Report 4: Groupy

Pierre FLEITZ

October 7, 2015

1 Introduction

The main goal of this seminar was to implement a group membership service, composed by several members.
In this group membership, each member has a state which is its color. On a random way, a member can generate a new color and multicast it to all other members in the group for updating. All the members have to be synchronized, i.e all the members GUI have to display the same color at the same time. We can have a really good representation of what's happening during our tests because of the choice of the state (a color) and the GUI.

2 Main problems and solutions

In order to ensure synchronisation between members, each message generated by a member is sent to all others members in the group, through the leader.

3 Chapter 2 : The first implementation

At the beggining, we have implemented (the code is given mostly) an implementation named gms1 who does not support fault-tolerance. In our a problem we represent a group by a leader and several slaves. When a slave receives a message from the application layer, it transmits the state to the leader that multicasts the message to all other members of the group. This is the reason why we can speak about synchronization: all the members receive the message for updating their colors (Assuming that there is no failures on the network).

To test that behaviour, we create one leader and different slaves. Their colors were synchronized so it meant that the implementation was correct. However, if the leader crashes, the two other slaves stop displaying colors because there is not a re-election of a new leader. During this test we have been able to try the test module with some freeze, go, and stop message.

Note : in the subject the given code is wrong with a second go(Wrk) instead of a stop(Wrk).

4 Chapter 3 : Handling failure

4.1 Module gms2 : Detection of a leader crash

Then we did a second implementation named gms2. In this one the point was to add the detection of the leader crash for the slaves. When it happens, the leader crashes, there is a re-election procedure who's launched. We've been able to do some test after implemented that and we can see that when leader crashes, other members continue to display different colors, and this is due to the re-election of a new leader.

During this test, if we have for example one leader with 2 slaves, we can notice that when the leader dies, the slaves become out of synchronization. Indeed we can notice than the second slave doesn't display the same color than the first slave. I think that we can explain that issue by taking a closer look at the bcast/3 method : the leader have sent a message to slave 1, then crashes - so, slave 2 didn't receive any message from the leader =¿ As a consequence, the slave 1 has received a message than slave 2 didn't receive, so they went out of synchronization.

4.2 Module gms3 : Reliable multicast

Finally, in order to avoid this out-of-synchronization-problem, we implemented a third solution named gms3. In this implementation each message has a number and each member contains in his state, the last message he has seen. Therefore, if the multicast of a message fail, the new leader will send again the message to all the members. To avoid duplicate messages, we check if the number of the message received is not inferior to the number of the last message seen. To do so, we use the when construction. For msg, I, $received, when I < N then the message is rejected.$

When we test gms3 with different slaves and that the leader crashed, we don't have problems of synchronization anymore : we see that because all the beautiful colors are still synchronized between all members ! *Yay*

5 Conclusions

With this seminar we saw how to manage a group membership service to deliver reliable messages to a group of nodes. At the end and after three different implementations the synchronization works, even if the leader crashes. On a personal point of view, I think that this seminar was really interesting

and well-explicated, the tests were really effective as there was a graphic side (beautiful colors mentionned before). I thought it was quite interesting to take a look at gui.erl to see how graphic contents work in Erlang. Note 1 : I am still not a huge fan of LaTeX. Note 2 : You should add a "Be careful don't do that seminar or don't play with the tests too much if you are epileptic." - I tried to synchronize a beat song with something like 20 slaves, didn't managed to do it but when everything went wrong it was quite epic. (I took a video it was fun) Note 3 : It is awesome to see how much we improved our Erlang skill in such a little amount of time, the philosophy behind the language is more and more clear for me..