11. NOVEMBER 2014

ENTWURF

VERTEILTE SYSTEME PRAKTIKUM 2

STEFFEN GIERSCH & MARIA LÜDEMANN
BAI 5 VERTEILTE SYSTEME BEI HERRN PROFESSOR KLAUCK
HAW Hamburg

Team: 2 Steffen Giersch & Maria Lüdemann

Aufgabenaufteilung:

Aufgabe	Teammitglied
Entwurfsplanung	Steffen Giersch
Entwurfsplanung	Maria Lüdemann
Implementation	Steffen Giersch
Implementation	Maria Lüdemann

Bearbeitungszeitraum:

Datum	Dauer	Teammitglied
01.11.2014	3 Stunden	Steffen & Maria
02.11.2014	5 Stunden	Steffen
03.11.2014	4 Stunden	Steffen
04.11.2014	2 Stunden	Steffen & Maria
06.11.2014	2,5 Stunden	Steffen & Maria
06.11.2014	3,5 Stunden	Maria
08.11.2014	4 Stunden	Steffen & Maria
09.11.2014	8 Stunden	Steffen & Maria

Gesamt: 32 Stunden

Quellen: Koordinator Prozessdiagramme erarbeitet mit einem Vorbild von Fabian Sawatzki und Birger Kamp

Aktueller Stand:

- Entwurfsdokument fertig gestellt
- Implementation fertig

1 INHALTSVERZEICHNIS

2 FUNKTIONALITÄT

Es soll eine Anwendung geschrieben werden die mittels verteilter GGT-Prozesse mit der der größte gemeinsame Teiler einer beliebigen Zahl ausgerechnet werden kann. Dabei soll ein Koordinator die leitende Rolle übernehmen und die verteilten GGT-Prozesse in einem Ring anordnen in dem sie jeweils nur ihre Nachbarn kennen und sich mit ihnen austauschen können.

2.1 KOMPONENTEN

Die Kommunikation läuft hierbei über einen Namensdienst. Die Anzahl der GGT-Prozesse soll beliebig sein und durch einen Starter-Prozess gestartet werden. Dabei besteht das System aus fünf Kernkomponenten.

1. **Dem Koordinator**, der die Schnittstelle zwischen Zookeeper und System darstellt, das System hoch und herunterfährt und die Berechnung überwacht und loggt.
2. **Den Starter-Prozessen** die jeweils einen Teil der GGT-Prozesse starten.
3. **Den GGT-Prozessen,** die die eigentliche Berechnung bewerkstelligen.
4. **Dem Namensdienst**, der den PIDs Namen zuordnet.
5. **Dem ZooKeeper**, dient der Nachrichtenübermittlung und vermittelt zwischen User und Koordinator.

2.2 PHASEN

Das System ist dabei in Phasen unterteil die sich kurz in folgende Phasen teilen:

1. **Initialisierungsphase**
2. **Arbeitsphase**
3. **Beendigungsphase**

2.2.1 Initialisierungsphase

Als erstes werden der Namensdienst und der Koordinator gestartet. Danach werden die Starter Prozesse gestartet und erfragen beim Koordinator die Werte für die GGT-Prozesse um damit die gewünschte Anzahl an Prozessen zu starten und sich dann selbständig zu beenden.

Mit der Nachricht „rebind“ beginnen nun die GGT-Prozesse sich beim Namensdienst eintragen zu lassen und melden sich dann mit der Nachricht „hello“ beim Koordinator.

Die Aufgabe des Koordinators ist in der Initialisierungsphase die Starter und GGT-Prozesse anzunehmen und zu warten bis er den Befehl bekommt den Ring aufzubauen. Der wird aufgebaut indem die Prozesse in einer zufälligen Reihenfolge in einem Ring angeordnet und sie mit der Nachricht „setneighbors“ über ihre jeweiligen Nachbarn informiert werden. Dadurch beendet der Koordinator die Initialisierungsphase.

2.2.2 Arbeitsphase

Am Anfang der Arbeitsphase informiert der Koordinator jeden GGT-Prozess über seinen Startwert „MI“ mittels der Nachricht „setpm“. Nachdem alle GGT-Prozesse ihren Startwert erhalten haben schickt der Koordinator an 15% der GGT-Prozesse (aber min. 2) eine Nachricht „sendy“ mit einem Startwert um die Berechnung zu starten und die 15% „anzustubsen“.

