Groupy - a group membership service

Mattias Cederlund, mcede@kth.se

October 8, 2014

1 Introduction

The task was to implement a group memership service that provides atomic multicast to keep several application layer processes in a coordinated state. When a process wishes to change state it must multicast it to the rest of the group. The multicast layer provides total order, making sure all nodes will be synchronized.

The objective of the assignment was to gain knowledge in how to achieve a coordinated (synchronized) state in a distributed system.

2 The initial implementation

In the initial given implementation of the group membership service the first process started becomes the leader and the rest becomes slaves. When testing I initially ran into some problems getting the implementation running. Firstly there was some confusion about what to run, and then the next problem was about what arguments to provide the worker with. It seemed like the worker would only accept numbers for Id. Another problem was that I initially never managed to start more than the first worker from the command line and passing the process identifier to the next, so I implemented a simple test module to start the workers. Lastly the GUI didn't update when receiving color updates, so I had to add wxFrame:refresh(Window) at the end of color(Window, Color) for it to work.

After I figured everything out I managed to get a few workers running and they synchronized without any problems.

3 Failure handling

To handle failure we set all workers to monitor the leader and continue to an election state if the leader dies. The election is just selecting the first node in the list of slaves. If the node finds itself at the first position in the list it will step up as leader, otherwise it will continue as a slave.

When testing the system after adding the risk of leader failure when sending multicasts the nodes (yet after quite a long time with 1/100 chance of crashing) eventually got out of synch. What's happening is that some multicast messages are sent to the nodes before crashing and some nodes were yet to receive their messages. In my test I was running 4 workers, and after the leader crashed two remained in synch and one was off synch.

4 Reliable multicast

Assuming that message delivery is reliable and that if two messages are sent after each other and the second one is delivered, then the first would also have been delivered we can make the multicaster reliable by saving the last message and having the new leader resending it if the previous leader dies. To remove duplicate messages both the leader and slaves save a counter with the value expected of the next message. If a message number is lower than the saved number, the message is a duplicate and gets ignored. The only thing I had to do was implement these two mechanisms and everything should work! That rather simple task only consisted of adding the extra parameters to the nodes and messages and I succeeded without any trouble.

After running the same test for a while with leaders dying everything stayed synchronized. After some research I also managed to start workers from the command line and adding new nodes after a leader died was no problem.

In Erlang we don't have any guarantee that messages actually arrive, but we know they are delivered in FIFO order if they are delivered. The nodes also knows what message number to expect next, so if a message is lost and a node receives a message with a higher number it could request to get all messages it missed. Although that would require the leader to keep a small history of sent messages. The list woudln't need to be too big since messages are not lost so frequently and it would barely affect performance. Another way to achieve reliable message passing would be to use acknowledgements, but that could affect performance a lot by increasing the number of messages

sent to twice as many.

As I don't know how monitors are implemented I can only guess, but if we rely on monitors and have a congested network where messages passed between the processes get delayed I bet the monitor might think the process is dead when it's actually not. If it happens and the group membership service elected a new leader, the previous leader may realize there is a new leader and ask to rejoin the group.

I did not come up with any answer to the questions on the third reason why things would not work because of messages sent by uncorrect nodes. Actually I'm not even sure I understand what the problem is.

5 Conclusions

Thanks to this assignment I learned some of the available techniques and possible pitfalls when coordinating nodes in a distributed system. I think this assignment was a good way of demonstrating just how easy nodes get unsynchronized and how to prevent it. Although, I expected more coding to be done in this assignment.