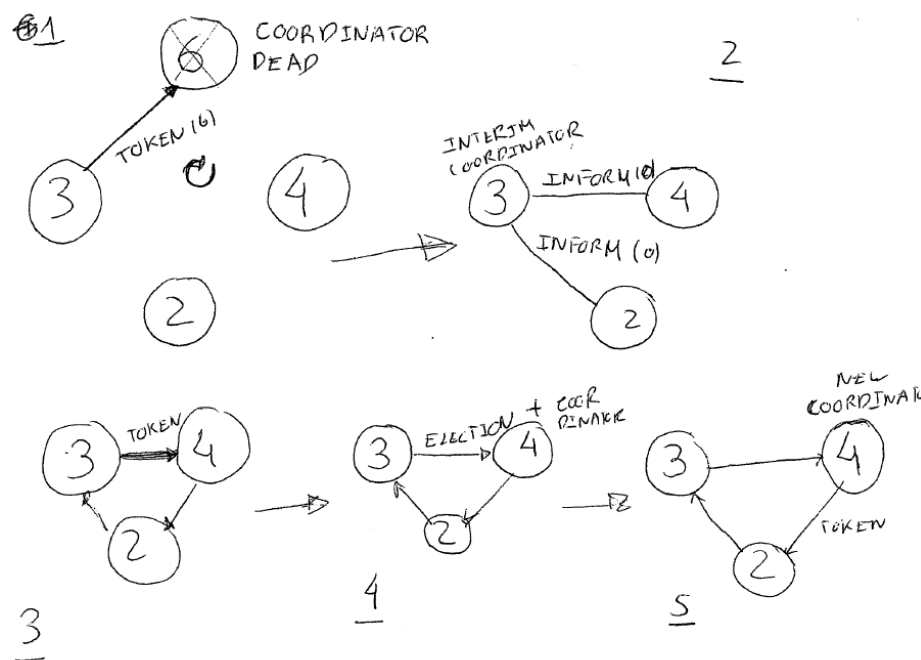


Figure 2.) Steps 1 through 5 illustrate the interim coordinator approach to dealing with coordinator failure.

Testing

Testing the network was difficult as it is difficult to automate this task. To test the different functionalities of the network certain error cases were introduced to allow for an easier simulation of the potential error case (i.e. never opening the socket for the coordinator to simulate coordinator failure). Screenshots of different tests conducted are shown below.

Double election

```

/usr/lib/jvm/java-1.8.0-openjdk/bin/java ...
4
My coordinator ID is: 5
Server started. Listening to port 8084
[2019-04-21 00:28:12.319 | 7] Receive from 5: PING
[2019-04-21 00:28:12.319 | 7] Receive from 5: INFORM 5,localhost,8085
[2019-04-21 00:28:13.295 | 7] Receive from 6: ELECTION {6,localhost,8081}
[2019-04-21 00:28:13.296 | 7] Receive from 6: TOKEN
[2019-04-21 00:28:13.296 | 7] Send to 5:ELECTION {7,localhost,8084}
[2019-04-21 00:28:13.297 | 7] Send to 5:TOKEN 67
[2019-04-21 00:28:13.297 | 7] Send to 5:ELECTION {6,localhost,8081;7,localhost,8084}
[2019-04-21 00:28:13.329 | 7] Receive from 6: TOKEN
[2019-04-21 00:28:13.329 | 7] Receive from 6: ELECTION {7,localhost,8084;5,localhost,8085;6,localhost,8081}
[2019-04-21 00:28:13.329 | 7] Send to 5:TOKEN 67
[2019-04-21 00:28:13.33 | 7] Receive from 6: COORDINATOR 7,localhost,8084
[2019-04-21 00:28:13.331 | 7] Send to 5:COORDINATOR 7,localhost,8084
[2019-04-21 00:28:13.334 | 7] Receive from 6: TOKEN
[2019-04-21 00:28:13.335 | 7] Send to 5:TOKEN 67
[2019-04-21 00:28:13.335 | 7] Send to 5:COORDINATOR 7,localhost,8084
[2019-04-21 00:28:13.339 | 7] Receive from 6: COORDINATOR 7,localhost,8084
I am coordinating, node ID: 7
[2019-04-21 00:28:13.345 | 7] Receive from 6: TOKEN
Network reforming. Wait for coordinator.
[2019-04-21 00:28:13.349 | 7] Send to 1:PING
: Node is not alive
[2019-04-21 00:28:13.35 | 7] Send to 6:PING
[2019-04-21 00:28:13.35 | 7] Send to 4:PING
: Node is not alive
[2019-04-21 00:28:13.35 | 7] Send to 5:PING
[2019-04-21 00:28:13.351 | 7] Send to 2:PING
: Node is not alive
Forming Ring
[2019-04-21 00:28:13.351 | 7] Send to 6:INFORM 5,localhost,8085;52
[2019-04-21 00:28:13.352 | 7] Send to 5:INFORM 7,localhost,8084;52
Linking myself(7) to 6
[2019-04-21 00:28:14.352 | 7] Send to 6:TOKEN 52
[2019-04-21 00:28:14.356 | 7] Receive from 5: TOKEN
[2019-04-21 00:28:14.356 | 7] Send to 6:TOKEN 52
[2019-04-21 00:28:14.358 | 7] Receive from 5: TOKEN
[2019-04-21 00:28:14.358 | 7] Send to 6:TOKEN 52
[2019-04-21 00:28:14.359 | 7] Receive from 5: TOKEN

