Verteile Systeme

Praktikum 2

Dokumentationskopf

Team

Team 14

- Anton Romanov
- Benjamin Rexin

Aufgabenaufteilung

Anton Romanov: *starter.erl, ggT.erl*, Dokumentation
Ben Rexin: koordinator.erl, Dokumentation

Quellenangabe

ggT.erl - Eduard Weigandt, Ivan Demin
shuffle.erl - Adam Lindberg
coordinator.erl - Patrick Detlefsen

Bearbeitungszeitraum

2012-11-08 - 4 Stunden Ben
2012-11-12 - 1 Stunde beide
2012-11-12 - 4 Stunden Ben
2012-11 - 10 Stunden Anton

Aktueller Stand

Aktuell ist befindet sich der Koordinator kurz vor der Fertigstellung. Der Starter ist fertig. ggT-Prozess muss noch getestet werden

Änderungen im Entwurf

- Mittlerweile haben wir den Koordinator von *Patrick Detlefsen* übernommen, damit ergibt sich, dass die verwendenten Strukturen nicht mehr übereinstimmen. So verwendet der Coordinator ein `orddict` anstelle einer `list`.
- Die Logs von ggT-Prozessen werden mit Timestamps versehen

- Der ggT-Prozess erwartet auch in der Arbeitsphase eine “setpm”-Nachricht und kann sofort in die Initialisierungsphase wechseln.
- [FIXED] Bei der Berechnung von neuem Mi-Wert wurde nicht der neue Wert zurückgegeben, sodass die Nachbarn nicht den neuen Wert erhalten haben
- [FIXED] Der Koordinator hat die PIDs anstatt von Namen an die Nachbarn übertragen
- Wartezeit in after-Blöcken vom ggT-Prozess wurd erhöht
- ggt.erl : Config wird an algo, für's logging, übergeben.
- starter.erl : Es wurde ein global:sync() nach dem pingen des Nameservice-Nodes eingefügt.
- coordinator.erl: Aufrufe von send_message_to_service werden vermieden, und dafür direkt die #gcd_client.servicepid verwendet.

Entwurf

Koordinator

Phase 1 - Initialisierung

Steuerwerte Abfragen - getsteeringval

Der Starter Prozess hat die Möglichkeit mit *getsteeringval* für Ihn wichtige Konfigurationsparameter abzufragen. Als Ergebnis erhält der Startprozess die Parameter die simulierte Arbeitszeit, Wartezeit bis zu Terminierung und die Anzahl der zu startenden Prozesse.

Client Anmeldung - hello

Mit der *hello* Nachricht meldet sich ein GGT Prozess am Koordinator an. Auf diese Nachricht erhält er direkt keine Antwort. Allerdings wird, zur späteren Nutzung, sein Name in einer Clients-Liste (`orddict` in der Implementierung) aufgenommen.

Manuelles Starten - start

Dem Nutzer ist es möglich, mit hilfe des *start* Kommandos den Koordinator manuell mitzuteilen wann er in die Bearbeiten-Phase übergehen soll. Beim erhalt der Nachricht werden die Clients zufällig sortiert, anschließend werden 15% (mindestens jedoch zwei) der Clients gewählt und an eben diese der Start der Berechnung kommuniziert.

Phase 2 - Bearbeiten

CMI Information - briefmi

Ein Client Informiert mit seinem Namen, dem Ergebnis und seiner aktuellen Systemzeit über ein gefundenes Ergebnis.

CMI Terminiert - briefterm

Sofern der Client in einer definierten Zeit keine neue Zahl zur Berechnung erhalten hat, startet er einer Terminierungsabstimmung und informiert den Koordinator mit der *briefterm* bei erfolg über diesen Zustand.

reset

Der Nutzer hat mit der *reset* Nachricht ein Werkzeug das gesamte System zurück zu setzen, dafür sendet der Koordinator allen GGT Prozessen ein *kill* und wechselt selbst wieder in die Initialisierungsphase.

kill

Beim erhalten der *kill,* durch den Nutzer, wechselt der Koordinator in die Beenden-Phase.

