Seminar Report
Groupy - A group membership service

Thomas FATTAL

September 27, 2012

1 Introduction

The goal of this exercise was to implement a group membership service, composed of several members. Each member has a state which is its color. Randomly, 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.

2 Main problems and solutions

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

2.1 Without fault-tolerance

First, an implementation without supporting fault-tolerance has been implemented. A group is composed of 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. That's why, we can speak about synchronization: all the members receive the message for updating their colors (if we imagine that there is no failures on the network).

To test that behaviour, we create one leader and two slaves. Their colors were synchronized so it means that the implementation is correct. However, if the leader crashes, the two other slaves stop to display colors because there is not a re-election of a new leader.

2.2 Detection of leader crash

A second implementation has been made. We added the detection of the leader crash for the slaves. When the leader crashes, there is a re-election procedure. When testing, we can see that when leader crashes, other members continue to display different colors, due to the re-election of a new leader.

If we insert a crash simulation when the leader multicasts the message to other nodes, we can notice that when the leader dies, the slaves became out of synchronization.
With one leader and two slaves, we can notice than the second slave doesn't display the same color than the first slave. We can explain that problem by watching at the bcast/3 method. The leader have sent a message to slave 1, then crashes. So, the 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.

2.3 Reliable multicast

To avoid this synchronization problem, we implemented a third solution. Each message has a number and each member contains in his state, the last message he has seen. If the multicast of a message failed, 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. If so, the message is rejected.

With that implementation, we don't have problems of synchronization when the leader crashes : all the colors are still synchronized between all members.

3 Conclusion

In conclusion, we saw how we can manage a group membership service to deliver reliable messages to a group of nodes. The synchronization works, even if the leader crashes.
This exercise was also very interesting, from the Erlang point of view, because it was a first introduction to graphical windows in Erlang.