Report 4: Groupy Group membership service

Simon Cathébras

September 26, 2012

1 Introduction

During this assignment we have implemented a group membership service. This particular device stands for an important problematic in distributed systems: Failure handling.
We could easily realise that implementing such a mechanism is kind of complex.

1.1 Context

All distributed systems are actually very dependant of the failure handling. As a matter of fact, this failure handling must be a transparency one for users. It means that the application must (in most of cases) continue running even if a node crashes.
For example can we take a cluster of server. If one server goes down, the cluster must continue to run.

1.2 What have been realised

What we realised so far in the development was the failure handling mechanism part. To do so, we have implemented first a basic disconnection handling mechanism: when the leader is disconnected, another leader is elected.
The thing is, if we now add a random "crash" of the leader node, several messages may not be fully transmitted to all of the slaves nodes. In order to do so, we add to implement a message identification (basically, each message now have a message number). This message identification is used in order to be able to resend messages, and eventually discard the wrong ones.
For now, we can successfully create groups and add some extra nodes to those groups. If a slave node crashes, the system is still running. And if the leader crashes, a new leader is successfully elected.

2 Main problems and solutions

2.1 Missing messages

When a leader crashes, a particular problem occurs. The leader can crash in the middle of a broadcast. Thus, it leads to de synchronisation of all nodes. The thing is, several nodes are receiving the message, and others don't. This means that the the next leader will certainly receive it, but the others won't.
This has been fixed with the following: The new leader automatically send first the last message it received, and then the others are checking each message they are receiving and discard message they have already received.

2.2 Message identification

Managing the identification of messages was a tricky part of this assignment. According to instructions:

- Slave: N in parameter stands for the next message the node expects to receive.
- Leader: The next message to send.

We could not easily decide when we were supposed to change the value. So we found a solution working with the following.

Slave When `msg` or `view` is received, we match the message value with `N`, and we expect the next message to arrive is `N+1`

Leader We send the next message with the stamp of `N` and we loop with `N+1`

Election For the new leader, we send the last message, we send the new broadcast with the message id `N` and we call leader with `N+1`. For the slave, we wait for the message `N`.

3 Conclusion

During this assignment, we discovered one way amongst others to solve the failure handling problem. But, our implementation still doesn't work very well. As a matter of fact, there are some troubles for example with the management of message ID. what if the counter explodes for example ?