Report 4: Groupy - A Group Membership Service

Ziheng Zhang

October 6, 2021

1 Introduction

This assignment is mainly about the multicast process and try to keep all nodes consistent. The implementations are in three files, respectively, `gms1`, `gms2`, `gms3`, and use `test` file to check the robustness of implementation.

2 Main problems and solutions

2.1 The First Version Without Handling Failures

In the first version of the implementation, I built a group member service without handling failures. In the beginning, the first node will be created and because of no node in the group, therefore, the first node will automatically become the leader. Then, when the second node is created, it will become a "slave", and send a `join` message to the group. When the first node receives the `join` message, it will respond to it, and add it to the group, then broadcast the `view` message. The rest of nodes follow the same process.

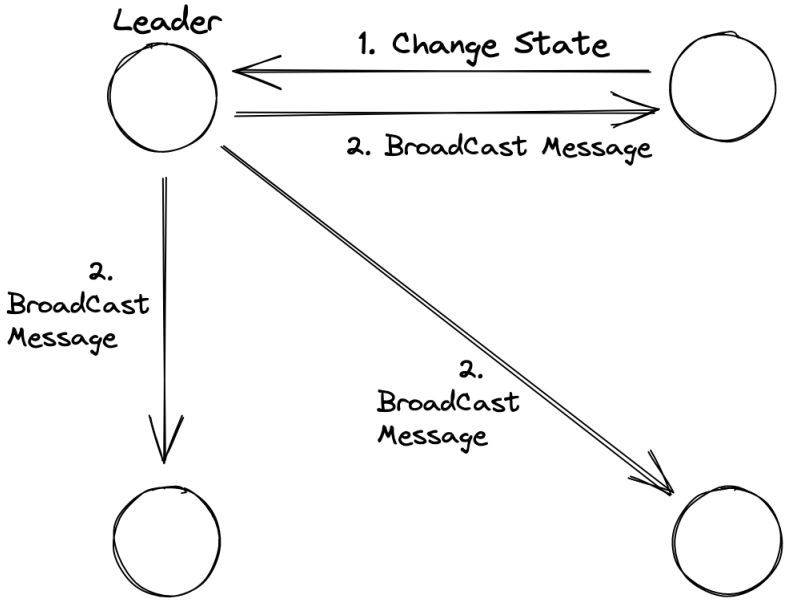

Figure 1: Change The State

We use color to represent the state of each node. When a node wants to change state, it will send the `change` message to the leader and the leader will broadcast it to each nodes so that all nodes will keep consistent.

2.2 The Second Version With Election

In the second version, the group membership service can handle several failures. I added `election` function into the file `gms2`. Specifically, the leader node will be monitored by other nodes. When the leader crashes, the new leader will be elected. The first node in the group will be the new leader.

However, this version of the implementation still lacks robustness. When the leader crashes before it send the messages, not all the slaves will receive the messages. Therefore, the nodes will be out of synchronization.

