Verteilte Systeme 2012: 6. Übungszettel

Max Michels
Schintke, Schütt — Philipp Borgers
07.06.2012 — Sascha Schönfeld

1 Konsistenzmodell für einen Aktienhandel

Für einen Aktienhandel sollte kausale Konsistenz verwendet werden. Die wichtigste Voraussetzung für einen Aktienhandel ist, dass die Änderungen der Werte (Aktienpreise) stets konsistent sind, die auch kausal voneinander abhängen. Wertänderungen, die voneinander unabhängig sind, sind nicht relevant für die einzelne Aktie.

2 Konsistenzmodelle

2.a

Zeigen sie, dass der folgende Verlauf nicht kausal konsistent ist:

P_1: W(a)0 W(a)1
P_2: R(a)1 W(b)2
P_3: R(b)2 R(a)0

Der Verlauf ist nicht kausal konsistent, da P_2 a = 1 liest, bevor P_3 a = 0 liest. Die Bedingung für kausale Konsistenz sagt aus, dass Schreiboperationen, die in Kausalität stehen (Was für die beiden Operationen in P_1 zutrifft, da sie die selbe Variable beschreiben), von allen Prozessen in genau der Reihenfolge gesehen werden müssen, in der sie ausgeführt wurden.

2.b

Ist der Speicher, der der folgenden Ausführung zugrundeliegt, sequentiell konsistent (vorausgesetzt, alle Variablen sind zunächst auf Null gesetzt)?

P_1: R(x)1 R(x)2 W(y)1
P_2: W(x)1 R(y)1 W(x)2

Der Speicher ist nicht sequentiell konsistent. Bei den jeweils zweiten Anweisungen gibt es bereits Probleme:
W(x)1 → R(x)1 → R(x)2 ist nicht korrekt, da x = 1.
W(x)1 → R(x)1 → R(y)1 ist ebenfalls falsch, da y noch nicht initialisiert wurde.

2.c

Welchen Konsistenzmodellen entspricht b) ggf. zusätzlich?
Die Reihenfolge entspricht lediglich der FIFO-Konsistenz.

Verteilte Systeme 2012: 6. Übungszettel

Max Michels
Schintke, Schütt — Philipp Borgers
07.06.2012 — Sascha Schönfeld

3

3.a Code

Wir haben lediglich einen Multicast in Erlang implementiert. Für die Umsetzung des Best-Effort-Mulicasts mit kausaler Ordnung fehlte uns wegen wenig Praxis in Erlang die nötige Vorstellungskraft, um das Verstandene in Code zu gießen. Zunächst werden fünf Prozesse gestartet, die fortwährend die loop-Funktion ausführen. Die Loop-Funktion wartet auf Nachrichten verschiedene Art: eine join-Nachricht, die das Beitreten einer Gruppe ermöglicht, eine mutlicast-Nachricht zum Senden von Nachrichten an alle Gruppenmitglieder und eine Nachricht zum Ausgeben der empfangegenen Nachrichten.

```
-module(multicast).

-compile(export_all).


test() ->
    P1 = start(),
    P2 = start(),
    P3 = start(),
    P4 = start(),
    P5 = start(),
    Group = [P1, P2, P3, P4, P5],
    P1 ! {join, Group},
    P1 ! {multicast, "This is a message"},
    Group.

start() ->
    Pid = spawn(multicast, loop, [{0, 0, []}]),
    Pid.

loop({S, R, Group}=State) ->
    receive
        {join, Members} ->
            io:format("Process ~p joined group ~p~n", [self(), Members]),
            loop({S, R, Members});
        %% start a multicast message
        {multicast, Message} ->
            io:format("Received multicast init message ~p~n", [Message]),
            io:format("Sending multicast message to group ~p~n", [Group]),
            % send message to each member of the group
            Fun = fun (Pid) ->
                    Pid ! {msg, self(), S, Message}
            end,
            lists:foreach(Fun, Group),
            %% increase seq. number
            loop({S+1, R, Group});
        {msg, Src, S2, Message} ->
            io:format("Process ~p received Message ~p~n", [self(), {Src, S2, Message}]),
            loop(State);
        X ->
            io:format("Fail...~p~n",[X]),
            loop(State)
    end.
```

Verteilte Systeme 2012: 6. Übungszettel

Max Michels
Schintke, Schütt — Philipp Borgers
07.06.2012 — Sascha Schönfeld

3.b Testlauf

```
Eshell V5.8  (abort with ^G)
1> c(multicast).
{ok,multicast}
2> multicast:test().
Process <0.39.0> joined group [<0.39.0>,<0.40.0>,<0.41.0>,<0.42.0>,<0.43.0>]
[<0.39.0>,<0.40.0>,<0.41.0>,<0.42.0>,<0.43.0>]
Received multicast init message "This is a message"
Sending multicast message to group [<0.39.0>,<0.40.0>,<0.41.0>,<0.42.0>,
                                    <0.43.0>]
Process <0.39.0> received Message {<0.39.0>,0,"This is a message"}
Process <0.40.0> received Message {<0.39.0>,0,"This is a message"}
Process <0.41.0> received Message {<0.39.0>,0,"This is a message"}
Process <0.42.0> received Message {<0.39.0>,0,"This is a message"}
Process <0.43.0> received Message {<0.39.0>,0,"This is a message"}
```