Immer wenn sich der „MI“ eines GGT-Prozesses ändert, informiert er den Koordinator mit der Nachricht „briefmi“ darüber und schickt eine Nachricht „sendy“ mit dem neuen Wert an seinen linken und rechten Nachbarn.

Wenn nun einer der GGT-Prozesse für "Wartezeit" Millisekunden keine Nachricht bekommt, startet er eine Terminierungsabstimmung. Dazu schickt er eine Nachricht "abstimmung" mit seinem eigenen registrierten Namen an seinen rechten Nachbarn. Wenn sein Nachbar "Wartezeit" / 2 Millisekunden gewartet hat schickt er diese Nachricht weiter. Kommt diese Nachricht beim ursprünglichen Versender wieder an, gilt die Berechnung als beendet und das Endergebnis der Berechnung wird mit der Nachricht "briefterm" an den Koordinator geschickt.

2.2.3 Beendigungsphase

In dieser Phase beendet der Koordinator das System und fährt es herunter in dem er jedem GGT-Prozess eine Nachricht „kill“ schickt. Die jeweiligen Prozesse nehmen dies zum Anlass sich beim Namensdienst mit der Nachricht „unbind“ abzumelden und dann zu beenden. Am Ende terminiert der Koordinator.

3 KOORDINATOR

3.1 ALLGEMEINES

Der Koordinator hat die Rolle das System hoch und runter zu fahren und die Berechnung zu verwalten. Bei ihm melden sich alle GGT und Starterprozesse und der Benutzer hat die

Möglichkeit über ihn mittels des Zookeepers auf das System durch gewisse Nachrichten Einfluss zu nehmen.

Die nötigen Konfig-Parameter mit der der Koordinator gestartet wird sind in einer koordinator.cfg hinterlegt. Dort finden sich Parameter wie:

- Die Erlang Node des Namensdienstes
- Die Anzahl der GGT-Prozesse pro Starter
- Die Verzögerungszeit der GGT-Prozesse in Millisekunden (simuliert einen Arbeitsaufwand bei der Berechnung)
- Terminierungszeit der GGT-Prozesse in Millisekunden (Zeit nach der die Prozesse eine Terminierungsanfrage starten wenn sie keine Nachrichten mehr bekommen)

Bsp:

```
{arbeitszeit, 2}.
{termzeit, 3}.
{ggtprozessnummer, 3}.
{nameservicenode, ns@Brummpa}.
{nameservicename, nameservice}.
{koordinatorname, chef}.
{korrigieren, 0}.
```

Das Verhalten in den einzelnen Phasen ist nochmal in einem jeweiligen Prozessdiagramm festgehalten

- Initialisierungsphase siehe Anhang 1
- Arbeitsphase siehe Anhang 2
- Beendigungsphase siehe Anhang 3

3.2 Methoden

Start() :: void -> Atom

Die Methode startet den Koordinator.

3.3 Befehle

Der Koordinator kann eine Reihe von Befehlen über den ZooKeeper erhalten:

- **reset:** Der Koordinator sendet allen GGT-Prozessen das kill-Kommando und bringt sich selbst in den initialen Zustand, indem sich Starter wieder melden können.
- **step:** Der Koordinator beendet die Initialphase und bildet den Ring. Er wartet nun auf den Start einer ggT-Berechnung
- **prompt:** Der Koordinator erfragt bei allen ggT-Prozessen per tellmi deren aktuelles Mi ab und zeigt dies im log an.
- **nudge**: Der Koordinator erfragt bei allen ggT-Prozessen per pingGGT deren Lebenszustand ab und zeigt dies im log an.

- **toggle:** Der Koordinator verändert den Flag zur Korrektur bei falschen Terminierungsmeldungen.
- **{calc,WggT}:** Der Koordinator startet eine neue ggT-Berechnung mit Wunsch-ggT WggT.
- kill: Der Koordinator wird beendet und sendet allen ggT-Prozessen das kill-Kommando.

4 GGT-PROZESS

Der GGT-Prozess meldet sich beim Namensdienst, Koordinator und der Erlang Node an. Dadurch erhält er vom Koordinator die Arbeitszeit und die Wartezeit. Wenn der Koordinator die Prozesse in einen Ring aufstellt werden ihm sein rechter und linker Nachbar bekannt gegeben, sowie ein Startwert.

