Aufgabe 2 Verteilter ggt Algorithmus

In dieser Aufgabe ist ein einfacher **verteilter Algorithmus** und **dessen Koordination** zu implementieren. Jeder Arbeitsprozess (ggT-Prozess) durchläuft den gleichen Algorithmus! Mit diesem Algorithmus kann man z.B. mit n ggT-Prozessen den ggT von n Zahlen nebenläufig bestimmen.

Lesen Sie sich die Aufgabenstellung sorgfältig, damit ihre Lösung am Ende ähnliche Resultate zeigt, wie die in den mitgelieferten Logs.

Aufgabenstellung

Das System für den verteilten Algorithmus ist so ausgelegt, dass es für eine längere Zeit installiert wird und dann für **mehrere ggT-Berechnungen** zur Verfügung steht, ohne dass die beteiligten Komponenten neu gestartet werden müssen! Für die Implementierung werden im Wesentlichen drei Module benötigt:

1. Der **Koordinator**, der den verteilten Algorithmus verwaltet:
    1. Hochfahren des Systems
    2. Start einer ggT-Berechnung
    3. Herunterfahren des Systems
    4. Der Koordinator verfügt über eine GUI (Textausgabe reicht aus), in der der Ablauf des Algorithmus beobachtet werden kann.
2. Den **Starter**, der das Starten von Prozessen auf für den Koordinator entfernten Rechnern ermöglicht.
3. Der **ggT-Prozess**, der den ggT verteilt berechnet

Alle Module erstellen ausführliche und eindeutig ihnen zugeordnete Logdateien. Zur Kommunikation wird das Message-Passing von Erlang/OTP verwendet.

Der Algorithmus: Der Algorithmus basiert auf dem ***Satz von Euklid***: Der größte gemeinsame Teiler (ggT) zweier positiver ganzer Zahlen x, y (mit x>=y>0) ist gleich dem ggT von y und dem Rest, der bei ganzzahliger Division von x durch y entsteht. Der verteilte Algorithmus lautet:

- Alle Prozesse sind in einem Ring angeordnet.
- Jeder Prozess Pi hat seine eigene Variable Mi, in der die von ihm zu verwaltende Zahl steht.
- Der ggT aller am Anfang bestehender Mi wird wie folgt berechnet:

```
{Eine Nachricht <y> ist eingetroffen}
if y < Mi
  then Mi := mod(Mi-1,y)+1;
       send #Mi to all neighbours;
fi
```

Initialisierungsphase:

- Starten des Namensdienst mit globaler Registrierung
- Starten des Koordinators mit Registrierung beim Namensdienst registriert
- Starten der Starter mit Erfragen der Konfigurationswerte beim Koordinator für die ggT-Prozesse (**get_ggt_vals**).

- Starten der ggT Prozesse nach den Vorgaben des Koordinators durch die Starter. Die ggT Prozesse registrieren sich beim Namensdienst (**rebind**) und melden sich beim Koordinator an (**check_in**).
- Aufbau eines zufällig gemischten Rings durch den Koordinator (Trigger ist die Nachricht **step**), der die ggT Prozesse über ihren linken und rechten Nachbarn informiert (**set_neighbours**).

- Beginn der Arbeitsphase

Arbeitsphase:

- Koordinator:
  - Initialisierung für die nächste Berechnung = Informieren aller ggT Prozesse über ihren Wert (Mi) mit **set_pmi**.
  - Zufälliges Auswählen von 15% aller ggT-Prozesse (mindestens zwei), die mit der Berechnung beginnen sollen (Trigger **send**).
- ggT-Prozess:
  - Verändert sich auf Grund einer Berechnung der Wert #Mi, so informiert er den Koordinator darüber (**brief_mi**) und seine linken und rechten Nachbarn (**send**).
  - Wird die Terminierung der aktuellen Berechnung festgestellt (zeitgesteuert), so meldet er dem Koordinator das Endergebnis der Berechnung (**brief_term**). Das System steht für weitere Berechnungen zur Verfügung!
  - Erhält der Prozess während einer Berechnung **<ttt>** Sekunden lang keine Zahl (**send** oder **set_pmi**), startet er eine **Terminierungsabstimmung**. Er seinen linken Nachbarn, ob dieser bereit ist, zu terminieren (**vote**). Verläuft die Abstimmung durch den ganzen Ring positiv, sendet er dem Koordinator eine entsprechende Mitteilung über die Terminierung der aktuellen Berechnung.

