Seminar Report: [seminar ID] (e.g. Paxy)

Name of team members

September 21, 2022

Upload your report in PDF format.
Use this LaTeX template to format the report.
A compressed file (.tar.gz) containing all your source code files must be submitted together with this report.

1 Introduction

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

2 Code modifications

In this section we are going to show the code introduced in order to make the algorithm work.

2.1 Proposer.erl

```
1       %Provided code
2
3       round(Name, Backoff, Round, Proposal, Acceptors,
    PanelId) ->
4       io:format("[Proposer ~w] Phase 1: round ~w proposal
    ~w~n",
5                 [Name, Round, Proposal]),
6       PanelId ! {updateProp, "Round: " ++ io_lib:format("
    ~p", [Round]), Proposal},
7       case ballot(Name, ..., ..., ..., PanelId) of
8       {ok, Value} ->
9         {Value, Round};
10      abort ->
11        timer:sleep(rand:uniform(Backoff)),
12        Next = order:inc(...),
13        round(Name, (2*Backoff), ..., Proposal, Acceptors
    , PanelId)
14
15 ------------------------------------------------------------
16      %Changed code
```

```
17
18      round(Name, Backoff, Round, Proposal, Acceptors,
    PanelId) ->
19        io:format("[Proposer ~w] Phase 1: round ~w proposal
     ~w~n",
20                  [Name, Round, Proposal]),
21        PanelId ! {updateProp, "Round: " ++ io_lib:format("
    ~p", [Round]), Proposal},
22        case ballot(Name, Round, Proposal, Acceptors,
    PanelId) of
23
24          %Consensus, return {Value, Round}
25          {ok, Value} ->
26            {Value, Round};
27          abort ->
28            timer:sleep(rand:uniform(Backoff)),
29            % Try again after sleeping, increment round and
    sleeptime
30            Next = order:inc(Round),
31            round(Name, (2*Backoff), Next, Proposal,
    Acceptors, PanelId)
32        end.
33
```

In this part of the code, we can see what we have introduced to make *round* function work. The first change are the parameters added to the case statement, which are the *Round, proposal and acceptors*. The next modification is in case we recieve an *abort*. As observed above, *Next* variable is defined as the increment of *Round* and we use this new incremented variable as *Round* in the recursion calling *round* function.

```
1
2 ballot(Name, Round, Proposal, Acceptors, PanelId) ->
3   prepare(..., ...),
4   Quorum = (length(...) div 2) + 1,
5   MaxVoted = order:null(),
6   case collect(..., ..., ..., ...) of
7     {accepted, Value} ->
8       io:format("[Proposer ~w] Phase 2: round ~w proposal
     ~w (was ~w)~n",
9                 [Name, Round, Value, Proposal]),
10      % update gui
11      PanelId ! {updateProp, "Round: " ++ io_lib:format("
    ~p", [Round]), Value},
12      accept(..., ..., ...),
13      case vote(..., ...) of
14        ok ->
15          {ok, ...};
16        abort ->
17          abort
```

```
18        end;
19      abort ->
20        abort
21    end.
```

```
1
2  ballot(Name, Round, Proposal, Acceptors, PanelId) ->
3    % Send prepare message with round information
4    prepare(Round, Acceptors),
5    % Necessary votes
6    Quorum = (length(Acceptors) div 2) + 1,
7    MaxVoted = order:null(),
8    % Quorum vamos haciendole -1 hasta llegar a 0
9    case collect(Quorum, Round, MaxVoted, Proposal) of
10     {accepted, Value} ->
11       io:format("[Proposer ~w] Phase 2: round ~w proposal
       ~w (was ~w)~n",
12                 [Name, Round, Value, Proposal]),
13       % update gui
14       PanelId ! {updateProp, "Round: " ++ io_lib:format
      ("~p", [Round]), Value},
15       % We got promised, lets ask for votes
16       accept(Round, Value, Acceptors),
17       case vote(Quorum, Round) of
18         ok ->
19           {ok, Value};
20         abort ->
21           abort
22       end;
23     abort ->
24       abort
```

