Report 4: Groupy Seminar Report

Alberto Lorente Leal

27 September 2011

1 Introduction

In this seminar, we implement a group membership service with atomic multicast. We get in contact with a scenario in which we are interested that the nodes participating in the group are synchronised and coordinated correctly. In this case we simply have an application layer which simply shows different colors which change accordingly depending with the information by the messages sended by the leader in this case. We get the opportunity to see how we can implement this by following some of the guidelines presented in class related to coordination algorithms and Multicast.

2 Task:

The main task we had to implement groupy through 3 different implementations which showed how the system reacted as we added the synchronization algorithm and later on the reliable multicast on the system for our group membership system.

2.1 First Implementation gms1:

Following the guidelines in the assignment we managed to run the first implementation which had simply the functionality to add the nodes to group with a leader running in it with the application Gui showing different colors. Each color showed a new different state which made all the nodes of the group to be constantly changing colours. Doing some testing, we saw that if having various nodes running, killing the leader made the rest of the nodes to stop reacting. This showed that the system lacked some way to select a new leader when this one crashed.

2.2 Second Implementation gms2:

The next stop now was to modify the system so it could detect crashes and in case this event affected the leader, we would choose a new leader from the list of peers. In this case we selected as a new leader as the next node that

appeared next to the leader in the list of peers. To detect this we used the crash detectors from the monitors in erlang support in order to track this in the leader which is the node which interests us most. Also we implemented some sort of random crash using a random variablethat made the nodes to crash randomly.

After that, we did some experiments by using a testing module where we launched several nodes in the group to see how the system reacted. We did a test in which we closed the leader first when all the nodes were up and running and we saw that the rest of the nodes lost synchronisation while we were choosing a new leader. On the other hand when a slave crashed the synchronisation wasnt lost and the nodes were following the rhythm given by the leader.

2.3 Third Implementation gms3:

The final implementation we had to developed for groupy a reliable multicast to solve the synchronisation issues we had when the leader crashed. In this case we keep some history of the last message or view received in order to have some sort of way to restore the synchronisation of the group when we selected a new leader. Also we added the concept of sequence number in the messages so the slaves could keep a track of the messages and remove the duplicate messages of the system. In this case we launched the system with our test module and saw how the nodes after a while started to crash and the system kept the synchronisation without any issues.

```
%% Author: alberto
%% Created: 24/09/2011
%% Description: TODO: Add description to test -module(test).

-export([start/0]).
start()->
        A=worker:start(leader,gms3,13,800),
        B=worker:start(b,gms3,13,A,800),
        C=worker:start(c,gms3,13,B,800),
        D=worker:start(d,gms3,13,C,800),
        E=worker:start(e,gms3,13,D,800),
        F=worker:start(f,gms3,13,E,500),
        G=worker:start(g,gms3,13,F,500),
        H=worker:start(h,gms3,13,G,500),
        I=worker:start(i,gms3,13,H,500),
        J=worker:start(i,gms3,13,I,500),
        K=worker:start(i,gms3,13,J,500),
        L=worker:start(i,gms3,13,K,500),
        M=worker:start(i,gms3,13,L,500),
        N=worker:start(i,gms3,13,M,500),
```

```
O=worker:start(i,gms3,13,N,500).
```

3 What could possibly go wrong:

In this section we address some questions about the implementation of groupy gms3 related to possible things that could go wrong with this implementation.

- How would we have to change the implementation to handle the possibly lost messages? How would this impact performance?

  We could address this issue if we implemented some sort of acknowledgement mechanism in order to check that the messages are received correctly. The problem with this implementation is that it will possibly increase the delay with which we receive the messages as in this case we need to wait that we receive an ACK before we send the next message.

- The second reason why things will not work is that we rely on that the Erlang failure detector is perfect i.e. that it will never suspect any correct node for having crashed. Is this really the case? Can we adapt the system so that it will behave correctly if it does make progress even though it might not always make progress?

  In reality we cant suspect that the failure detector is perfect because in asynchronous systems we cant determine if the system has crashed completely as it is possible that in real life we could have messages that arrive later due to a congestion in the network. This could make us believe that the node crashed and that could be not the case. We could monitor somehow the queries with some sort of timer in order to catch the timeout values due to network congestion. This could gives some sort of accurate hint of a node crashing and select the new leader on this values. If there is the case that in reality the node is still alive, we could have the situation where 2 leaders are working. Some sort of solution is to make the old leader accept the new one, making it change back to a slave.

- The third reason why things do not work is that we could have a situation where one node delivers a message that will not be delivered by any correct node. This could happen even if we had reliable send operations and perfect failure detectors. How could this happen and how likely is it that it does? What would a solution look like?

  One possibility is that one of the messages gets corrupted during it transport making it impossible for the nodes to deliver or one of the nodes starts to behave strangely (due to node stability issues possibly).

Therefore it would start acting strangely and send messages that do not correspond to the program protocol. Those messages would not fit in the state parameters of the other nodes and so those messages will never be delivered. Also we see that the nodes appears to still be functional for the rest of the group but in reality it is not working correctly. This problem will probably pass by and in the end it could make the rest of the nodes to start behaving erratically due to this messages.

4 Conclusions

Thanks to this groupy assignment, we get to know how is it possible to address the problems related to coordination and agreement using some approaches based in the theory we have seen in class. We see how a reliable multicast works in a possible scenario and also we get to see how it is possible to synchronise the nodes using techniques based on the implementation of coordination algorithms. This makes us realise the difficulties that arise with the problems related to the coordination of multiple nodes in a distributed system.