University of St Andrews

Distributed Systems

CS4103

Ring-Based Distributed System

Author:
150008022

April 18, 2019

Goal

To demonstrate an understanding of leader election and mutual exclusion in distributed systems by developing a ring-based distributed social media application.

Contents

1 Initial Set-up

Java 8 was chosen for this project due to it's friendly socket API, and Maven [1] was used as the build tool. TDD was implemented with JUnit4 as the the test suite framework.

The project was started with the intention of implementing the bully algorithm as an extension, as it provided fault tolerance with low overhead, but also to provide the option to use the ring based algorithm for comparison.

1.1 Configuration

Since each node would be running on an isolated machine, the planned testing environment would involve using *ssh* to start the nodes remotely. Providing the configuration as command line arguments was considered simpler than other methods, and was implemented using the Apache Commons CLI library [2].

1.2 Sockets

Due to different patterns of communication during recovery and regular message passing phases both TCP and UDP were used for this project, as shown in Figure 1.
TCP was used for communication **around** the ring:

1. Reliable communication avoids token being lost.

2. Connection is reused frequently between predecessors and successors, justifying handshake overhead.

UDP was used for communication **across** the ring:

1. Low communication overhead allows for messages to be sent to multiple nodes quickly, improving recovery time from node failure.

2. No session maintenance allows coordinator to handle more members in the ring.

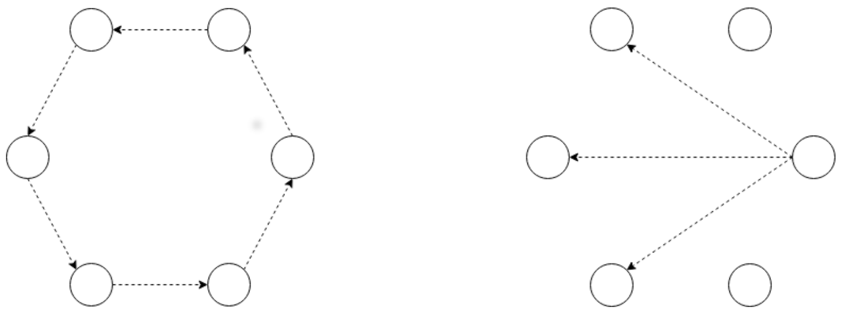
(a) TCP Around Ring (b) UDP Across Ring

Figure 1: The different types of communication used

2 Ring Formation

2.1 Initialization

On initialization, each node sends out join messages to the coordinator. The joining protocol is similiar to that described in [3], and is shown in Figure 2. Figure 3 shows how the state of each connection changes over the course of the joining procedure.
When node J wants to join the ring:

1. J sends join request to coordinator C.
2. C sends successor to J, telling it to connect to B.
3. C sends successor to A, telling it to connect to J.
4. A disconnects from B, B begins listening for new predecessor.
5. J connects to B, A connects to J.

In order for this process to work for a ring network with a single node, it required another thread to wait on the connection, and the main thread would then request the connection to itself. For two nodes and more the joining node would be able to first connect to its new successor once its predecessor had disconnected from its old successor as shown in figure 3.

The coordinator maintains a virtual representation of the network ring in terms of IDs, and keeps track of the last added node. When a new node joins, the connection occurs between itself and the last node that was added.

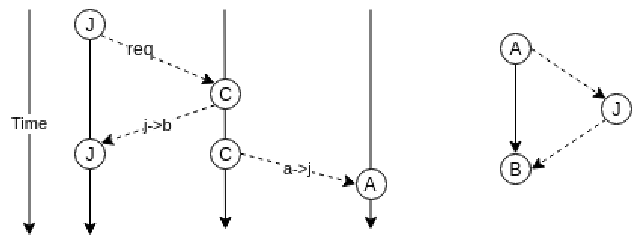

Figure 2: Joining protocol over time, with topology shown on right.