Werden einige von ihnen „angestubst“ beginnt die Berechnung. Die Prozesse berechnen ihren GGT mittels des Satzes von Euklid und warten die vom Koordinator vorgegebene Arbeitszeit um eine Berechnungszeit zu simulieren, die sonst bei solch trivialen Berechnungen nicht auftreten würden. Hat er seine neue Zahl berechnet schickt er diese an den Koordinator. Wenn der Prozess für eine gewisse Zeit (Wartezeit) gewartet und keine neue Zahl erhalten hat, startet er eine Abstimmungsumfrage indem es eine Abstimmungsnachricht an seinen rechten Nachbarn weitersendet. Dieser kann wenn er ebenfalls min Wartezeit/2 gewartet hat die Anfrage nach rechts weiter reichen. Wenn nicht ignoriert er sie einfach. Ein Prozess der die Anfrage von Links bekommt und als Absender sich selbst sieht, erkennt sich dadurch als Initiator und sendet die Anfrage sammt des errechneten Wertes an den Koordinator.

Terminiert wird der GGT-Prozess durch einen „kill“ Befehl des Koordinators. Dadurch meldet sich der Prozess beim Namensdienst ab und beendet sich dann.

Die Konfigdatei des GGT-Prozesses beinhaltet die für den Namen wichtigen Parameter und sieht z.B so aus:

Ggt.cfg

```
{praktikumsgruppe, 4}.
{teamnummer, 88}.
{nameservicenode, ns@Brummpa}.
{nameservicename, nameservice}.
{koordinatorname, chef}.
```

Jeder GGT-Prozess hat einen Namen der sich wie folg zusammensetzt:

<PraktikumsgruppenID><TeamID><Nummer des GGT-Prozess><Nummer des Starters>,

Das Verhalten ist in einem Prozessdiagramm festgehalten siehe Anhang 4

4.1 METHODEN

Die vom GGT-Prozess bereit gestellten Methoden sind:

start(Arbeitszeit, Wartezeit, Name) :: Nummer x Nummer x String -> Atom

- Arbeitszeit (Zeit die der Prozess das Arbeiten durch Warten simulieren soll in Millisekunden)
- Wartezeit (Zeit bevor eine Terminierungsanfrage gestartet wird in Millisekunden)
- Name (Name des GGT-Prozesses)

Der Rückgabewert ist ein einfaches OK wenn der Prozess vom Koordinator beendet wird.

5 STARTER-PROZESS

Der Starter Prozess meldet sich beim Koordinator und bekommt von ihm seine initialen Werte, darüber wie viele GGT-Prozesse er starten soll. Alle weiteren Daten werden aus der ggt.cfg ausgelesen wie:

- die Erlang-Node des Namensdienstes,
- der Name des Koordinators,
- die Nummer der Praktikumsgruppe
- die Nummer des Teams

Dann startet er die gewünschte Anzahl an GGT-Prozessen und gibt ihnen folgende Werte mit:

- der Verzögerungszeit
- die Warteszeit
- den Namen

5.1 METHODEN

Die Funktion startet den Starter Prozess
Start(StarterNummer) :: Nummer -> Atom

StarterNummer -> Nummer des Starters bei der Erzeugung
Atom -> Wenn alle Prozesse gestartet wurden gibt die Funktion ein einfaches OK zurück

6 ZooKeeper

Der ZooKeeper ermöglich es dem Benutzer Einfluss auf das System auszuüben. Welche Möglichkeiten dem Benutzer gegeben werden ist in 3.2 Befehle definiert.

6.1 Methoden

Start() :: void -> void

Die Methode startet den Zookeeper

reset() :: void -> Atom
sendet den Reset Befehl an den Koordinator und gibt seine Antwort zurück
step() :: void -> Atom
sendet den Step Befehl an den Koordinator und gibt seine Antwort zurück
prompt:: void -> Atom
sendet den Prompt Befehl an den Koordinator und gibt seine Antwort zurück
nudge :: void -> Atom
sendet den nudge Befehl an den Koordinator und gibt seine Antwort zurück
toggle:: void -> Atom
sendet den togglet Befehl an den Koordinator und gibt seine Antwort zurück
calc(WggT) :: Zahl -> Atom
WggT -> Zahl deren GGT berechnet werden soll
sendet den calc Befehl an den Koordinator und gibt seine Antwort zurück

