Verteilte Systeme: Portfolioprüfung Lamporttime

Jost-Tomke Müller, Nathalie Möck, Babett Müller, Mathis Neunzig
July 29, 2022

Agenda

Aufgabe A
- Hausarbeit über Erlang und Verteilte Entwicklung
- Gliederung der Hausarbeit

Aufgabe B
- Code
- Auswertung

Aufgabe C
- Code
- Auswertung

Aufgabe D
- Idee
- Code
- Auswertung

Aufteilung

Aufgabe A

Aufgabe A:

Installieren Sie den Erlang-Compiler und ggf. eine geeignete Entwicklungsumgebung. Arbeiten Sie sich in die Grundlagen der Programmiersprache ein und machen Sie sich mit Aktoren und der verteilten Programmierung vertraut. Verwenden Sie hierfur insbesondere die auf erlang.org bereitgestellten Unterlagen. Schreiben Sie ein vier- bis funfseitiges Summary hierzu. Vergessen Sie nicht, verwendete Quellen anzugeben.

Hausarbeit

- Siehe .zip-Verzeichnis
- Grundlagen Erlang
- Grundlagen Verteilte Entwicklung

Gliederung

- Einleitung
- Aktoren & Verteilte Programmierung
- Erlang
  - Grundlagen
  - Die Rolle in Verteilter Programmierung
  - Vorteile
  - Nachteile

Aufgabe B

Aufgabe B:

Bringen Sie das Programm (s. Anhang) zum Laufen. Welche Ausgabe wird erzeugt? Beschreiben Sie detailliert, wie das Programm funktioniert.

Grundidee des Programms

- 4 Worker
- 1 Logger
- Worker senden und empfangen Nachrichten an bzw. von einem anderen
- Logger loggt die jeweiligen Nachichten

Aufgabe B

```
-module(mylogger).
-export([start/1, stop/1]).

start(Nodes) ->
  spawn_link(fun() -> init(Nodes) end).

stop(Logger) ->
  Logger ! stop.

init(_) ->
  loop().

loop() ->
  receive {log, From, Time, Msg} ->
      log(From, Time, Msg),
      loop();
    stop ->
      ok
end.

log(From, Time, Msg) ->
  io:format("log: ~w ~w ~p~n", [Time, From, Msg]).
```

Aufgabe B

```erlang
-module(worker).
-export([start/4, stop/1, peers/2]).

start(Name, Logger, Sleep, Jitter) ->
  spawn_link(fun() -> init(Name, Logger, Sleep, Jitter) end).

stop(Worker) ->
  Worker ! stop.

init(Name, Log, Sleep, Jitter) ->
  receive
    {peers, Peers} ->
      loop(Name, Log, Peers, Sleep, Jitter);
        stop ->
          ok
end.

peers(Wrk, Peers) ->
  Wrk ! {peers, Peers}.

loop(Name, Log, Peers, Sleep, Jitter) ->
  Wait = rand:uniform(Sleep),
  receive
    {msg, Time, Msg} ->
      Log ! {log, Name, Time, {received, Msg}},
      loop(Name, Log, Peers, Sleep, Jitter);
        stop ->
          ok;
    Error ->
      Log ! {log, Name, time, {error, Error}}
    after Wait ->
      Selected = select(Peers),
      Time = erlang:system_time(millisecond),
      Message = {hello, rand:uniform(100)},
      Selected ! {msg, Time, Message},
      jitter(Jitter),
        Log ! {log, Name, Time, {sending, Message}},
        loop(Name, Log, Peers, Sleep, Jitter)
end.

select(Peers) ->
  lists:nth(rand:uniform(length(Peers)), Peers).

jitter(0) -> ok;
jitter(Jitter) -> timer:sleep(rand:uniform(Jitter)).
```

