Report 4: Groupy

Alan Goran

October 4, 2017

1 Introduction

In this assignment, we will create a group membership service where different nodes use atomic multicast to communicate. The goal is to have several application layer processes with a coordinated state, they should all perform the same sequence of color changes. The goal is also to have all the nodes synchronized, even though nodes may crash in the middle of the process.

Simply explained, we will have a set of nodes and one of the nodes is elected as the "leader" and the rest of the nodes will be "slaves". When a node wants to change color, multicast a message, it will send a message to the leader and the leader will then multicast that to all the slaves in the group.

2 Main problems and solutions

The main problem in this assignment was to handle failures in the group membership service. The first failure that needed to be handled was detecting that a process has crashed. This was important because the crashing of a leader led to a situation where communication between the slaves were lost. For this, we used the Erlang built in support which has the ability to monitor a node (the leader node) and if that node dies a message (DOWN) will be received. So by using this indicator message to initiate an election process, a new leader was chosen shorty after that. The next elected leader will always be the first node in the slaves list.

The second failure was also a tricky situation to handle. Assume that the leader of a group of 10 nodes dies in the middle of the multicasting process, where the new state has been sent to 5 of the slave nodes and the other 5 slaves are remained to receive their new state. Obviously, this will lead to an out of sync problem when the new leader is elected and started multicast state changes. To solve this problem, a new implementation was implemented, a reliable multicast implementation. In the new implementation

when a leader crashes there is still an election and a new leader is picked, although instead of letting the new leader multicast new state changes we will first have it multicast the previous messages. All the nodes are able to identify a message that has been seen before and therefore discard a message if the node has already received it. So by always keeping track of the previous message this issue was solved and no matter how many times the leader crashes the nodes will still sync themselves to each other.

3 Evaluation

All of the implementations mentioned above were tested and the results were correct. The nodes are kept in sync and the any crashes that could occur in the middle of a multicasting process is handled. The test result for the final implementation (gms3), with 5 nodes and 1ms sleep, is shown below:

```
Leader 1: crash
New Leader: 2
Still Slaves: 4
Still Slaves: 5
Still Slaves: 3
Leader 2: crash
New Leader: 3
Still Slaves: 4
Still Slaves: 5
Leader 3: crash
New Leader: 4
Still Slaves: 5
leader 4: crash
New Leader: 5
```

4 Conclusions

There are some assumptions that we take in this assignment and those are; assuming that the monitor works perfectly and that it will never report a death message when the node is in fact alive, and we also assume that the message that tells us about the death of a node is the last message received from that node. Additionally, we are strongly relying on the assumption that all the messages between the nodes are all delivered and received by the intended node. We are not taking lost messages in to account.

These assumptions are quite unreliable, so to have a more robust system we will need to come up with solutions to handle the cases where the assumptions above don't hold. For example, to guarantee that the messages

are delivered we can have the slaves sending back a confirmation message to ensure that they have gotten the message. And if one node didn't we can send the message again.

To conclude, this assignment is a very simplified version of the the multicast system that it represents. In reality, all system crashes and good counter solutions to the crashes is always needed.