Report 4: Groupy - a group membership service

Bernardo González Riede

October 5, 2017

1 Introduction

The purpose of the assignment is to show an implementation of atomic multicasts. A multicast is often implemented via a layer on top of membership management and refers to a broadcast where not all receivers are intended receivers. Using a multicast in a view which contains all nodes results in a multicast indistinguishable from a broadcast. Hence the namig the proceduro atomic broadcast or atomic multicast is sometimes used interchangeably. It's regarded as atomic since it will either be delivered to every correct node or all nodes will abort without side effects.

2 Main problems and solutions

2.1 Complexity of the code

The complexity of the code rised substantially from the previous assingment, loggy. It was difficult to keep track of the flow of the programm without having it read extensively. To be able to understand it I had to write & draw the flow of the processes, starting from the test module.

2.2 observations

- After finishing gms2 and introducing just the crash, the non-candidates to being leader would crash if the leader crashed, while the candidate aquired leader status.
- Using the test:more/4 to create >7 nodes and a crash value of 10% resulted in all nodes disappearing instantly.

3 Going further

The implementation, in it's current state, handles messages based on sending and thus receving, a message at most once. A lost message won't be resent,

which could create problems if a node doesn't receive a new view in the following case:

- The node is the next in line to be Leader
- The leader dies immediatly aftewards

This would have an impact in the new leader not having a complete view, therefore not sending messages to a newly joined member.

As always, there are multiple approaches to remedy this problem:

1. Storing a history of messages
2. Implementing an acknowleged response by the slaves

1 has a problem with performance impact. The more nodes exists, the more possible messages are needed to be stored. Implicitly, if we receive a new message from the node, the node has received the previous message. This implies to store all messages after the oldest heartbeat (in form of a message). Second, since asynchronous networks don't have a upper bound on latency, a timeout has to be implemented. Otherwise all history has to be logged since it's not possible to know if a resent request will arrive, the nodes dead or has received the message. Third, difficulties arise when the leader dies in the same moment the node asks for a message to be resent again.

2 creates a large overhead, since every message will include a response, effectively doubling the amount of messages. Additionally, the leader would need to keep track of who has received the message and who hasn't, which in turn will make it more difficult to elect a new leader.