Aufgabe B

```
1  -module(test).
2  -export([run/2]).
3
4  run(Sleep, Jitter) ->
5    Log = mylogger:start([einstein, euler, curie, turing]),
6    A = worker:start(einstein, Log, Sleep, Jitter),
7    B = worker:start(euler, Log, Sleep, Jitter),
8    C = worker:start(curie, Log, Sleep, Jitter),
9    D = worker:start(turing, Log, Sleep, Jitter),
10   worker:peers(A, [B, C, D]),
11   worker:peers(B, [A, C, D]),
12   worker:peers(C, [A, B, D]),
13   worker:peers(D, [A, B, C]),
14   timer:sleep(5000),
15   mylogger:stop(Log),
16   worker:stop(A),
17   worker:stop(B),
18   worker:stop(C),
19   worker:stop(D).
```

Aufgabe B

Befehl: `test:run(1000,0).`

```
log: 1657231719756 curie {sending,{hello,27}}
log: 1657231719756 euler {received,{hello,27}}
log: 1657231720004 einstein {sending,{hello,28}}
log: 1657231720004 euler {received,{hello,28}}
log: 1657231720220 turing {sending,{hello,100}}
log: 1657231720220 euler {received,{hello,100}}
log: 1657231720236 turing {sending,{hello,67}}
log: 1657231720236 curie {received,{hello,67}}
log: 1657231720627 curie {sending,{hello,9}}
log: 1657231720627 turing {received,{hello,9}}
log: 1657231720689 curie {sending,{hello,42}}
log: 1657231720689 einstein {received,{hello,42}}
log: 1657231720843 euler {sending,{hello,63}}
log: 1657231720843 turing {received,{hello,63}}
log: 1657231721261 euler {sending,{hello,26}}
log: 1657231721261 einstein {received,{hello,26}}
log: 1657231721463 curie {sending,{hello,11}}
log: 1657231721463 einstein {received,{hello,11}}
log: 1657231721511 turing {sending,{hello,73}}
log: 1657231721511 einstein {received,{hello,73}}
log: 1657231721695 turing {sending,{hello,98}}
log: 1657231721695 einstein {received,{hello,98}}
log: 1657231721803 euler {sending,{hello,20}}
log: 1657231721803 turing {received,{hello,20}}
log: 1657231721818 curie {sending,{hello,38}}
```

Aufgabe B

Befehl: `test:run(1000,1000).`

```
log: 1657231764878 einstein {received,{hello,35}}
log: 1657231764970 euler {received,{hello,95}}
log: 1657231764878 curie {sending,{hello,35}}
log: 1657231765220 curie {received,{hello,78}}
log: 1657231764970 turing {sending,{hello,95}}
log: 1657231765609 einstein {sending,{hello,89}}
log: 1657231765885 turing {received,{hello,68}}
log: 1657231765885 einstein {sending,{hello,70}}
log: 1657231765885 curie {sending,{hello,68}}
log: 1657231765885 curie {received,{hello,70}}
log: 1657231765220 euler {sending,{hello,78}}
log: 1657231765609 euler {received,{hello,89}}
log: 1657231766259 curie {received,{hello,69}}
log: 1657231766321 curie {received,{hello,56}}
log: 1657231766259 euler {sending,{hello,69}}
log: 1657231766474 turing {received,{hello,27}}
log: 1657231766474 curie {sending,{hello,27}}
log: 1657231766788 curie {received,{hello,53}}
log: 1657231766321 einstein {sending,{hello,56}}
log: 1657231766818 einstein {received,{hello,90}}
log: 1657231766818 curie {sending,{hello,90}}
log: 1657231767129 curie {received,{hello,84}}
log: 1657231767300 curie {received,{hello,71}}
log: 1657231766788 euler {sending,{hello,53}}
log: 1657231767346 euler {received,{hello,91}}
```

Aufgabe C

Aufgabe C:

Erweitern Sie das Programm um Lamport-Uhren. Implementieren Sie für die Lamport'schen Zeitstempel ein Modul lamporttime. Senden Sie die Timestamps an den Logger mit, der sie mit ausgibt. Experimentieren Sie mit unterschiedlichen Werten fur den Jitter. Welche Probleme mit der Reihenfolge der Ausgabe zeigen sich? Fügen Sie Ihrem Programmcode eine Beschreibung bei.