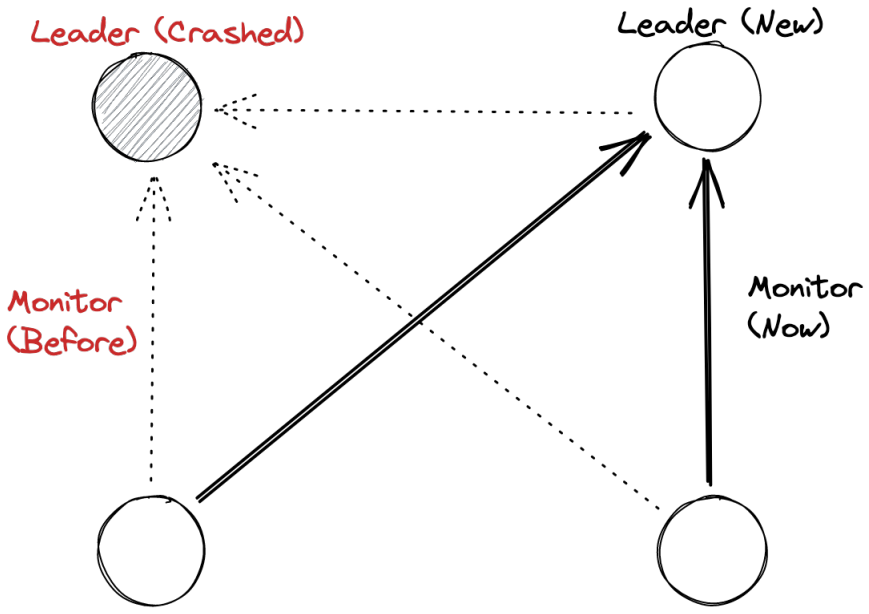

Figure 2: Elect A New Leader

2.3 The Third Version With Election And Crash Tolerance

In the third version, I solved the non-synchronized problem in the file `gms3`. I maintained the *last* message that should be sent to all nodes. If the leader node crashes and the message from it does not sent to every slave node, the next leader will multicast the message. In the third version, message will have a sequence number called `N` which can be used to mark the message in the queue. When slave nodes receive the message, they will compare the received sequence number `I` with the current sequence number `N`. If $I < N$, it means this slave node has already received this message, so this message will be ignored. If $I > N$, it means this slave node needs to update, and deal with this message, so the messaged will be handled.

Therefore, even the leader crashes before it sends the message to all nodes, the group will keep consistent.

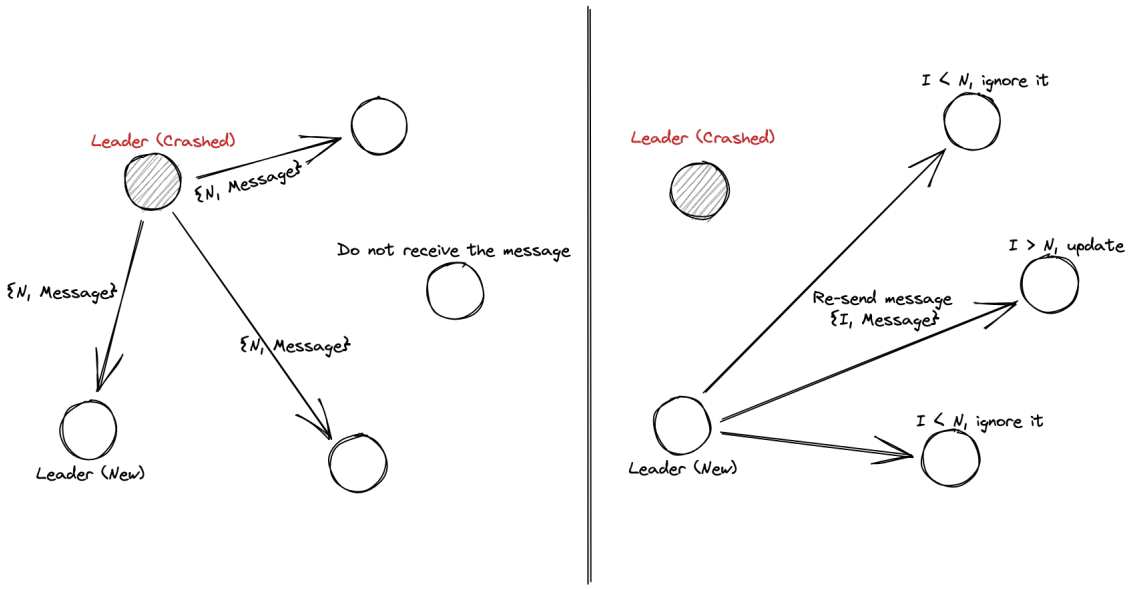

Figure 3: Keep Consistent

3 Evaluation

The evaluation will be shown in the report meeting.

4 Conclusions

In this assignment, I understand multicast better, and also master the skill of the usage of `Erlang` build-in function `apply`.