```

Figure 3.) When the original coordinator ID is 5, nodes 6 and 7 will send ELECTION messages and COORDINATOR messages to set Node 7 as the coordinator.

Election, Token and Accessing Q

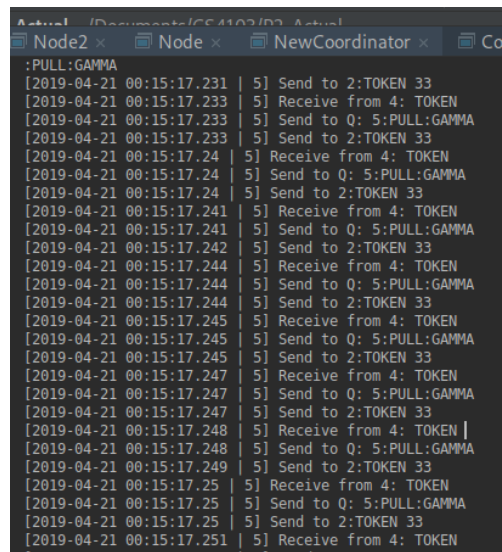
```

019-04-21 00:51:41.92 | 6] Receive from 0: ADD
019-04-21 00:52:00.765 | 6] Receive from 5: PING
019-04-21 00:52:00.772 | 6] Receive from 5: INFORM 4,localhost,8082
019-04-21 00:52:01.801 | 6] Receive from 1: TOKEN
019-04-21 00:52:01.801 | 6] Send to 0: 6:PULL:ALPHA
019-04-21 00:52:01.818 | 6] Send to 0: 6:ADD:OMEGA:ALPHA:Hello
019-04-21 00:52:01.82 | 6] Send to 4:ELECTION {6,localhost,8081}
019-04-21 00:52:01.82 | 6] Send to 4:TOKEN 57
019-04-21 00:52:02.002 | 6] Receive from 1: ELECTION {6,localhost,8081;4,localhost,8082;2,localhost,8083;7,localhost,8084;5,localhost,8085;1,localhost,8080}
019-04-21 00:52:02.01 | 6] Send to 4:COORDINATOR 7,localhost,8084
019-04-21 00:52:02.02 | 6] Receive from 1: TOKEN
019-04-21 00:52:02.02 | 6] Receive from 1: ELECTION {7,localhost,8084;5,localhost,8085;1,localhost,8080}
019-04-21 00:52:02.02 | 6] Send to 4:ELECTION {7,localhost,8084;5,localhost,8085;1,localhost,8080;6,localhost,8081}

```

Figure 4.) Elections and Q operations running simultaneously.

