SODX LAB3 - Preguntes

Ixent Cornella, Eric Gonzalez

INTRODUCCIÓ

El Laboratori 3 consisteix en implementar un sistema multicast d'ordenació completa utilitzant un algorisme distribuït. L'algoritme que utilitzarem es el que s'utilitza en el sistema operatiu ISIS i es basa en sol·licitar propostes a tots els nodes d'un grup

1. BASIC MULTICAST

i) Set up the basic multicast system, and use the following test program to experiment with different values for `Sleep` and `Jitter`. Does it keep workers synchronized? Justify why.

Quan modifiquem el valor de "Jitter" perquè sigui superior a 0, es produeixen els canvis esperats en la GUI. Això es deu al fet que "Jitter" és el valor del retard de xarxa simulat, la qual cosa provoca que els peers no estiguin sincronitzats i per tant no tinguin colors coincidents. Per altra banda, un valor de 0 com Jitter fa que els treballadors comparteixin el mateix valor de RGB, sempre que el valor de repòs sigui l'adequat (almenys 10 ms perquè puguem notar els canvis de color).

Per altra banda, fent referencia a la pregunta; els workers no es mantenen sincronitzats ja que quan el *basic.erl* s'encarrega de mantenir el sistema de multicast, i sí que hi ha la part en que el sistema s'espera pel Jitter per enviar missatges, però el missatge, però la part més essencial del multicast no està. Aquesta és la part en la que el sistema de multicast contacta tots els nodes per a que es fiquin d'acord, d'aquesta manera evitem que es desincronitzin. Aquest mecanisme no està a *basic.erl*.

```
-module(total).
-export([start/3]).

start(Id, Master, Jitter) ->
    spawn(fun() -> init(Id, Master, Jitter) end).

init(Id, Master, Jitter) ->
    {A1,A2,A3} = now(),
    random:seed(A1, A2, A3),
    receive
        {peers, Nodes} ->
            server(Master, seq:new(Id), seq:new(Id), Nodes, [], [], Jitter)
    end.

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

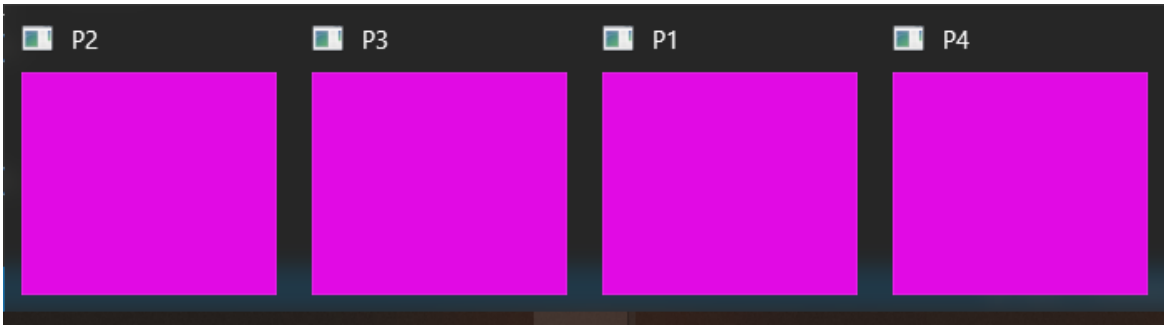
```
        server(Master, MaxPrp, NewMaxAgr, Nodes, Cast, NewQueue, Jitter);
    stop ->
        ok
end.

%% Sending a request message to all nodes
request(Ref, Msg, Nodes, 0) ->
    Self = self(),
    lists:foreach(fun(Node) ->
                      Node ! {request, Self, Ref, Msg}
                  end,
                  Nodes);
request(Ref, Msg, Nodes, Jitter) ->
    Self = self(),
    lists:foreach(fun(Node) ->
                      timer:sleep(random:uniform(Jitter)),
                      Node ! {request, Self, Ref, Msg}
                  end,
                  Nodes).

%% Sending an agreed message to all nodes
agree(Ref, Seq, Nodes)->
    lists:foreach(fun(Pid)->
                      Pid ! {agreed, Ref, Seq}
                  end,
                  Nodes).

%% Delivering messages to the master
deliver(Master, Messages) ->
    lists:foreach(fun(Msg)->
                      Master ! {deliver, Msg}
                  end,
                  Messages).

%% Adding a new entry to the set of casted messages
cast(Ref, Nodes, Cast) ->
    L = length(Nodes),
    [{Ref, L, seq:null()}|Cast].

