Seminar Report: Groupy

Pedro Garcia Mota, Pau Espin Pedrol

April 26, 2013

1 Introduction

Introduce in a couple of sentences the seminar and the main topic related to distributed systems it covers.
The aim of this seminar is to implement a group membership service that provides atomic multicast, in order to have several processes all performing the same sequence of state changes.

2 Work done

Present the new code that you have added or how you have implemented a required functionality by using small Erlang code snippets (you do not need to copy&paste all the code).

2.1 gms1

```
leader(Id, Master, Peers) ->
    receive
    {mcast, Msg} ->
      bcast(Id, {msg, Msg}, Peers), %% TODO: COMPLETE
      Master ! {deliver, Msg},
      leader(Id, Master, Peers);
    {join, Peer} ->
      Master ! request,
      joining(Id, Master, Peer, Peers); %% TODO: COMPLETE
    stop ->
      ok;
    Error ->
      io:format("leader ~w: strange message ~w~n", [Id, Error])
  end.
```

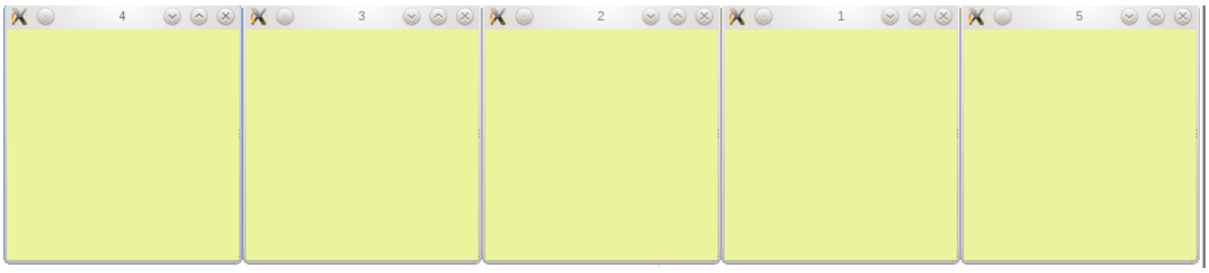
```
slave(Id, Master, Leader, Peers) ->
  receive
    {mcast, Msg} ->
      Leader ! {mcast, Msg},
      slave(Id, Master, Leader, Peers);
    {join, Peer} ->
      Leader ! {join, Peer},
        slave(Id, Master, Leader, Peers);
    {msg, Msg} ->
        Master ! {deliver, Msg},
        slave(Id, Master, Leader, Peers);
    {view, _, _, View} ->
        slave(Id, Master, Leader, View);
    stop ->
        ok;
    Error ->
        io:format("slave ~w: strange message ~w~n", [Id, Error])
  end.
```

2.2 gms2

2.3 gms3

3 Experiments

- Experiments 1
  - Do some experiments to see that you can create a group, add some peers and keep their state coordinated. You can use the following code to start and stop the whole system. Note that we are using the name of the module (i.e. gms1) as a parameter to the start procedure.

  - Split the groupy module and make the needed adaptations to enable each worker to run in different machines. Remember how names registered in remote nodes are referred.
- Experiments 2
  - Do some experiments to see if the peers can keep their state coordinated even if nodes crash.

    We run different erlang processes in parallel in different terminals and once they are all working, we kill the leader process and we see the overall system keeps going fine.
  - Repeat the experiments and see if you can have the state of the workers become out of synch

    Yes, they become out of synch if some process crashes in the middle of the multicast send, as they don't check atomicity (all processes or none of them receive the message).

- Experiments 3
  - Repeat the experiments to see if now the peers can keep their state coordinated even if nodes crash.

    It seems it's working correctly.
  - Try to keep a group rolling by adding more nodes as existing nodes die.

    It works too.

4 Open questions

Open Questions 1

- gms2
  - Why is this happening?

    This happens if some process crashes in the middle of the multicast send, as they don't check atomicity (all processes or none of them receive the message). This means that some processes get extra messages which make its steps different and so their colours change.
- gms3
  - How would we have to change the implementation to handle the possibly lost messages?

    We could add code in the slave to check if received N is bigger than last known local N, and if it's true, that means some message has been lost. We could then add some buffering/queue mechanism in all nodes (as they are future leader candidates) so they store last N messages (to be N-fail tolerant). When some slave detects Nleader¿Nlocal, it can request the leader to send back missinggmessages.
  - How would this impact performance?

    As the queue/buffer grows the costs of the operations over it (looking for elements, removing elements, etc) will grow as well. The memory used will also be affected by the number of elements that will be stored in it, obviously N should be a finite number.
  - Is really the case that we will never suspect any correct node for having crashed?

    No, the functioning of the monitor process is not perfect.

5 Personal Opinion

Your personal opinion of the seminar, including whether it should be included in next year's course or not.
The seminar should be included in the next course as it expands the concepts of messaging between processes seen in the last seminar (Toty) and helps to solidify some concepts of the Erlang programming language