7 Nachrichten und Datentypen

7.1 Zustand „Initial“

7.1.1 Starter

Koordinator ! {getsteeringval, PID}
PID -> ProzessID des Starters.
Starter ! {steeringval, Arbeitszeit, TermZeit, GGTProzessAnzahl}
Arbeitszeit -> Simulierte Rechenzeit des GGT-Prozesses in Millisekunden.
TermZeit -> Wartezeit eines GGT-Prozesses auf eine neue Nachricht bis eine Abstimmung gestartet wird.
GGTProzessAnzahl -> Anzahl der zu erzeugenden GGT-Prozesse
Log: Starter <PID> Angemeldet

7.1.2 GGT-Prozess

Koordinator ! {hello, Clientname}

Clientname -> Atom welches den Clientnamen repräsentiert. Dieser Name kann vom Namensdienst in eine eindeutige ProzessID übersetzt werden.

Log: PID- ggT-Prozess angemeldet <Timestamp>

7.1.3 ZooKeeper

Koordinator ! {step, PID}

PID -> ProzessID des Zookeepers

Der Koordinator wechselt in die Arbeitsphase baut den GGT-Prozess-Ring auf. Danach wartet er auf das Signal zum Start der GGT-Berechnung

ZooKeeper ! ok

Erfolgsmeldung für den Wechsel in die Arbeitsphase

7.2 Zustand „bereit“

7.2.1 GGT-Prozess

GGT-Prozess ! {sendy, Y}

Y -> Neu aufgerufenes Y

Rekursiver Aufruf der GGT-Berechnung.

Koordinator ! {briefmi, {Clientname, NewMi, CZeit}}

ClientName -> Atomarer Name des reportierenden Prozesses

NewMi -> Neue ermittelte Mi

CZeit -> Uhrzeit als String zu der dieser neue Wert ermittelt wurde

GGT-Prozess ! {abstimmung, Initiator}

Initiator -> atomarer Name des Initiators der Abstimmung.

Abstimmung über die Terminierung der Berechnung. Wenn der empfangende GGT-Prozess der Initiator ist wird eine Nachricht "briefterm" an den Koordinator verschickt.

Koordinator ! {briefterm, {Clientname, NewMi, CZeit}}

ClientName -> Atomarer Name des reportierenden Prozesses

NewMi -> Neue ermitteltes Mi

CZeit -> Uhrzeit als String zu der dieser neue Wert ermittelt wurde

Terminierungsnachricht des GGT-Prozesses an den Koordinator mit dem entsprechenden Wert

GGT-Prozess ! {tellmi, From}

From -> Atomarer Name des anfragenden Prozesses

From ! {mi, Mi}

Mi -> Aktuelles Mi

7.2.2 Koordinator

GGT-Prozess ! {setpm, NewMi}

NewMi -> Neues Mi für den jeweiligen GGT-Prozess

Setzt das Mi im empfangenen GGT-Prozess neu.

GGT-Prozess ! {setneighbors, LeftNeighbor, RightNeighbor}

LeftNeighbor -> Atom-name des neuen linken Nachbarn des GGT-Prozesses.

RightNeighbor -> Atom-name des neuen rechten Nachbarn des GGT-Prozesses.

GGT-Prozess ! {sendy, Y}

Y -> Neu aufgerufenes Y

"Anstupsen" eines GGT-Prozesses zum weiterführen oder starten der Berechnung.

GGT-Prozess ! kill

Der GGT-Prozess meldet sich vom Namensdienst ab und beendet dann sofort alle Kind-Prozesse und sich selber.

GGT-Prozess ! {tellmi, From}

From -> Atomarer Name des Koordinators

From ! {mi, Mi}

Mi -> Aktuelles Mi

GGT-Prozess ! {pingGGT, From}

From -> atomarer Name des Anfragenden Prozesses

From ! {pongGGT, GGTname}

GGTName -> atomarer Name des Antwortenden Prozesses

7.2.3 ZooKeeper

Koordinator ! {calc, WunschGGT, PID}

WunschGGT -> Wunsch-GGT der berechnet werden soll.

PID -> ProzessID des Zookeepers

Nach dem Erhalten dieser Nachricht wird eine neue GGT-Berechnung gestartet.

ZooKeeper ! ok

Wenn eine Berechnung gestartet werden kann.

ZooKeeper ! error

Wenn bereits eine Berechnung ausgeführt wird.

