Groupy: a group membership service

Vasigaran Senthilkumar

October 6, 2021

1 Introduction

In this exercise, the implementation of Groupy, a group membership service that provides atomic multicast is discussed. There are several nodes in the network . Each nodes changes its state along with the change in color of GUI window, and it is multicasted to all the other nodes. All The nodes should keep up with the same color change indicating synchronisation of the state. Fault tolerant mechanisms have to be adopted to handle node failures in order to maintain a consistent state in group communication.

2 Main problems and solutions

Initially,It was very difficult to understand the implementation of the gms1 module. The application process, the worker process , leader , slaves all seemed confusing. After revising the code and using debug lines, I was able to understand the implementation.Every node is separated in two layers, the application layer and the group process layer. The application layer contains master processes for each node and it communicates with the GUI to display the state. It can send messages and receive messages from group process. The group process layer consists of the slaves processes. Slaves redirects the messages from the Master process to the leader. The slaves can also send a message to the leader and the leader can multicast the message to all the members of the group. Each node knows the other nodes of the network by having a list of them and updating it.

The Gms1 was implemented without any failure detection. There is a leader created and the nodes can join the network. Since we assume that there is no failure of the leader, the consistency of state between the nodes is maintained. The Gms2 introduces failure detection of the leader. The slave process monitors the leader using erlang:Monitor. When the monitor detects the leader is 'DOWN', then a new leader is elected. The next node in the list of nodes after the dead leader becomes the new leader.The new view of the group is broadcast ed to the other nodes. But Gms2 had it cons. When a leader dies before multicasting a message to the some of

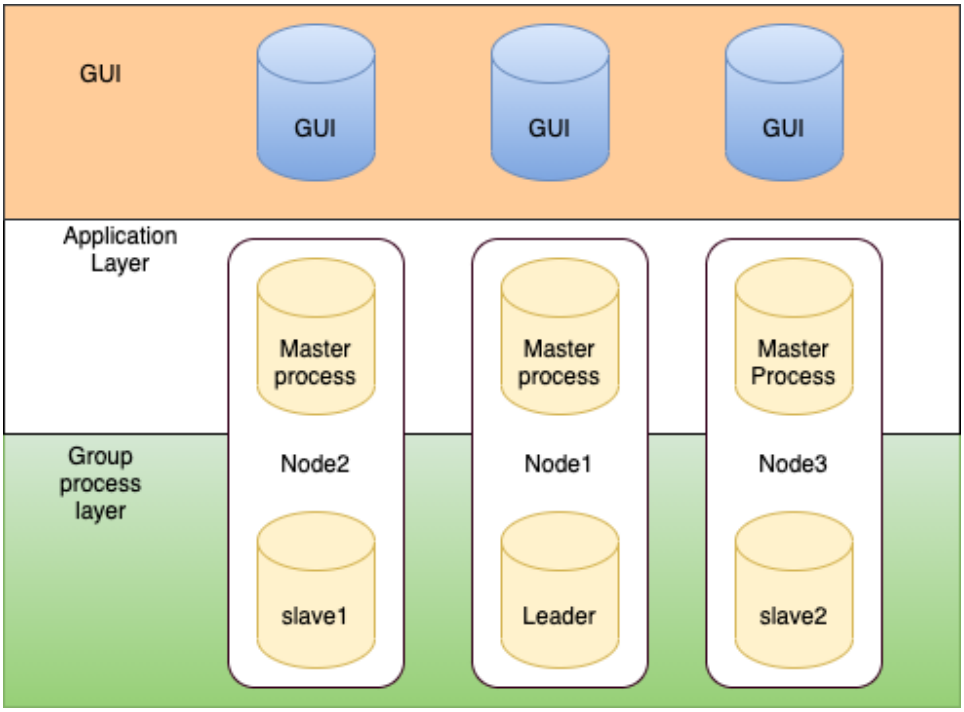

Figure 1: Groupy implementation

Nodes, the consistency of the state is not maintained. So in that scenario, we had to resend the last message from the new leader. So the track of the last message has to be kept. This is implemented in gms3. This will also cause nodes receiving duplicate messages because they might have received the message from the dead leader. So a sequence number is also provided along with each message. Then when the slave receives a message, the sequence number is compared with the expected sequence number. If the sequence number in the message is less than the expected sequence number, the slave has received the message already and can be ignored. Otherwise, the message was missed and should be multicasted to that slave. The sequence number is incremented everytime after the leader multicasts a message and the receiver after receiving a message.

3 Evaluation

For testing and evaluating the modules, I considered 4 slaves and a leader. First gms1 code without the failure detection and reelection of new leader was executed. The group was in consistent state and the colors changed simultaneously in 5 windows. I tried to kill one of the nodes which is the slave and there was synchronisation with the other group members as the leader was still up. When the leader process was killed using exit command, the colors stop changing. Since the leader is dead, there is no one to broadcast the message to the group.

Then the gms2 module was tested with the same test case. Also This time, a random crash was added. This time when the leader crashed, a new leader was reelected. But the state was not in synchronisation, because the dead leader had not multicasted the last message to all the group process.

When gms3 was executed using the same test case, after the leader crash and reelection of a new leader, the state of the slaves are still consistent in the network. The new elected leader resent the messages to the group processes that didn't receive the message when the previous leader died.

4 Conclusions

The exercise was interesting and I managed to understand how a group membership service deliver reliable messages to a group of nodes . Failure detection mechanisms prevent the system collapse even though nodes fail. There are some issues that might still occur in the group membership service. We have assumed that the messages arrive at nodes and not get lost in the middle. In order to address the problem we can either have a holdback queue where we store each message that is being sent and keep track of the sequence number. we can identify if the message is lost by comparing the sequence number of the message with the existing sequence number. In this case if the

sequence number in message is greater than the expected sequence number, the message has been lost. And we have to broadcast the message in the queue.Also we could introduce acknowledgement messages from slaves back to leader but the performance would be impacted.