Grundidee des Programms

- 4 Worker
- 1 Logger
- Worker senden und empfangen Nachrichten an bzw. von einem anderen
- Logger loggt die jeweiligen Nachichten, jedoch mit einem Lamport-Zeitstempel

Aufgabe C

```erlang
-module(lamporttime).
-export([zero/0, inc/1, merge/2,leq/2]).

% Eine einfache Funktion, welche nur den Wert '0' zurückgibt
zero() ->
    0.

% Diese Funktion nimmt eine Zahl entgegen und gibt diese inkrementiert zurück
inc(T) ->
    T + 1.

% Diese Funktion vergleicht zwei übergebende Werte und gibt den größeren zurück
merge(Ti, Tj) ->
   if
        % So Ti größer als Tj ist, gebe Ti zurück.
        Ti > Tj -> Ti;
        % So Ti kleiner als Tj ist, gebe Tj zurück.
        Ti < Tj -> Tj;
        % Wenn beide Werte gleich groß sind, so gebe Ti zurück.
        true -> Ti
   end.

% Diese Funktion gibt zurück, ob Ti kleiner oder gleich Tj ist.
leq(Ti, Tj) -> (Ti =< Tj).
```

Aufgabe C

```erlang
-module(worker).
-export([start/4, stop/1, peers/2]).

start(Name, Logger, Sleep, Jitter) ->
  spawn_link(fun() -> init(Name, Logger, Sleep, Jitter) end).

stop(Worker) ->
  Worker ! stop.

init(Name, Log, Sleep, Jitter) ->
  receive
    {peers, Peers} ->
      loop(Name, Log, Peers, Sleep, Jitter, lamporttime:zero());
    stop ->
      ok
end.

peers(Wrk, Peers) ->
  Wrk ! {peers, Peers}.

```

```erlang
loop(Name, Log, Peers, Sleep, Jitter, LocalTime) ->
  Wait = rand:uniform(Sleep),
  receive
    {msg, Time, Msg} ->
      NewLocalTime = lamporttime:inc(lamporttime:merge(Time, LocalTime)),
      Log ! {log, Name, NewLocalTime, {received, Msg}},
      loop(Name, Log, Peers, Sleep, Jitter, NewLocalTime);
    stop ->
          ok;
    Error ->
      Log ! {log, Name, time, {error, Error}}
    after Wait ->
      Selected = select(Peers),
      NewLocalTime = lamporttime:inc(LocalTime),
      Message = {hello, rand:uniform(100)},
      Selected ! {msg, NewLocalTime, Message},
      jitter(Jitter),
        Log ! {log, Name, NewLocalTime, {sending, Message}},
        loop(Name, Log, Peers, Sleep, Jitter, NewLocalTime)
end.

select(Peers) ->
  lists:nth(rand:uniform(length(Peers)), Peers).

jitter(0) -> ok;
jitter(Jitter) -> timer:sleep(rand:uniform(Jitter)).
```

Aufgabe C

Befehl: `test:run(1000,0).`

```
log: 1 curie {sending,{hello,83}}
log: 2 einstein {received,{hello,83}}
log: 1 turing {sending,{hello,7}}
log: 2 curie {received,{hello,7}}
log: 1 euler {sending,{hello,43}}
log: 2 turing {received,{hello,43}}
log: 3 curie {sending,{hello,25}}
log: 4 turing {received,{hello,25}}
log: 3 einstein {sending,{hello,68}}
log: 4 euler {received,{hello,68}}
log: 4 curie {sending,{hello,79}}
log: 5 einstein {received,{hello,79}}
log: 5 euler {sending,{hello,59}}
log: 6 turing {received,{hello,59}}
log: 6 einstein {sending,{hello,77}}
log: 7 turing {received,{hello,77}}
log: 8 turing {sending,{hello,12}}
log: 9 einstein {received,{hello,12}}
log: 6 euler {sending,{hello,91}}
log: 7 curie {received,{hello,91}}
log: 9 turing {sending,{hello,68}}
log: 10 einstein {received,{hello,68}}
log: 7 euler {sending,{hello,16}}
log: 11 einstein {received,{hello,16}}
log: 10 turing {sending,{hello,6}}
log: 11 curie {received,{hello,6}}
```