%% Update the set of casted messages
proposal(Ref, Proposal, [{Ref, 1, Sofar}|Rest])->
    {agreed, seq:max(Proposal, Sofar), Rest};
```

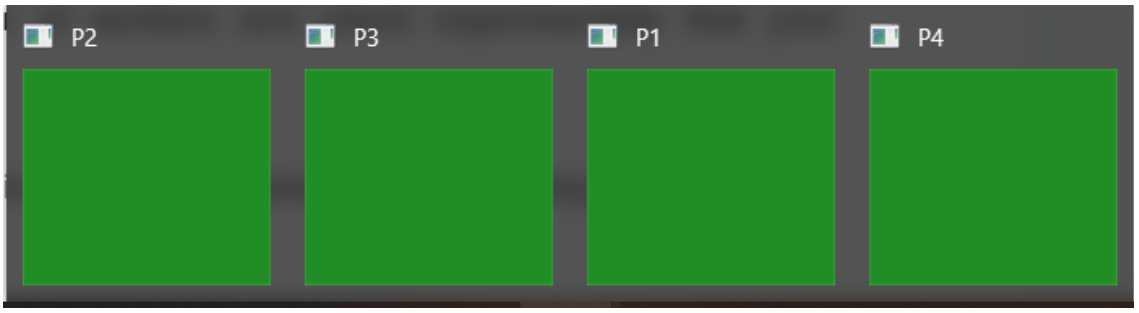
```
proposal(Ref, Proposal, [{Ref, N, Sofar}|Rest])->
    [{Ref, N-1, seq:max(Proposal, Sofar)}|Rest];
proposal(Ref, Proposal, [Entry|Rest])->
    case proposal(Ref, Proposal, Rest) of
        {agreed, Agreed, Rst} ->
            {agreed, Agreed, [Entry|Rst]};
        Updated ->
            [Entry|Updated]
    end.

%% Remove all messages in the front of the queue that have been agreed
agreed([{_Ref, Msg, agrd, _Agr}|Queue]) ->
    {Agreed, Rest} = agreed(Queue),
    {[Msg|Agreed], Rest};
agreed(Queue) ->
    {[], Queue}.

%% Update the queue with an agreed sequence number
update(Ref, Agreed, [{Ref, Msg, propsd, _}|Rest])->
    queue(Ref, Msg, agrd, Agreed, Rest);
update(Ref, Agreed, [Entry|Rest])->
    [Entry|update(Ref, Agreed, Rest)].

%% Insert a new message into the queue
insert(Ref, Msg, Proposal, Queue) ->
    queue(Ref, Msg, propsd, Proposal, Queue).

