Seminar Report: Muty

Pedro Garcia Mota, Pau Espin Pedrol

April 3, 2013

1 Introduction

Introduce in a couple of sentences the seminar and the main topic related to distributed systems it covers.

The task is to implement a total order multicast service using a distributed algorithm. The algorithm is based on requesting proposals from all nodes in a group.

2 Work done

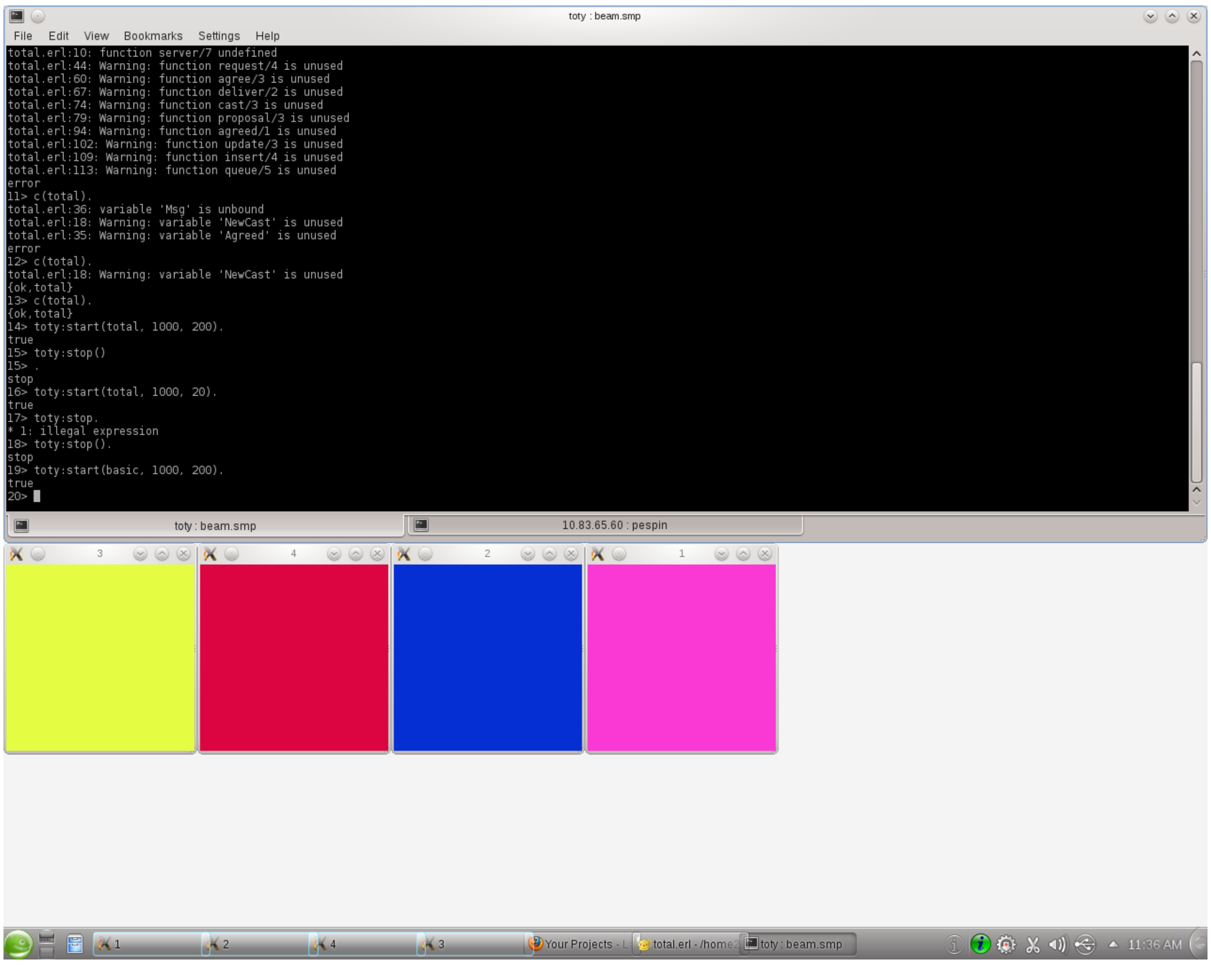
Present the new code that you have added or how you have implemented a required functionality by using small Erlang code snippets (you do not need to copy&paste all the code).

```
server(Master, NextPrp, MaxAgr, Nodes, Cast, Queue, Jitter) ->
    receive
    {send, Msg} ->
      Ref = make_ref(),
      request(Ref, Msg, Nodes, Jitter),
      NewCast = cast(Ref, Nodes, Cast),
      server(Master, NextPrp, MaxAgr, Nodes, NewCast, Queue, Jitter)
        ;
    {request, From, Ref, Msg} ->
      From ! {proposal, Ref, NextPrp},
      NewQueue = insert(Ref, Msg,  NextPrp, Queue),
      NewNextPrp = seq:increment(seq:max(NextPrp, MaxAgr)),
      server(Master, NewNextPrp, MaxAgr, Nodes, Cast, NewQueue,
         Jitter);
    {proposal, Ref, Proposal} ->
      case proposal(Ref, Proposal, Cast) of
        {agreed, MaxSeq, NewCast} ->
          agree(Ref , MaxSeq, Nodes),
          server(Master, NextPrp, MaxSeq, Nodes, NewCast, Queue,
             Jitter);
        NewCast ->
          server(Master, NextPrp, MaxAgr, Nodes, NewCast, Queue,
             Jitter)
        end;
    {agreed, Ref, Seq} ->
```

```
      Updated = update(Ref, Seq, Queue),
      {Agreed, NewQueue} = agreed(Updated),
      deliver(Master, Agreed),
      NewMaxAgr = seq:max(Seq, MaxAgr),
      server(Master, NextPrp, NewMaxAgr, Nodes, Cast, NewQueue,
          Jitter);
    stop ->
      ok
  end.
```