Phase 3 - Beenden

In dieser Phase räumt der Koordinator auf, dazu gehört den angeschlossenen Clients mitzuteilen das sie ihre Arbeit beenden und herunterfahren sollen.

ggT-Prozess

Der ggT-Prozess besteht im Wesentlichen aus einer Initialisierungmethode und einer Loop-Methoden. Der ggT-Prozess bekommt vom Starter folgende Parameter:

- KoordName
- Arbeitzeit
- TermZeit
- ProzessNR
- StarterNR
- NamensDPID
- Praktikumsgruppe
- TeamID

In der Initialisierungsmethode findet Folgendes statt:

1. lokale Regisrierung (register(...))
2. Registrierung beim Namensdiesnt (rebind)
3. Anmelden beim Koordiantor (hello)
4. Erhalten der Mitteilung über die Nachbarn vom Koordinator (setneighbors)
5. Auflösung vom rechten und linken Nachbar (lookup)
6. Erhalten des Mi-Wertes vom Koordinator (setpm)

In der Loop-Methode findet die tatsächliche Arbeit statt:

1. Abstimmung
    a. Wenn es zu einem Timeout kommt, findet eine Abstimmung statt. Dabei wird dem rechten Nachbar {abstimmung, myID} gesendet. Kommt die selbe Nachricht bei “unserem” ggT-Prozess an, so gehen wir davon aus, dass alle anderen Knoten auch für die Terminierung abgestimmt haben. Dann wird “brieftem” dem Koordinator gesendet. Wenn myID nicht unserem ID entspricht, so leiten wir die Anfrage an den rechten Nachbar weiter.