Koordinator ! {reset, PID}

PID -> ProzessID des Zookeepers

Der Server sendet jedem GGT-Prozess die Nachricht "kill" und bringt sich selber zurück in den Zustand "initial"

ZooKeeper ! ok

Erfolgsmeldung für den Wechsel in die Arbeitsphase

Koordinator ! {step, PID}

Ungültige step-Nachricht

ZooKeeper ! error

Fehlernachricht, da step in der "bereit"-Phase ein ungültiger Befehl ist

Koordinator ! {prompt, PID}

Aufforderung des ZooKeepers die aktuellen Mis aller GGT-Prozesse abzufragen und im Log einzutragen.

ZooKeeper ! ok

Bestätigung der Anfrage

Koordinator ! {nudge, PID}

Aufforderung des ZooKeepers die den Lebenszustand aller GGT-Prozesse abzufragen und im Log einzutragen.

ZooKeeper ! ok

Bestätigung der Anfrage

Koordinator ! {toggle, PID}

Aufforderung des ZooKeepers das Flag zur Korrektur bei falschen Terminierungsmeldungen umzusetzen.

ZooKeeper ! {ok, NewFlag}

NewFlag -> Neuer Wert des Flags

Koordinator ! {kill, PID}

Aufforderung des ZooKeepers an den Koordinator in den Zustand "beenden" zu wechseln.

ZooKeeper ! ok

Erfolgsmeldung für den Wechsel in die Beendigungsphase

Zustand „beenden“

7.2.4 Koordinator

GGT-Prozess ! kill

Der GGT-Prozess meldet sich vom Namensdienst ab und beendet dann sofort alle Kind-Prozesse und sich selber.

8 LOGGING

8.1 KOORDINATOR

8.1.1 „Initialisierungs“ Phase

GGT-Prozess meldet sich an

<Name> ggT-Prozess angemeldet <Timestamp>

Starter-Prozess meldet sich an

Steuernde Werte an Starter <PID> versandt

8.1.2 „Arbeits“ Phase

ZooKeeper fordert per prompt die Werte aller Prozesse an

ggT-Prozess: <Name> mit MI <MI>

ZooKeeper fordert per nudge den Lebenszustand aller Prozesse an

ggT-Prozess: <Name> mit <Zustand>

Terminierungsanfrage gestellt:

ggT-Prozess: <Name> stellt mit <MI> eine Terminierungsanfrage

Terminierungsanfrage zu gelassen

ggT-Prozess: <Name> beendet Berechnung, Ergebnis <MI>

Terminierungsanfrage unzulässig

ggT-Prozess: <Name> Terminierungsanfrage abgelehnt, errechneter Wert <MI> richtiger Wert <richtiger Wert>

8.1.3 „Beendigungsphase“ Phase

Koordinator fährt das System herunter

System wird heruntergefahren

8.2 GGT-Prozesse

Nachbarn werden bekannt gegeben

Nachbarn gesetzt linker Nachbar <Name> rechter Nachbar <Name>

Setpm Empfangen

Setpm empfangen, setzte MI auf <neueMI>

Sendy empfangen

Neues MI wird berechnet mit <MI> und <Y>

Abstimmung starten

Starte Abstimmung mit <MI> berechnetem Wert

Abstimmung erhalten und angenommen

Abstimmungsanfrage erhalten und weitergeschickt

Abstimmung erhalten und abgelehnt

Abstimmungsanfrage ignoriert

Tellmi erhalten

Tellmi erhalten und mit <MI> geantwortet

PingGGT erhalten

Ping erhalten und beantwortet

Kill erhalten

Kill erhalten, abmelden und beenden

9 Nachtrag – Änderungen

9.1 Nachtrag aufgrund von Uneindeutigkeiten bei erster Abgabe

Der GGT-Prozess schaut bei Abstimmungsnachrichten nicht nur ob eine Nachricht von Links kommt, sondern auch nach dem Verfasser um zu entscheiden ob er der Initiator der Abstimmung ist. Dies wurde nicht im Text, sondern nur im Prozessdiagramm eindeutig.

9.2 Praktikum

In unserer Implementation lasen wir aus der Aufgabenstellung, dass bei der Kommunikation jeweils der Name des Prozesses mitgeschickt wird. Diesen haben wir dann jeweils beim Namensdienst nachgeschlagen und daran verschickt. Wir haben im Nachhinein Änderungen eingeführt die darauf prüfen in welcher Art der Identifikation übergeben wird und senden entweder direkt oder sehen weiterhin beim Namensdienst ab und verschicken nicht mehr nur den Namen beim Senden sondern die PID