3 Experiments

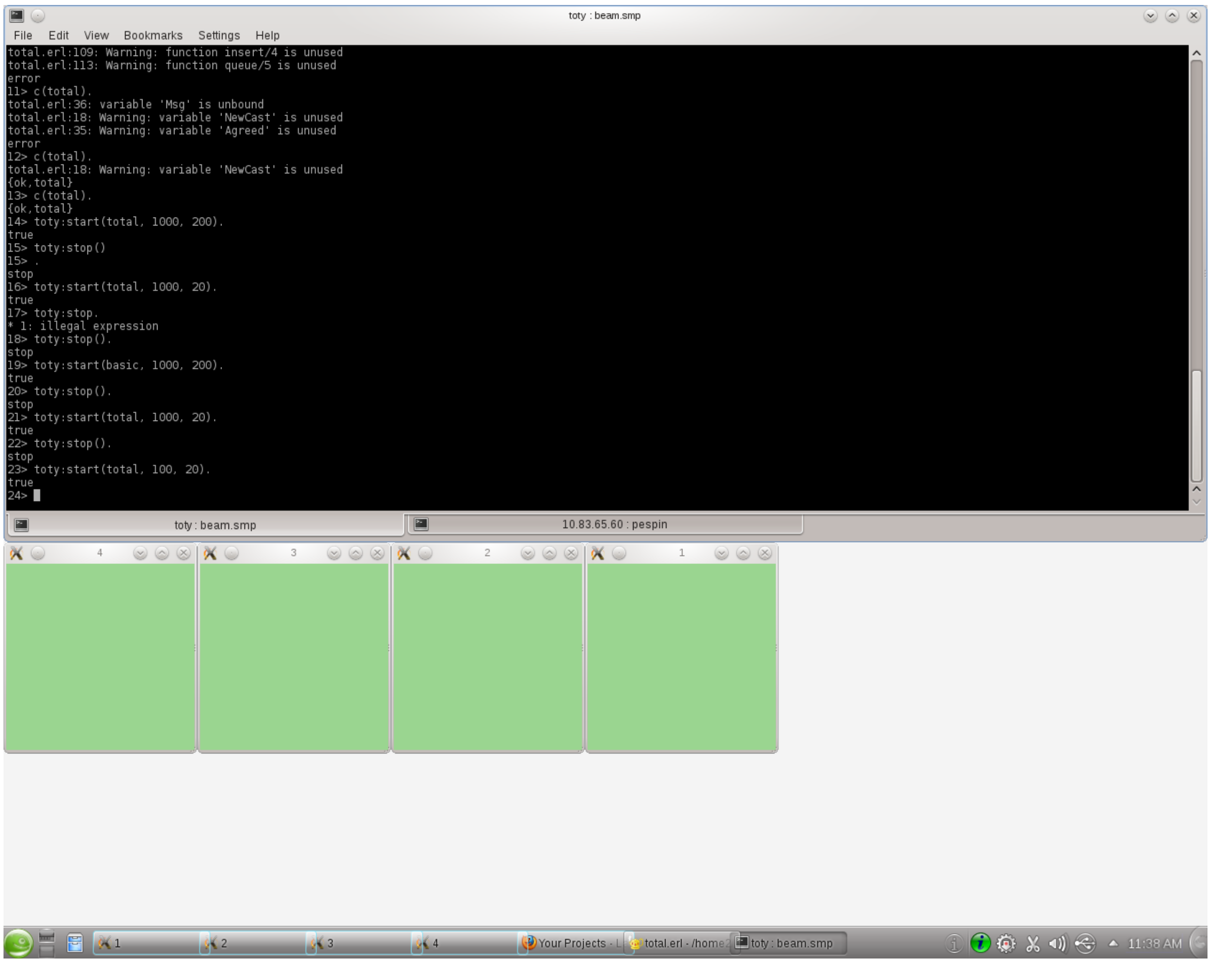
- *BASIC - Set up the basic multicast system, and use the following test program to experiment with different values for Sleep and Jitter and see when messages are not delivered in total order. Note that we are using the name of the module (i.e. basic) as a parameter to the start procedure. We will easily be able to test different multicast implementations.*



```
total.erl:10: function server/7 undefined
total.erl:44: Warning: function request/4 is unused
total.erl:60: Warning: function agree/3 is unused
total.erl:67: Warning: function deliver/2 is unused
total.erl:74: Warning: function cast/3 is unused
total.erl:79: Warning: function proposal/3 is unused
total.erl:94: Warning: function agreed/1 is unused
total.erl:102: Warning: function update/3 is unused
total.erl:109: Warning: function insert/4 is unused
total.erl:113: Warning: function queue/5 is unused
error
11> c(total).
total.erl:36: variable 'Msg' is unbound
total.erl:18: Warning: variable 'NewCast' is unused
total.erl:35: Warning: variable 'Agreed' is unused
error
12> c(total).
total.erl:18: Warning: variable 'NewCast' is unused
{ok,total}
13> c(total).
{ok,total}
14> toty:start(total, 1000, 200).
true
15> toty:stop()
15> .
stop
16> toty:start(total, 1000, 20).
true
17> toty:stop.
* 1: illegal expression
18> toty:stop().
stop
19> toty:start(basic, 1000, 200).
true
20>
```

