Verteilte Systeme 2012: 9. Übungszettel

Max Michels
Philipp Borgers
Sascha Schönfeld

Schintke, Schütt
05.07.2012

1 Nebenläufige Transaktionen

- Rückwärtsvalidation: T wird vor U validiert.
  - T wird validiert, es gibt vorher keine Transaktion. Die Validierung wird passiert.
  - U wird validiert, nachdem T geschrieben hat. U hat kein Read-Set und passiert die Validierung.
  - **x = 0; i = 55; j = 66;**
- Rückwärtsvalidation: U wird vor T validiert.
  - U wird validiert, es gibt vorher keine Transaktion. Die Validierung wird passiert.
  - T wird validiert, nachdem U geschrieben hat. Das Read-Set umfasst i, das von U geschrieben wurde. Daher wird T abgebrochen.
  - **x = 0; i = 55; j = 66;**
- Vorwärtsvalidation: T wird vor U validiert.
  - T wird gegen U validiert, U hat kein Read-Set. Die Validierung passiert
  - Wenn U validiert wird, ist keine andere Transaktion mehr aktiv, auch hier wird die Validierung passiert.
  - **x = 0; i = 55; j = 66;**
- Vorwärtsvalidation: U wird vor T validiert.
  - U wird validiert, während T noch in der Arbeitsphase ist. Der Konflikt zu read(i) in T wird erkannt.

  Fall 1: T wird abgebrochen. **Ergebnis: x = 0; i = 55; j = 66;**

  Fall 2: Die Validierung von U wird verzögert. T wird beendet und validiert, U wird erneut validiert und passiert die Validierung, da keine Transaktion mehr aktiv ist. **Ergebnis: x = 0; i = 55; j = 66;** (Verhält sich wie "‘T wird vor U validiert"’)

  - **x = 0; i = 55; j = 66;**

Verteilte Systeme 2012: 9. Übungszettel

Max Michels
Philipp Borgers
Sascha Schönfeld

Schintke, Schütt
05.07.2012

2 Chang-Roberts-Algorithmus

2.1 Code

```erlang
-module('chang-roberts').
-export([init/1, loop/1, messageCounter/1, tester/1]).


%% Implementation of the Chang-Roberts ring algorithm
%%
%% For testing please send a message to the tester
%% tester ! test1  <--- starts test1
%% tester ! test2  <--- starts test2

%% Call this function to start with NumProcs processes
init(NumProcs) ->
    io:format("Spawning processes~n"),
    PIDs = init(NumProcs,[]),
    %% we only register once so that we can execute multiple times
    case whereis(messageCounter) of
        undefined ->
            register(messageCounter, spawn('chang-roberts', messageCounter, [0])),
            register(tester, spawn('chang-roberts', tester, [PIDs]));
        _ ->
            ok
    end.

%% Spawning of processes
init(NumProcs, PIDs) when NumProcs > 0 ->
    InitState = {_Participant = false, _Leader = no, _Successor = undefined},
    PID = spawn('chang-roberts', loop, [InitState]),
    io:format("Spawned process ~p~n",[PID]),
    init(NumProcs-1, PIDs ++ [PID]);

%% Spawning of processes done
init(0, PIDs) ->
    establishCircle(PIDs, 0).

%% Used to establish the circle
establishCircle(PIDs, ShiftPos) ->
    case ShiftPos of
        N when N >= length(PIDs) ->
            io:format("established circle.~n");
        _ ->
            Cur = ShiftPos+1,
            Suc = (ShiftPos + 1) rem length(PIDs) + 1,
            lists:nth(Cur, PIDs) ! {setSuccessor, lists:nth(Suc, PIDs)},
            establishCircle(PIDs, ShiftPos+1)
    end,
    PIDs.

%%Tester to control tests
tester(PIDs) ->
    receive
        test1 ->
            messageCounter ! reset,
            test1(PIDs);
        test2 ->
            messageCounter ! reset,
            test2(PIDs);
        true ->
            done
    end,
    tester(PIDs).

%% Testing with random starting node
test1(PIDs) ->
    random:seed(now()),
    {StartKnoten, _X} = random:uniform_s(length(PIDs),random:seed(now())),
    StartPID = lists:nth(StartKnoten, PIDs),
    io:format("Starting election with ~p~n", [StartPID]),
    StartPID ! startElection.