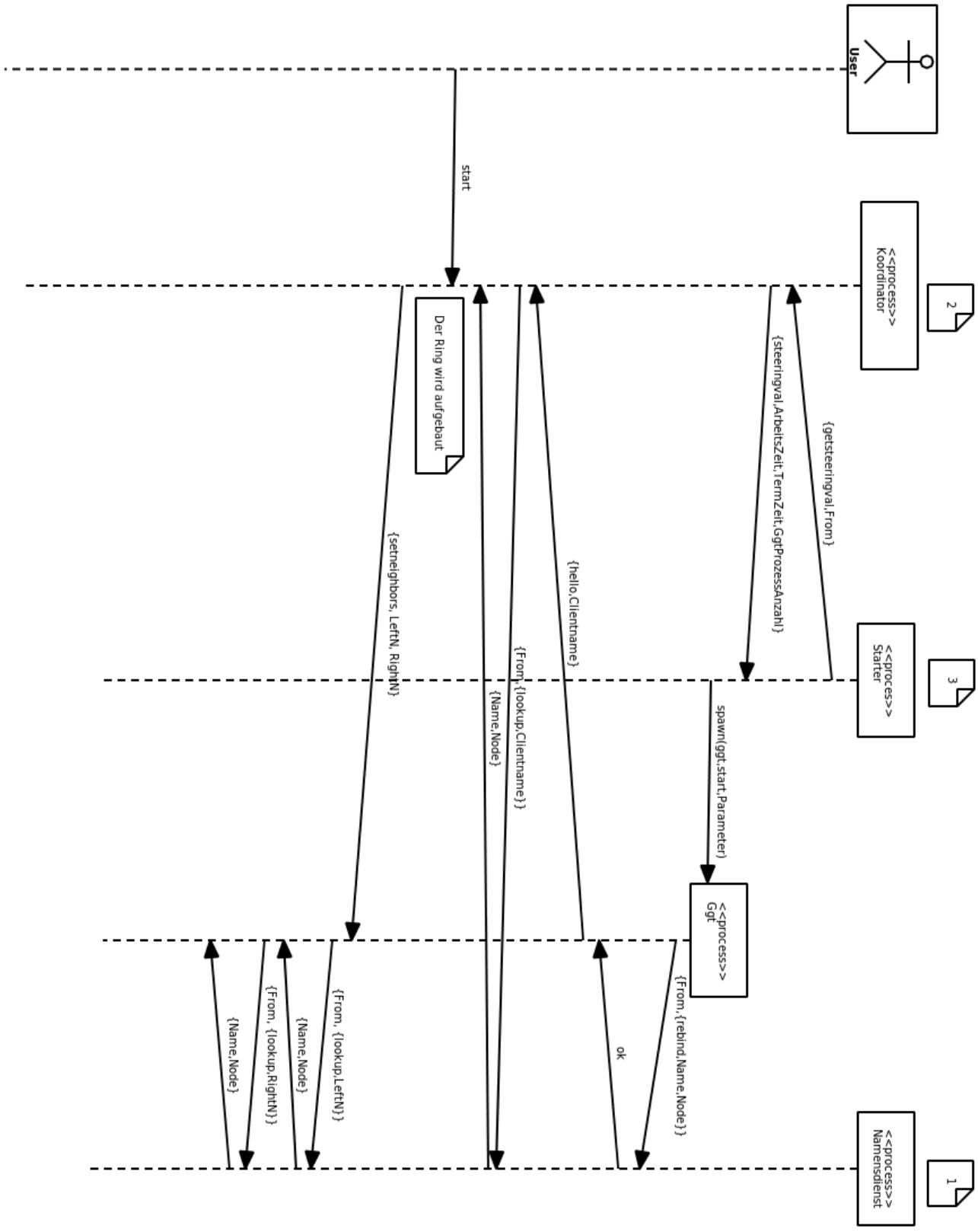
   b. Auf Timeout wird an 2 Stellen geprüft. Ganz am Anfang der Loop-Methode (beim ersten Aufruf nicht wirklich wichtig, aber bei weiteren Aufrufen (durch Rekursion) macht es Sinn hier gleich die “Abbruchbediengung” zu überprüfen und ggf. die Abstimmung starten) und im after-Teil des Receive-Blocks.
   c. Der Zeitpunkt, wann die letzte für die Terminierung relevante Nachricht (sendy, setpm) ankam, wird in einem Record gespeichert.
2. Rekursiver Aufruf der ggT-Berechnung (sendy)
   a. Ändert sich der Mi-Wert , so werden beide Nachbarn und der Koordinator darüber informiert.

In beiden Methoden wird die kill-Nachricht erwartet. Kommt eine solche Nachricht an, wird der Namensdienst darüber informiert (unbind) und der ggT-Prozess wird tatsächlich beendet
Die Eindeutigkeit in Logs wird durch die Benennung der Log-Dateien erreicht (GGTP_myID@Host).

Starter

Der Starter wir mit einer eindeutigen Nummer initialisiert. Er lernt die Nameservie kennen (ping + whereis_name) und findet (lookup) über sie den Koordinator. Der Starter meldet sich beim Koordinator mit *getsteeringval* . Mit dem Erhalten der steeringval-Nachricht werden so viele ggT-Prozesse gestartet wie in der .cfg-Datei festgelegt ist.

Sequenzdiagramm (ggT-Initialisietungsphase)


User
start
<<process>>
Koordinator
2
Der Ring wird aufgebaut
{steeringval.ArbeitsZeit,TermZeit,ggtProzessAnzahl}
{getsteeringval,From}
{hello,Clientname}
{From,{lookup,Clientname}}
{Name,Node}
{setneighbors, LeftN, RightN}
<<process>>
Starter
3
spawn(ggt,start,Parameter)
<<process>>
Ggt
{From,{rebind,Name,Node}}
ok
{From, {lookup,LeftN}}
{Name,Node}
{From, {lookup,RightN}}
{Name,Node}
<<process>>
Namensdienst
1

Sequenzdiagramm (ggT-Arbeitsphase) [möglicher Ablauf]