Pulling from Q



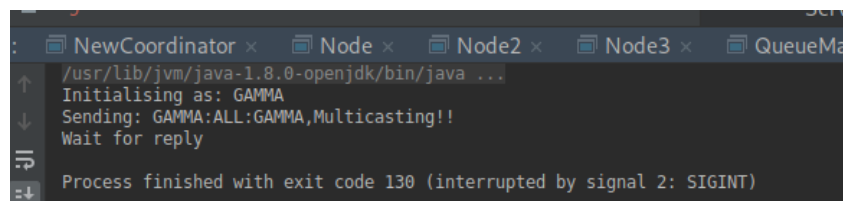
```

Node2 x Node x NewCoordinator x Co
: PULL:GAMMA
[2019-04-21 00:15:17.231 | 5] Send to 2:TOKEN 33
[2019-04-21 00:15:17.233 | 5] Receive from 4: TOKEN
[2019-04-21 00:15:17.233 | 5] Send to Q: 5:PULL:GAMMA
[2019-04-21 00:15:17.233 | 5] Send to 2:TOKEN 33
[2019-04-21 00:15:17.24 | 5] Receive from 4: TOKEN
[2019-04-21 00:15:17.24 | 5] Send to Q: 5:PULL:GAMMA
[2019-04-21 00:15:17.24 | 5] Send to 2:TOKEN 33
[2019-04-21 00:15:17.241 | 5] Receive from 4: TOKEN
[2019-04-21 00:15:17.241 | 5] Send to Q: 5:PULL:GAMMA
[2019-04-21 00:15:17.242 | 5] Send to 2:TOKEN 33
[2019-04-21 00:15:17.244 | 5] Receive from 4: TOKEN
[2019-04-21 00:15:17.244 | 5] Send to Q: 5:PULL:GAMMA
[2019-04-21 00:15:17.244 | 5] Send to 2:TOKEN 33
[2019-04-21 00:15:17.245 | 5] Receive from 4: TOKEN
[2019-04-21 00:15:17.245 | 5] Send to Q: 5:PULL:GAMMA
[2019-04-21 00:15:17.245 | 5] Send to 2:TOKEN 33
[2019-04-21 00:15:17.247 | 5] Receive from 4: TOKEN
[2019-04-21 00:15:17.247 | 5] Send to Q: 5:PULL:GAMMA
[2019-04-21 00:15:17.247 | 5] Send to 2:TOKEN 33
[2019-04-21 00:15:17.248 | 5] Receive from 4: TOKEN
[2019-04-21 00:15:17.248 | 5] Send to Q: 5:PULL:GAMMA
[2019-04-21 00:15:17.249 | 5] Send to 2:TOKEN 33
[2019-04-21 00:15:17.25 | 5] Receive from 4: TOKEN
[2019-04-21 00:15:17.25 | 5] Send to Q: 5:PULL:GAMMA
[2019-04-21 00:15:17.25 | 5] Send to 2:TOKEN 33
[2019-04-21 00:15:17.251 | 5] Receive from 4: TOKEN

```

Figure 5.) Accessing Q requires the TOKEN.

Multicasting

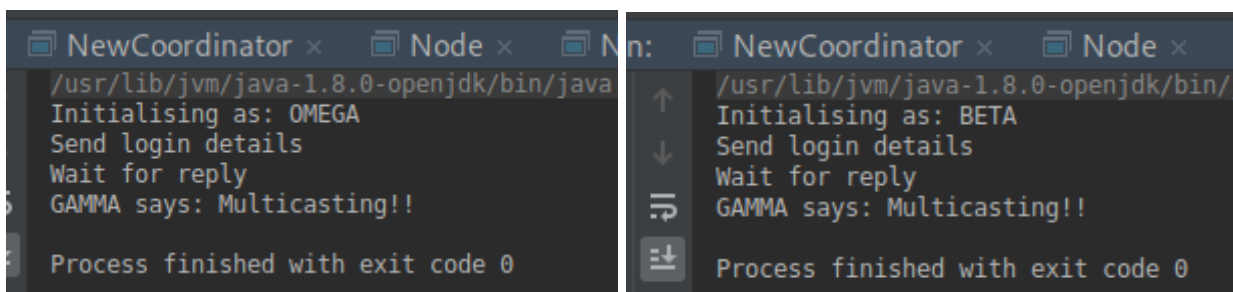


```

NewCoordinator x Node x Node2 x Node3 x QueueMa
/usr/lib/jvm/java-1.8.0-openjdk/bin/java ...
Initialising as: GAMMA
Sending: GAMMA:ALL:GAMMA,Multicasting!!
Wait for reply
Process finished with exit code 130 (interrupted by signal 2: SIGINT)

```

Figure 6.) GAMMA client multicasting.



```

NewCoordinator x Node x Nn: NewCoordinator x Node x
/usr/lib/jvm/java-1.8.0-openjdk/bin/java
Initialising as: OMEGA
Send login details
Wait for reply
GAMMA says: Multicasting!!
Process finished with exit code 0

/usr/lib/jvm/java-1.8.0-openjdk/bin/
Initialising as: BETA
Send login details
Wait for reply
GAMMA says: Multicasting!!
Process finished with exit code 0