```

```
%% Testing with three random starting nods
test2(PIDs) ->
    test1(PIDs),
    test1(PIDs),
    test1(PIDs).

%% global message counter
messageCounter(Messages) ->
    receive
        count ->
            io:format("Number of messages: ~p~n", [Messages]),
            messageCounter(Messages+1);
        reset ->
            messageCounter(0);
        true ->
            Messages
    end.

%% main loop for the processes
loop({Participant, Leader, Successor}) ->
    %io:format("~p: State ~p~n",[self(), State]),
    NewState =
        receive
            {setSuccessor, PID} ->
                io:format("~p: My successor is now ~p~n",[self(),PID]),
                {Participant, Leader, _Successor = PID};
            startElection ->
                Successor ! {election, self()},
                {_Participant = true, Leader, Successor};
            {election, PID} ->
                if
                    PID > self() ->
                        Successor ! {election, PID},
                        messageCounter ! count,
                        {_Participant = true, Leader, Successor};
                    PID < self() ->
                        if
                            Participant =:= false ->
                                Successor ! {election, self()},
                                messageCounter ! count;
                            true ->
                                ok
                        end,
                        {Participant,Leader,Successor};
                    PID == self() ->
                        Successor ! {elected, self()},
                        messageCounter ! count,
                        {_Participant = false, Leader, Successor}
                end;
            {elected, PID} ->
                if
                    PID =/= self() ->
                        Successor ! {elected, PID},
                        messageCounter ! count,
                        io:format("~p: Leader found (~p)~n", [self(), PID]);
                    true ->
                        ok
                end,
                {_Participant = false, _Leader = PID, Successor}
        end,
    loop(NewState).
```

2.2 Testläufe

```
> c(’chang-roberts’).
> ’chang-roberts’:init(8).
Spawning processes
Spawned process <0.58.0>
Spawned process <0.59.0>
Spawned process <0.60.0>
Spawned process <0.61.0>
Spawned process <0.62.0>
Spawned process <0.63.0>
Spawned process <0.64.0>
Spawned process <0.65.0>
<0.61.0>: My successor is now <0.62.0>
established circle.
<0.58.0>: My successor is now <0.59.0>
<0.59.0>: My successor is now <0.60.0>
<0.60.0>: My successor is now <0.61.0>
<0.62.0>: My successor is now <0.63.0>
<0.63.0>: My successor is now <0.64.0>
<0.64.0>: My successor is now <0.65.0>
<0.65.0>: My successor is now <0.58.0>
ok


Testen Sie den Wahlalgorithmus mit verschiedenen Startknoten
und zahlen Sie die Anzahl der benotigten Nachrichten:

> tester ! test1.
Starting election with <0.43.0>
<0.39.0>: Leader found (<0.43.0>)
<0.40.0>: Leader found (<0.43.0>)
<0.41.0>: Leader found (<0.43.0>)
<0.42.0>: Leader found (<0.43.0>)
Number of messages: 8


> tester ! test1.
Starting election with <0.41.0>
<0.39.0>: Leader found (<0.43.0>)
<0.40.0>: Leader found (<0.43.0>)
<0.41.0>: Leader found (<0.43.0>)
<0.42.0>: Leader found (<0.43.0>)
Number of messages: 10


> tester ! test1.
Starting election with <0.42.0>
<0.39.0>: Leader found (<0.43.0>)
<0.40.0>: Leader found (<0.43.0>)
<0.41.0>: Leader found (<0.43.0>)
<0.42.0>: Leader found (<0.43.0>)
Number of messages: 9


Testen Sie den Wahlalgorithmus mit mehreren nebenlaufigen Starts in verschiedenen Prozessen
und zahlen Sie die Anzahl der benotigten Nachrichten.

> tester ! test2.
```

```
Starting election with <0.40.0>
Starting election with <0.40.0>
Starting election with <0.40.0>
<0.40.0>: Leader found (<0.43.0>)
<0.41.0>: Leader found (<0.43.0>)
<0.42.0>: Leader found (<0.43.0>)
<0.39.0>: Leader found (<0.43.0>)
<0.40.0>: Leader found (<0.43.0>)
<0.41.0>: Leader found (<0.43.0>)
<0.42.0>: Leader found (<0.43.0>)
<0.39.0>: Leader found (<0.43.0>)
<0.40.0>: Leader found (<0.43.0>)
<0.41.0>: Leader found (<0.43.0>)
<0.42.0>: Leader found (<0.43.0>)
Number of messages: 35
```