Aufgabe C

Befehl: `test:run(1000,1000).`

```
log: 2 euler {received,{hello,32}}
log: 3 euler {sending,{hello,22}}
log: 4 euler {received,{hello,84}}
log: 5 euler {received,{hello,65}}
log: 1 turing {sending,{hello,65}}
log: 1 curie {sending,{hello,84}}
log: 1 einstein {sending,{hello,32}}
log: 4 einstein {received,{hello,22}}
log: 3 curie {received,{hello,19}}
log: 5 einstein {sending,{hello,37}}
log: 2 turing {sending,{hello,19}}
log: 7 turing {received,{hello,4}}
log: 4 curie {sending,{hello,100}}
log: 6 euler {sending,{hello,4}}
log: 7 euler {received,{hello,100}}
log: 8 euler {received,{hello,37}}
log: 9 euler {received,{hello,59}}
log: 8 turing {received,{hello,97}}
log: 6 einstein {sending,{hello,59}}
log: 5 curie {sending,{hello,97}}
log: 11 curie {received,{hello,19}}
log: 10 euler {sending,{hello,19}}
log: 11 euler {received,{hello,70}}
log: 13 einstein {received,{hello,89}}
```

Aufgabe D

Aufgabe D:

Ändern Sie nun den Logger und das Modul lamporttime so ab, dass die Reihenfolgeprobleme (unabhängig vom Jitter) nicht mehr auftreten. Erweitern Sie insbesondere den Logger hierzu um eine Message-Queue, welche die eingehenden Messages vor dem Ausgeben genau solange zuruckbehält, bis es sicher ist, dass die korrekte Reihenfolge eingehalten warden kann. Überlegen Sie zunächst, wie sich das erreichen lässt. Beschreiben Sie Ihre Idee und bewerten Sie Ihre Lösung!

Grundidee des Programms

- 4 Worker
- 1 Logger
- Worker senden und empfangen Nachrichten an bzw. von einem anderen
- Logger loggt die jeweiligen Nachichten, jedoch mit einem Lamport-Zeitstempel
- Die Logs werden sortiert und erst ausgegeben, wenn die Reihenfolge stimmt

Aufgabe D

```
23  % initialisiert die Liste der Uhren mit ihren Zeitstempeln
24  initClocks(Nodes) -> lists:foldl(fun(Node, Clocks) -> [{Node, zero()} | Clocks] end, [], Nodes).
25
26  % Einen bestimmten Zeitstempel in Liste aktualisieren
27  updateClocks(Node, Time, Clocks) -> lists:keyreplace(Node, 1, Clocks, {Node, Time}).
28
29  % Überpfüft, ob eine Message geloggt werden kann, indem der kleinste Zeitstempel zurückgegeben wird.
30  canLog(Clocks) -> element(2, hd(lists:keysort(2, Clocks))).
```

Aufgabe D

```erlang
-module(mylogger).
-export([start/1, stop/1]).

% Startet einen neuen Worker
start(Nodes) ->
  spawn_link(fun() -> init(Nodes) end).

% Stoppt den Logger
stop(Logger) ->
  Logger ! stop.

% Initialisiert die Liste der Uhren und start den Loop
init(Nodes) ->
  loop(lamporttime:initClocks(Nodes), []).