```

Figure 7.) OMEGA and BETA receiving multicast message.

```

Reply Sent.
Multicast queue is:1
Waiting for clients request
Current queue size is:0
Connection opened
5:PULL:GAMMA
Message received: 5:PULL:GAMMA
Send reply:OK
Reply Sent.
Waiting for clients request
Connection opened
5:ALL:GAMMA:Multicasting!!
Message received: 5:ALL:GAMMA:Multicasting!!
Send reply:OK
Reply Sent.
Waiting for clients request
Connection opened
1:PULL:BETA
Message received: 1:PULL:BETA
Send reply:BETA:GAMMA:Multicasting!!
Reply Sent.

```

Figure 8.) Q handling of multicast.

Coordinator Fault Tolerance

```

My coordinator ID is: 5
Server started. Listening to port 8080
[2019-04-21 00:49:32.275 | 1] Receive from 5: PING
[2019-04-21 00:49:32.279 | 1] Receive from 5: INFORM 5,localhost,8085
[2019-04-21 00:49:33.264 | 1] Receive from 5: TOKEN
Node is not alive. Check Coordinator.,
coordinator is dead
I am coordinating, node ID: 1
[2019-04-21 00:49:33.264 | 1] Send to 5:TOKEN 13
[2019-04-21 00:49:33.282 | 1] Send to 6:PING
: Node is not alive
[2019-04-21 00:49:33.282 | 1] Send to 4:PING
[2019-04-21 00:49:33.285 | 1] Send to 5:PING
: Node is not alive
[2019-04-21 00:49:33.285 | 1] Send to 2:PING
: Node is not alive
[2019-04-21 00:49:33.285 | 1] Send to 7:PING
: Node is not alive
Forming Ring
[2019-04-21 00:49:33.287 | 1] Send to 4:INFORM 1,localhost,8080:50
Linking myself(1) to 4
[2019-04-21 00:49:34.288 | 1] Send to 4:TOKEN 50
[2019-04-21 00:49:34.289 | 1] Receive from 4: TOKEN
[2019-04-21 00:49:34.29 | 1] Send to 4:ELECTION {1,localhost,8080}
[2019-04-21 00:49:34.292 | 1] Receive from 4: ELECTION {1,localhost,8080;4,localhost,8082}
[2019-04-21 00:49:34.298 | 1] Send to 4:COORDINATOR 4,localhost,8082
[2019-04-21 00:49:34.301 | 1] Receive from 4: COORDINATOR 4,localhost,8082
My coordinator ID is: 4
[2019-04-21 00:49:34.311 | 1] Receive from 4: PING
[2019-04-21 00:49:34.312 | 1] Receive from 4: INFORM 4,localhost,8082
[2019-04-21 00:49:35.314 | 1] Receive from 4: TOKEN
[2019-04-21 00:49:35.314 | 1] Send to 4:TOKEN 21
[2019-04-21 00:49:35.318 | 1] Receive from 4: TOKEN
[2019-04-21 00:49:35.319 | 1] Send to 4:TOKEN 21

```

Figure 8.) Node 1 detecting coordinator failure, becoming interim coordinator and sending election message.

Limiting Client Connections to 2

```

bash-4.4$ java Client.Client RECEIVER 0
Initialising as: OMEGA
Send login details
Wait for reply
Your connection has been declined!
bash-4.4$

```

Figure 9.) Rejecting third connection

Node failure

```

[2019-04-21 00:28:15.674 | 5] Receive from 6: TOKEN
[2019-04-21 00:28:15.674 | 5] Send to 7:TOKEN 52
[2019-04-21 00:28:15.674 | 5] Receive from 6: TOKEN
[2019-04-21 00:28:15.675 | 5] Send to 7:TOKEN 52
[2019-04-21 00:28:15.675 | 5] Receive from 6: TOKEN
[2019-04-21 00:28:15.675 | 5] Send to 7:TOKEN 52
[2019-04-21 00:28:15.676 | 5] Receive from 6: TOKEN
[2019-04-21 00:28:15.676 | 5] Send to 7:TOKEN 52
[2019-04-21 00:28:15.676 | 5] Receive from 6: TOKEN
[2019-04-21 00:28:15.676 | 5] Send to 7:TOKEN 52
[2019-04-21 00:28:15.677 | 5] Receive from 6: TOKEN
[2019-04-21 00:28:15.677 | 5] Send to 7:TOKEN 52
[2019-04-21 00:28:15.692 | 5] Receive from 6: TOKEN
[2019-04-21 00:28:15.692 | 5] Send to 7:TOKEN 52

Process finished with exit code 130 (interrupted by signal 2: SIGINT)

