Report 4: Groupy

Yanghang Zeng

October 5, 2022

1 Introduction

In this assignment, the main task is to implement a distributed system that provides group membership service with atomic multicast. The aim is to have multiple nodes with a synchronized state. The problem in this assignment is to ensure the synchronization of node state when nodes join and leave, and to ensure the atomicity when nodes crash during the multicasting. The important topic related to the distributed system is the coordination. The coordination handles the communication between processes in a distributed system so that a system won't get into a failed state. The code template was given, and the main contribution is complete the failure handling function.

2 Main problems and solutions

The essential problem of a multicasting system is to ensure the failure handling. Specifically, the system should be able to detect node crashes, and to elect a new leader node, and finally ensure the atomic multicast.

The first problem is detecting node crashes. In this assignment, Erlang was used so Erlang built-in support can be utilized. The erlang:monitor/2 function is used to report the crash of a node. Another scenario is a new node that wants to join should detect the death of the leader. Here a waiting mechanism with timeout is applied. If a new node doesn't receive an invitation message from the leader after a certain timeout, it will know that the leader was already dead.

The second problem is handling the crash of a node and electing a new leader. In this case, we assume that the only node to be monitored is the leader. If a slave detects the death of its leader, it will move to an election state. The rule of election is selecting the first node in the group list to be the new leader.

The third problem is ensuring atomic multicasting when a leader node crashed during broadcasting to the group. If a leader crashed before sending a message to all members of the group, it might be the case that part of the

group don't receive the message, while the others do. This cause a problem in synchronization. Here the solution is to keep a copy of the last message seen from the leader, and also add a number to each message. A slave will know the next number is the expected message to receive, otherwise, the message is duplicate and should be discarded. The crucial part is the newly elected leader should resend the last message it received to all peers in the group. This step will help the group to be synchronized.

3 Evaluation

For the verification, the program used GUI to represent different worker nodes, the color inside the window means the state of the worker. By observing the synchronization of colors, we can verify the multicast system.

For the gms2 module, which adds a failure detection without handling. A test with 5 nodes is run. During the test, it is noticed that the system would easily run out of synchronization, i.e., windows have different colors. This is due to the reason described in last section.

After adding a failure handling machenism, the multicating becomes more reliable in gms3 module. By running multiple tests, it is noticed that the system always runs in synchronization.

4 Conclusions

In this assignment, a group membership service system is built step by step. I learned how to implement failure detection and ensure atomicity. The key idea is to keep a sequence number of messages. This implementation is reliable under the assumption that all messages eventually arrive. But considering Erlang doesn't guarantee delivery, extra function to handle lost of messages should be added, according to the optional task.