```

Aufgabe D

```
% Loop, der durchgehent Nachrichten received und verwaltet
loop(Clocks, Queue) ->

  receive

    % Eine Log-Nachricht kommt an
    {log, From, Time, Msg} ->

        % Die zum Log gehörende Uhr wird in der Liste der Uhren geupdatet, der neue Zeitstempel wird hinterlegt
        UpdatedClocks = lamporttime:updateClocks(From, Time, Clocks),
        % Der kleinste Timestamp wird gesucht, damit geprüft werden kann, ob der momentane Log geloggt werden kann
        CheckedTimestamp = lamporttime:canLog(UpdatedClocks),

        case Time of
            1 ->
              % Wenn der Timestamp 1 ist, kann die Nachricht so geloggt werden
                SortedQueue = lists:keysort(2, Queue),
                log(From, Time, Msg);
            _ ->
              % Sonst wird der Log sortiert der Queue hinzugefügt
                SortedQueue = lists:keysort(2, Queue ++ [{From, Time, Msg}])
        end,

        % Die Liste aller Logs momentan wird geteilt:
        % Die ReadyQueue beinhaltet alle Logs, die jetzt geloggt werden können
        % Die UpdatedQueue beinhaltet alle Logs, die noch nicht geloggt werden können und erst in der nächsten Loop-Iteration erneut geprüft werden müssen
        {ReadyQueue, UpdatedQueue} = partition(SortedQueue, CheckedTimestamp),

        % Alle Logs aus der ReadyQueue werden ausgegeben
        printAll(ReadyQueue),

        % Die Liste mit den geupdateten Uhren und die Queue mit den nicht geloggten Logs wird in die nächste Loop-Iteration gegeben
        loop(UpdatedClocks, UpdatedQueue);

    stop ->
        ok

  end.
```

Aufgabe D

```
55  partition(SortedQueue, CheckedTimestamp) ->
56    % Die SortedQueue wird aufgeteilt:
57    % Alle Logs mit einem Timestamp, der kleiner als der CheckedTimestamp ist, werden in die erste Liste gespeichert
58    % Alle Logs mit einem zu großen Timestamp, werden in der zweiten Liste gespeichert
59    lists:splitwith(fun({_, Timestamp, _}) ->
60      lamporttime:leq(Timestamp, CheckedTimestamp)
61      end,
62    SortedQueue).
63
64  printAll(ReadyQueue) ->
65
66    % Alle logs werden hintereinander ausgegeben
67    lists:foreach(
68      fun({Sender, Timestamp, Message}) ->
69        log(Sender, Timestamp, Message)
70      end,
71    ReadyQueue).
72
73  % Log wird auf dem Standardoutput ausgegeben
74  log(From, Time, Msg) ->
75    io:format("log: ~w ~w ~p~n", [Time, From, Msg]).
```

Aufgabe D

Befehl: `test:run(1000,0).`

```
log: 1 einstein {sending,{hello,21}}
log: 1 euler {sending,{hello,93}}
log: 2 curie {received,{hello,21}}
log: 2 einstein {sending,{hello,51}}
log: 2 euler {sending,{hello,61}}
log: 3 curie {received,{hello,93}}
log: 3 turing {received,{hello,51}}
log: 4 turing {sending,{hello,12}}
log: 4 curie {sending,{hello,52}}
log: 5 einstein {received,{hello,12}}
log: 5 turing {received,{hello,52}}
log: 6 einstein {sending,{hello,2}}
log: 6 turing {sending,{hello,6}}
log: 7 curie {received,{hello,2}}
log: 7 einstein {sending,{hello,42}}
log: 7 turing {sending,{hello,22}}
log: 8 curie {received,{hello,42}}
log: 8 einstein {received,{hello,6}}
log: 8 euler {received,{hello,22}}
log: 8 turing {sending,{hello,32}}
log: 9 curie {received,{hello,61}}
log: 9 euler {sending,{hello,90}}
log: 9 turing {sending,{hello,58}}
log: 10 curie {sending,{hello,85}}
```