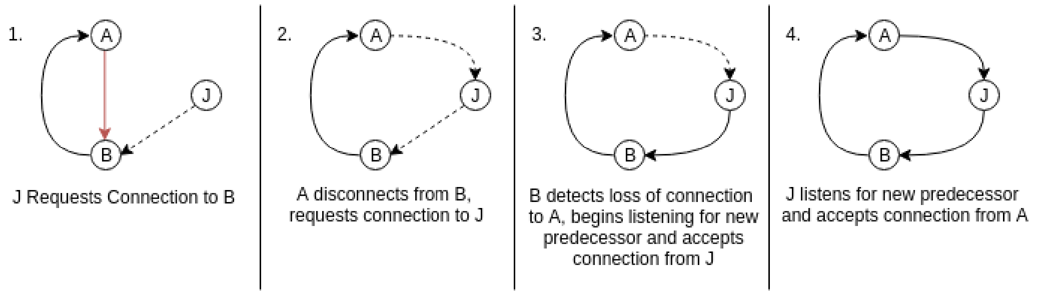
1.
A
J
B
J Requests Connection to B
2.
A disconnects from B, requests connection to J
3.
B detects loss of connection to A, begins listening for new predecessor and accepts connection from J
4.
J listens for new predecessor and accepts connection from A

Figure 3: State of each connection as a new node joins the ring. The red arrow shows the edge that is replaced with the new node, and solid and dashed arrows represent ongoing and pending connections respectively. Either A or B can be the coordinator in this scenario.

A different heuristic could be used to determine whereabouts the new node should be inserted into the ring such as

2.2 General Node Recovery

Node failure is detected whenever either the predecessor or successor of the failing node attempts to write or read to the relevant TCP socket. The predecessor is likely first to discover the failure, as they read from the socket until the token is available. The successor will notice either when it tries to forward the token or when it tries to send a keepalive.

The successor of the failing node will simply begin listening for a new predecessor. The predecessor of the failing node will request a new successor from the coordinator node. The coordinator will then reply with the ID of the node after the failed node for the predeccessor to connect to, at which point normal network behaviour can resume. Figure 4 shows the topology change when a node fails.

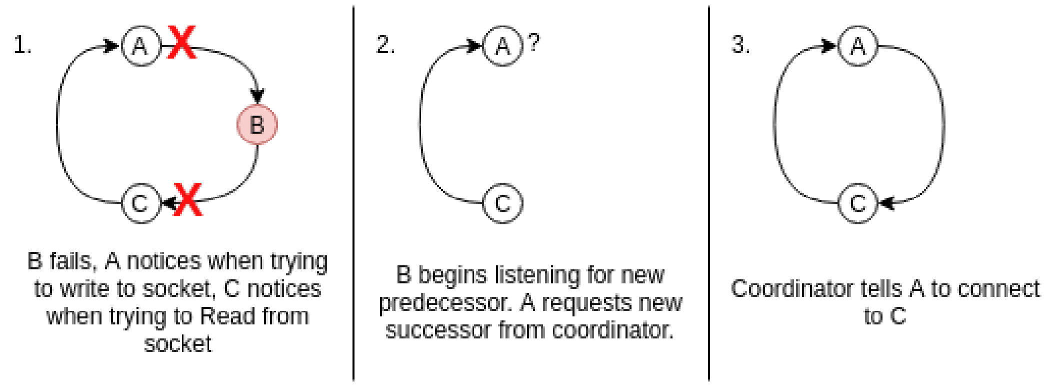
1.
A
B
C
B fails, A notices when trying to write to socket, C notices when trying to Read from socket
2.
A
?
C
B begins listening for new predecessor. A requests new successor from coordinator.
3.
A
C
Coordinator tells A to connect to C

Figure 4: Network topology during the recovery from node B failing. Either A or C are the coordinator in this scenario.

Since TCP sockets do not provide a method for confirming that a token has been received by next node in the ring at the application level, an acknowledgement mechanism was included in token passing.

3 Leader/Coordinator Election

4 Receive/Send Posts

References

[1] Apache. Apache maven project. `https://maven.apache.org/`.

[2] Apache Commons. Apache commons cli. `https://commons.apache.org/proper/commons-cli/`.

[3] D. Lee, A. Puri, P. Varaiya, R. Sengupta, R. Attias, and S. Tripakis. A wireless token ring protocol for ad-hoc networks. In *Proceedings, IEEE Aerospace Conference*, volume 3, pages 3–3, March 2002.