The colour of each worker's window is dependant on the order of the messages received, as the colour is modified by the integer value received in those messages. If messages are not necessarily received in the same order, which is the case with the Basic system, colours on different windows may be different at any given time. If the jitter time increases, the divergence in colour should be greater.

- *TOTAL - Set up the total order multicast system, and repeat the previous tests. Does it keep workers synchronized?*



```
total.erl:109: Warning: function insert/4 is unused
total.erl:113: Warning: function queue/5 is unused
error
11> c(total).
total.erl:36: variable 'Msg' is unbound
total.erl:18: Warning: variable 'NewCast' is unused
total.erl:35: Warning: variable 'Agreed' is unused
error
12> c(total).
total.erl:18: Warning: variable 'NewCast' is unused
{ok,total}
13> c(total).
{ok,total}
14> toty:start(total, 1000, 200).
true
15> toty:stop()
15> .
stop
16> toty:start(total, 1000, 20).
true
17> toty:stop.
* 1: illegal expression
18> toty:stop().
stop
19> toty:start(basic, 1000, 200).
true
20> toty:stop().
stop
21> toty:start(total, 1000, 20).
true
22> toty:stop().
stop
23> toty:start(total, 100, 20).
true
24> 
```

Yes, it does. The colours shown on all windows are always the same, as we can see in the image below. The Total implementation ensures that all multicast messages are ordered the same way by all workers.

- *TOTAL - We have a lot of messages in the system. Quantify the number of messages needed to deliver a multicast message and how this depends on the number of workers and try to calculate how many messages we multicast per second while increasing the number of worker.*

Let's suppose we have N nodes. Each node that wants to send a multicast message, needs to send N requests first. Then, the N nodes need to answer the request with a proposal, and the first then needs to send back the agreed value to all nodes. Note we send N messages each time instead of N-1 messge because of implementation details (nodes send messages to themselves too). We have a total of 3·N messages per multicast message.

Calculate how many message we multicast: Let's suppose computing time and underlying network time as negligible, and use our own latency (intrdouced by Sleep calls in our code). Let's assume the worst case, which means that each multicast message needs Jitter·N time to send the requests, and as the other parts are negligible, that's the total time a multicast message needs to be sent successfully. One process sends each message after sleeping for 1 seconds, so each worker sends multicast messages each (Sleep·1000 + (Jitter·N·1000)) seconds. As we have N nodes, N multicast messages are send through that time, as all workers are doing the same. that means that the throughtput can be calculated with the expression below:

$$\text{Throughput} = \frac{N}{(Sleep + N \cdot Jitter) \cdot 1000} \; [\frac{MulticastMsg}{s}]$$

Or if preferred, in actual underlying messages needed to send each multicast message successfully:

$$\text{Throughput} = \frac{3N^2}{(Sleep + N \cdot Jitter) \cdot 1000} \; [\frac{Msg}{s}]$$

4 Open questions

There are no open questions.

5 Personal Opinion

Your personal opinion of the seminar, including whether it should be included in next year's course or not.

The seminar was quite easy compared to the previous two, we got it working at the first try, but it helped us understand the mechanics of total order multicast communication. It should be included in the next course for sure.