Aufgabe D

Befehl: `test:run(1000,1000).`

```
log: 1 einstein {sending,{hello,66}}
log: 1 euler {sending,{hello,20}}
log: 1 curie {sending,{hello,58}}
log: 2 turing {received,{hello,58}}
log: 2 einstein {received,{hello,20}}
log: 3 turing {received,{hello,66}}
log: 3 einstein {sending,{hello,29}}
log: 4 euler {received,{hello,29}}
log: 4 turing {sending,{hello,1}}
log: 4 einstein {sending,{hello,3}}
log: 5 curie {received,{hello,1}}
log: 5 euler {received,{hello,3}}
log: 6 curie {sending,{hello,47}}
log: 7 turing {received,{hello,47}}
log: 8 turing {sending,{hello,6}}
log: 9 euler {received,{hello,6}}
log: 10 euler {sending,{hello,32}}
log: 11 einstein {received,{hello,32}}
log: 11 euler {sending,{hello,20}}
log: 12 einstein {sending,{hello,47}}
log: 12 euler {sending,{hello,95}}
stop
```

Aufgabe D

Befehl: `test:run(1,1).`

(Die Laufzeit des Programmes wurde auf 500 ms verringert, um Überbelastung der Ausgabe zu vermeiden)

```
log: 1 einstein {sending,{hello,45}}
log: 1 euler {sending,{hello,44}}
log: 1 curie {sending,{hello,76}}
log: 1 turing {sending,{hello,49}}
log: 2 einstein {received,{hello,44}}
log: 2 euler {received,{hello,45}}
log: 2 curie {received,{hello,49}}
log: 2 turing {sending,{hello,79}}
log: 3 einstein {received,{hello,76}}
log: 3 euler {sending,{hello,82}}
log: 3 curie {sending,{hello,21}}
log: 4 einstein {sending,{hello,25}}
log: 4 euler {received,{hello,21}}
log: 4 curie {sending,{hello,95}}
log: 5 einstein {received,{hello,82}}
log: 5 turing {received,{hello,25}}
log: 5 euler {sending,{hello,50}}
log: 5 curie {sending,{hello,23}}
log: 6 einstein {received,{hello,79}}
log: 6 turing {sending,{hello,21}}
log: 6 euler {sending,{hello,16}}
log: 7 einstein {sending,{hello,74}}
log: 7 curie {received,{hello,16}}
log: 8 einstein {received,{hello,21}}
log: 8 turing {received,{hello,74}}
log: 8 curie {sending,{hello,29}}
```

Fazit

- Aufgabe A
  - Hausarbeit
- Aufgabe B
  - Funktionierender Worker & Logger
  - Systemzeitstempel
  - Jitter -> bringt logische Reihenfolge durcheinander
- Aufgabe C
  - Lamporttimestamp
  - Jitter -> bringt logische Reihenfolge durcheinander
- Aufgabe D
  - MessageQueue -> Aufteilung in 2 Queues
  - Jitter -> bringt logische Reihenfolge nicht mehr durcheinander
  - Richtige Reigenfolge

Aufteilung

- Aufgabe A
  - Ausarbeitung: Nathalie Möck & Babett Müller
  - Vertonung: Mathis Neunzig
- Aufgabe B
  - Ausarbeitung: Alle ☺
  - Vertonung: Jost-Tomke Müller
- Aufgabe C
  - Ausarbeitung & Code: Jost-Tomke Müller
  - Vertonung: Babett Müller
- Aufgabe D
  - Ausarbeitung & Code: Mathis Neunzig
  - Vertonung: Nathalie Möck
- Video
  - Aufnahme: Alle ☺
  - Schnitt: Jost-Tomke Müller

Thank you.

Contact information:

Jost-Tomke Müller, Nathalie Möck, Babett Müller, Mathis Neunzig

jost-tomke.mueller@sap.com, nathalie.moeck@sap.com,
babett.mueller@sap.com, mathis.neunzig@sap.com