```

Figure 10.) Node 5 dying.


```

[2019-04-21 00:28:15.677 | 7] Receive from 5: TOKEN
[2019-04-21 00:28:15.677 | 7] Send to 6:TOKEN 52
[2019-04-21 00:28:15.692 | 7] Receive from 5: TOKEN
[2019-04-21 00:28:15.693 | 7] Send to 6:TOKEN 52
[2019-04-21 00:28:34.353 | 7] Send to 1:PING
: Node is not alive
[2019-04-21 00:28:34.353 | 7] Send to 6:PING
[2019-04-21 00:28:34.353 | 7] Send to 4:PING
: Node is not alive
[2019-04-21 00:28:34.353 | 7] Send to 5:PING
: Node is not alive
[2019-04-21 00:28:34.354 | 7] Send to 2:PING
: Node is not alive
Forming Ring
[2019-04-21 00:28:34.354 | 7] Send to 6:INFORM 7,localhost,8084:63
Linking myself(7) to 6
[2019-04-21 00:28:35.354 | 7] Send to 6:TOKEN 63
[2019-04-21 00:28:35.355 | 7] Receive from 6: TOKEN
[2019-04-21 00:28:35.355 | 7] Send to 6:TOKEN 63
[2019-04-21 00:28:35.355 | 7] Receive from 6: TOKEN
[2019-04-21 00:28:35.355 | 7] Send to 6:TOKEN 63
[2019-04-21 00:28:35.355 | 7] Receive from 6: TOKEN
[2019-04-21 00:28:35.355 | 7] Send to 6:TOKEN 63

```

Figure 11.) Network reforming.

Network reforming with token invalidation

```

My coordinator ID is: 5
Server started. Listening to port 8082
[2019-04-22 15:08:53.74 | 4] Receive from 5: INFORM 1,localhost,8080
[2019-04-22 15:08:53.74 | 4] Receive from 5: PING
[2019-04-22 15:08:54.717 | 4] Receive from 5: TOKEN
[2019-04-22 15:08:54.717 | 4] Send to 1:TOKEN 39
[2019-04-22 15:08:54.719 | 4] Receive from 5: TOKEN
[2019-04-22 15:08:54.719 | 4] Send to 1:TOKEN 39
[2019-04-22 15:08:54.72 | 4] Receive from 5: TOKEN
[2019-04-22 15:08:54.721 | 4] Send to 1:TOKEN 39
[2019-04-22 15:08:54.722 | 4] Receive from 5: TOKEN
[2019-04-22 15:08:54.722 | 4] Send to 1:TOKEN 39
[2019-04-22 15:08:54.722 | 4] Receive from 5: TOKEN

```

Figure 12.) Node 4 joining network

```

[2019-04-22 15:08:53.714 | 5] Receive from 1: TOKEN
[2019-04-22 15:08:53.714 | 5] Send to 1:TOKEN 33
[2019-04-22 15:08:53.714 | 5] Receive from 1: TOKEN
[2019-04-22 15:08:53.714 | 5] Send to 1:TOKEN 33
[2019-04-22 15:08:53.714 | 5] Receive from 1: TOKEN
[2019-04-22 15:08:53.714 | 5] Send to 1:TOKEN 33
[2019-04-22 15:08:53.715 | 5] Receive from 1: TOKEN
[2019-04-22 15:08:53.715 | 5] Send to 1:TOKEN 33
: Node is not alive
: Node is not alive
: Node is not alive
Forming Ring
[2019-04-22 15:08:53.715 | 5] Send to 1:PING
[2019-04-22 15:08:53.715 | 5] Send to 6:PING
[2019-04-22 15:08:53.715 | 5] Send to 4:PING
[2019-04-22 15:08:53.715 | 5] Send to 2:PING
[2019-04-22 15:08:53.715 | 5] Send to 7:PING
[2019-04-22 15:08:53.716 | 5] Send to 4:INFORM 1,localhost,8080:39
[2019-04-22 15:08:53.716 | 5] Send to 1:INFORM 5,localhost,8085:39
Linking myself(5) to 4
[2019-04-22 15:08:53.717 | 5] Receive from 1: TOKEN
Old token
[2019-04-22 15:08:54.716 | 5] Send to 4:TOKEN 39
[2019-04-22 15:08:54.718 | 5] Receive from 1: TOKEN
[2019-04-22 15:08:54.718 | 5] Send to 4:TOKEN 39
[2019-04-22 15:08:54.72 | 5] Receive from 1: TOKEN
[2019-04-22 15:08:54.72 | 5] Send to 4:TOKEN 39
[2019-04-22 15:08:54.721 | 5] Receive from 1: TOKEN
[2019-04-22 15:08:54.721 | 5] Send to 4:TOKEN 39