%% Queue a new entry
queue(Ref, Msg, State, Proposal, []) ->
    [{Ref, Msg, State, Proposal}];
queue(Ref, Msg, State, Proposal, Queue) ->
    [Entry|Rest] = Queue,
    {_, _, _, Next} = Entry,
    case seq:lessthan(Proposal, Next) of
        true ->
            [{Ref, Msg, State, Proposal}|Queue];
        false ->
            [Entry|queue(Ref, Msg, State, Proposal, Rest)]
    end.
```

2. TOTAL ORDER MULTICAST

i) Set up the total order multicast system, and repeat the previous tests. Does it keep workers synchronized.

Ara amb *total.erl* ja codificat, (s'adjunta a l'entrega), podem afirmar que el sistema de multicast permet que els *workers* estiguin sincronitzats.





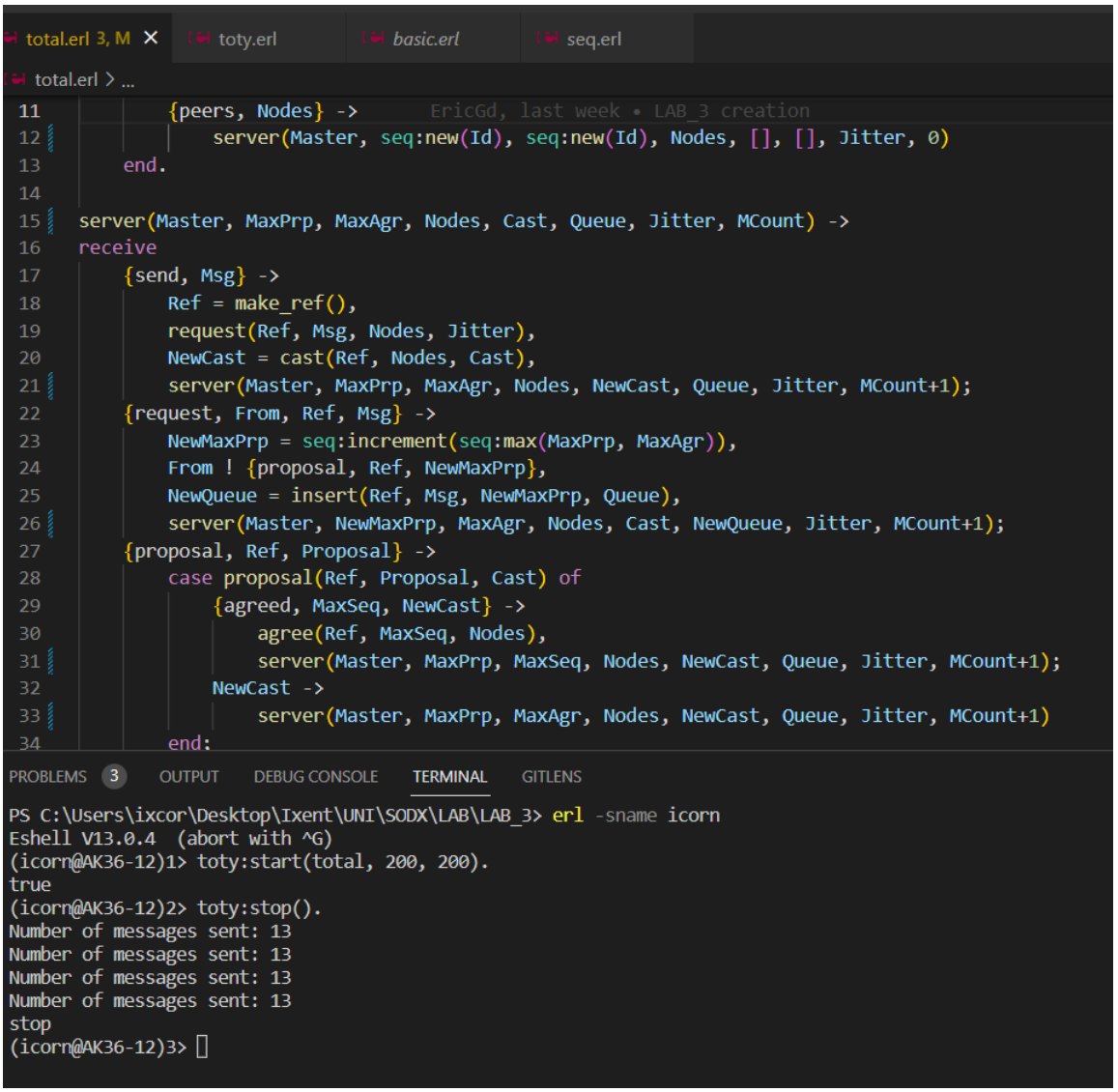
ii) We have a lot of messages in the system. Derive a theoretical quantification of the number of messages needed to deliver a multicast message as a function of the number of workers and check experimentally that your formulation is correct.

En aquest sistema, el nombre de missatges necessaris per lliurar un missatge multicast dependrà del nombre de workers. En general, cada node haurà d'enviar i rebre diversos missatges per tal de lliurar un missatge de multicast, incloent request, proposal i agrees.

Quan un node ha rebut els missatges acordats per a tots els missatges de la seva queue, es lliuren en ordre els missatges al node master.

Pel que fa al nombre de missatges, podem veure que el primer pas de l'algorisme requereix un missatge (l'enviament del missatge al sistema de multicast). El segon pas de l'algorisme requereix 1 missatge per node (els missatges de request a tots els nodes). El tercer pas de l'algorisme requereix també 1 missatge per node (els missatges de proposal de tots els nodes al remitent). Finalment, el quart pas de l'algorisme requereix un missatge per node (els missatges agree del remitent a tots els nodes).

En total, això significa que el nombre de missatges necessaris per lliurar un missatge multicast és de (3*N) + 1, on N és el nombre de workers de toty.



En aquesta captura, s'ha modificat el codi de total per a que s'imprimeixi el nombre de missatges rebuts pel sistema de multicast (amb la variable MCount), i com es pot observar, en un període breu d'execució de toty, el nombre de missatges rebut és de

13. Això és coherent segons la fòrmula donada, ja que s'hauran enviat (3*__4__)+1 missatges, que és igual a 13.

iii) Compare with the basic multicast implementation regarding the number of messages needed.

La implementació bàsica de multicast és diferent del sistema de multicast d'ordre total pel que fa al nombre de missatges necessaris per lliurar un missatge de multicast. A la implementació bàsica, el remitent envia un missatge send al sistema multicast amb el missatge com a paràmetre i, a continuació, el sistema envia un missatge multicast a tots els nodes. Cada node rep el missatge multicast i el lliura al node master.

Pel que fa al nombre de missatges, podem veure que la implementació bàsica requereix 2 missatges per node (1 missatge send al sistema multicast, i 1 missatge multicast del sistema de multidifusió a cada node). En total, això significa que el nombre de missatges necessaris per lliurar un missatge multicast a la implementació bàsica és **N+1**, on N és el nombre de workers de toty.

En comparació, està clar que aquesta implementació requereix menys missatges, ja que no hi ha cap sistema d'ordenació de missatges, per tant només s'envien els missatges "a lo loco", sense garantir sincronització. Això és degut al fet que el sistema multicast d'ordre total utilitza missatges addicionals per garantir que els missatges s'entreguin en l'ordre correcte.