



6. NOVEMBER 2014

ENTWURF

VERTEILTE SYSTEME PRAKTIKUM 2

STEFFEN GIERSCHE & 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

Aktueller Stand:

- ❖ Entwurfsdokument fertig gestellt
- ❖ Implementation begonnen

1 INHALTSVERZEICHNIS

1	Inhaltsverzeichnis.....	2
2	Funktionalität.....	3
2.1	Komponenten	3
2.2	Phasen	3
2.2.1	Initialisierungsphase.....	3
2.2.2	Arbeitsphase	4
2.2.3	Beendigungsphase	4
3	Koordinator	4
3.1	Allgemeines	4
3.2	Methoden.....	5
3.3	Befehle	5
4	GGT-Prozess.....	6
4.1	Methoden.....	6
5	Starter-Prozess.....	7
5.1	Methoden.....	7
6	ZooKeeper	7
6.1	Methoden.....	8
7	Nachrichten und Datentypen.....	8
7.1	Zustand „Initial“	8
7.1.1	Starter.....	8
7.1.2	GGT-Prozess	8
7.1.3	ZooKeeper.....	9
7.2	Zustand „bereit“	9
7.2.1	GGT-Prozess	9
7.2.2	Koordinator.....	10
7.2.3	ZooKeeper.....	10
7.2.4	Koordinator.....	12
8	Logging	12
8.1	„Initialisierungs“ Phase.....	12
8.2	„Arbeits“ Phase	12
8.3	„Beendigungsphase“ Phase.....	13

2 FUNKTIONALITÄT

Es soll eine Anwendung geschrieben werden die mittels verteilter GGT-Prozesse 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 User 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 unterteilt die sich kurz in

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 "max_wait_time" 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 "max_wait-time" / 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 auf das System durch gewisse Nachrichten Einfluss zu nehmen.

Die nötigen Config 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 Terminierungszeit. 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“ werden 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 (Terminierungszeit) gewartet hat 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 Terminierungszeit/2 gewartet hat die Anfrage nach rechts weiter reichen. Wenn nicht ignoriert er sie einfach. Ein Prozess der die Anfrage von Links bekommt, 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 Rückgabe wert ist ein einfaches OK wenn das starten gelingt.

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 Terminierungszeit
- der Startnummer dieses Prozesses (also der wievielte gestartete ggT-Prozess er ist)
- seine eindeutige Starternummer
- die Praktikumsgruppennummer
- die Teamnummer
- Namensdienst
- Koordinator.

5.1 METHODEN

Die Funktion startet den Starter Prozess

Start(StarterNummer) :: Nummer -> void

StarterNummer -> Nummer des Starters bei der Erzeugung

6 ZOOKEEPER

Der ZooKeeper ermöglicht 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 „INITIALISIERUNGS“ PHASE

GGT-Prozess meldet sich an

PID- ggT-Prozess angemeldet <Timestamp>

Starter-Prozess meldet sich an

Starter <PID> Angemeldet

8.2 „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>

8.3 „BEENDIGUNGSPHASE“ PHASE

Koordinator fährt das System herunter

System wird heruntergefahren