```

Figure 13.) Coordinator adding Node 4 to network and rejecting old token.

ADD/PULL client requests

```

Run: NewCoordinator x Node x
/usr/lib/jvm/java-1.8.0-openjdk/bin/
Initialising as: OMEGA
Send login details
Wait for reply
ALPHA says: Hello
Process finished with exit code 0

Run: NewCoordinator x Node x
/usr/lib/jvm/java-1.8.0-openjdk/bin/ja
Initialising as: ALPHA
Sending: ALPHA:ADD:OMEGA,ALPHA,Hello
Process finished with exit code 0

```

Figure 14.) Add and Pull Client requests from OMEGA and ALPHA

Evaluation

First and foremost, the quality of the code is low, and the code needs extensive refactoring even though it's fully functional. Lack of structure code structure makes adding functionality harder as objects are highly interlinked.

The system developed attempts to fulfil the requirements of the practical however there are certain events that are only possible as the network was tested in the local network (fast/reliable communication). In example, the coordinator pings the node and assumes it's dead if it cannot open the connection. An Acknowledge (ACK) reply with a timeout and retry counter would ensure that the node is dead and not just busy.

The system's communications work on the principle that sockets should be opened and closed when sending a message. Even though this might be useful for sporadic, less frequent connections, it severely hinders the performance of the network as opening a socket is a very time expensive process. Leaving them open would reduce the duration of coordinator operations and enable an increase in their frequency without becoming too intrusive.

As a HashMap is used to store the clients connected to server node, client starvation can occur as the map will be ordered when iterated through. This causes client starvation if the first client to be serviced always has requests for Q which means the other client requests cannot be addressed.

In multicasting, the message is delivered to clients by issuing a PULL command to Q. Since only one post can be removed, the server node will only push to a single client and the message will be removed from the multicast queue once the issuing node has the token. This means that the message might not be send to half of the users (assuming 2 clients per node). A solution is to add the message to Q and let the server node send it to the other client directly. Currently, there isn't a distinction between targeted and multicast messages when reading from Q. A separated data request would address all these issues. Additionally, there's the risk of indefinitely sending the multicast message if the node that issued it dies or does not have any clients connected. The multicast message will only be removed from Q when a PULL request is made by the original issuing node.

Dead nodes will temporarily halt the network until the schedule coordinator routine (20 seconds) is performed. The dead node's predecessor could inform the coordinator of its successor's status and request a new successor to the coordinator to avoid full network reshaping. The same concept of localised network reshaping in the case of node joining would be beneficial as the new nodes could be added as the coordinators' successors (in chain) hence minimising the disruption to the ring and preventing the token from being lost instead of always creating new TOKENs.

Coordinator fault tolerance only works if the coordinator's predecessor attempts to send a message and detects the node is dead. The network will not recover if the coordinator fails whilst holding the token. Since the COORDINATOR message is not sent, the coordinator's ID will not be updated in the server nodes.

If the “interim” coordinator dies before the token is received and the election called and resolved, its predecessor will not perform the same process (thinks successor is normal node). It will also only work when there is no election running.

Developing a chat system in the client node which could connect to any server and with any username would facilitate testing as all the parameters (i.e. messages) are hardcoded.

Conclusion

This was an interesting assignment that showed the difficulty of designing a distributed system. In terms of future work besides addressing the design flaws mentioned, commands JOIN and DELETE would be implemented for nodes to join and leave the network (updating P accordingly). Load balancing on the server nodes could be an extension as the servers only support requests from two clients. Any additional clients are rejected, when they could be instructed to connect to the server’s successor and so on until a free server was found.