For the *ballot* function we need to add *Round and Acceptors* as parameters for the *prepare* function. With this function we send the prepare message with the round information. In the next step, we need is calculate the necessary votes, which are calculated as the number of Acceptors divided by 2, plus 1. The parameters of *collect* function must be *Quorum, Round, MaxVoted, Proposal*. In case of this function returns an accept we call the *accept* function with *Round, Value, Acceptors* values as parameters. Following that function, we call *vote* fuction inside of a case statement, in case we recieve an ok we return *{ok, Value}*.

```
1
2  collect(N, Round, MaxVoted, Proposal) ->
3    receive
4      {promise, Round, _, na} ->
5        collect(..., ..., ..., ...);
6      {promise, Round, Voted, Value} ->
7        case order:gr(..., ...) of
8          true ->
```

```
9             collect(..., ..., ..., ...);
10          false ->
11            collect(..., ..., ..., ...)
12        end;
13      {promise, _, _,  _} ->
14        collect(N, Round, MaxVoted, Proposal);
15      {sorry, {prepare, Round}} ->
16        collect(..., ..., ..., ...);
17      {sorry, _} ->
18        collect(N, Round, MaxVoted, Proposal)
19    after ?timeout ->
20      abort
21    end.
```

```
1
2 collect(N, Round, MaxVoted, Proposal) ->
3   receive
4     % Promise received, no previous votes. Keep
      collecting 'support'
5     {promise, Round, _, na} ->
6       collect(N-1, Round, MaxVoted, Proposal);
7     {promise, Round, Voted, Value} ->
8       % We got the promise. Update the maximum Voted/
      Proposal
9       case order:gr(Voted, MaxVoted) of
10  % Learn value
11        true ->
12          collect(N-1, Round, Voted, Value);
13  % Keep this proposal
14        false ->
15          collect(N-1, Round, MaxVoted, Proposal)
16      end;
17    % TODO: Old message, ignore and keep going?
18    {promise, _, _,  _} ->
19      collect(N, Round, MaxVoted, Proposal);
20    % Rejected, just keep gathering support
21    {sorry, {prepare, Round}} ->
22      collect(N, Round, MaxVoted, Proposal);
23    % TODO: Old message from message or whatever?
24    {sorry, _} ->
25      collect(N, Round, MaxVoted, Proposal)
26  after ?timeout ->
27    abort
```

Terminar cuando TODO esté hecho

```
1
2 vote(N, Round) ->
3   receive
4     {vote, Round} ->
```

```
5         vote(..., ...);
6       {vote, _} ->
7         vote(N, Round);
8       {sorry, {accept, Round}} ->
9         vote(..., ...);
10      {sorry, _} ->
11        vote(N, Round)
12    after ?timeout ->
13      abort
14    end.
```

```
1
2 vote(N, Round) ->
3   receive
4     {vote, Round} ->
5       vote(N-1, Round); % voto ganado, uno menos
6     {vote, _} -> % voto desactualizado?
7       vote(N, Round);
8     {sorry, {accept, Round}} ->
9       vote(N, Round); % Rejected, keep going
10    {sorry, _} ->
11      vote(N, Round) % Rejected from other round or from
      the promise
12  after ?timeout ->
13    abort
14  end.
```

In the *{vote}* function we expect to recieve a vote or sorry message from the Acceptors. In this function we define the behavior depending what we receive. In case of receive a vote like *{vote, Round}*, we subsctact 1 to the vote necessary for consensus. If we receive *{sorry, {accept, Round}}* that means that the vote was rejected and we keep trying, so we do not substract nothing to the number of votes remaining for achieve consensus.

3 Experiments

Provide evidence of the experiments you did (e.g., use screenshots) and discuss the results you got. In addition, you may provide figures or tables with experimental results of the system evaluation. For each seminar, we will provide you with some guidance on which kind of evaluation you should do.

4 Open questions

Try to answer all the open questions in the documentation. When possible, do experiments to support your answers.

5 Personal opinion

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