Beendigung(sphase):
- Starter: in der Initialisierungsphase nach dem Start aller ggT-Prozesse
- ggT-Prozesse: über **kill**, initiiert durch den Koordinator:
  - der ggT Prozess terminiert unverzüglich, unabhängig von seinem aktuellen Zustand, insbesondere auch dann, wenn er in der Berechnung „steckt“
  - die einzige noch zulässige Aktion ist, sich beim Namensdienst abzumelden (**unbind**).
- Koordinator: terminiert, nachdem er ein **kill** an alle ggT-Prozesse gesendet hat
  - Er meldet sich vorher beim Namensdienst ab.

Funktionalität

Koordinator

1. Die benötigten steuernden Werte werden aus der Datei koordinator.cfg ausgelesen. Dazu gehören
   - Erlang-Node des Namensdienstes
   - Registrierungszeit **<rt>** für die Starter in Sekunden
   - Anzahl der ggT-Prozesse pro Starter

- Verzögerungszeit der ggT-Prozesse **<ttw>** als simulierter Arbeitsaufwand für die Berechnung in Sekunden
- Terminierungszeit **<ttt>**, nach der ein ggT-Prozess eine Abstimmung durchführt, in Sekunden

2. Nach dem Start des Koordinators können sich Starter und/oder ggT-Prozesse innerhalb der **<rt>**Sekunden bei ihm melden. In diesem Zustand (**register**) können
   - Startern die benötigten Informationen über Anzahl der zu startenden ggT-Prozesse, deren jeweilige Verzögerungszeit **<ttw>** und deren Terminierungszeit **<ttt>** erfragen.
   - In seiner GUI (Textausgabe) und seiner Logdatei werden die sich angemeldeten ggT-Prozesse angezeigt.
3. Nach den **<rt>** Sekunden geht der Koordinator in den Zustand **init** über. Er gibt keinem Starter mehr Auskunft und registriert keine ggT-Prozesse mehr!
4. Über **step** wird der Koordinator veranlasst den ggT Ring aufzubauen. Die Reihenfolge soll zufällig bestimmt werden. Danach wechselt der Koordinator inf den Zustand **ready**.
5. Im Zustand **ready**: Starten einer Berechnung über die Nachricht {**calc target**}
   - Der Koordinator informiert alle ggT-Prozesse über ihre Startwerte (**set_pmi**)
   - **target** ist der gewünschte ggt (dieser ist zur manuellen Überprüfung der Berechnung gedacht). Mit **werkzeug:bestimme_mis/2** werden n (n=Anzahl der ggT Prozesse) Vielfache von target berechnet, die die Startwerte für die ggT Prozesse sind.
   - Er startet die Berechnung und sendet 15% der ggT Prozesse (mindestens 2 und zufällig gewählt) eine Zahl (Vielfaches von **target**) über **send**
   - Der Koordinator wird von den ggT-Prozessen über deren Terminierung informiert (**brief_term**). Sollte die gesendete Zahl größer als die bisher bekannte kleinste Zahl sein, notiert der Koordinator diese Fehlmeldung in seiner log. Ist ein spezielles Flag (Nachricht **toggle**) gesetzt, sendet er dem ggT-Prozess die kleinste Zahl per **send**.
   - Per manueller Eingabe kann der Koordinator in den Zustand "beenden" (Nachricht **kill**) oder in den Zustand **register** (Nachricht **reset**) versetzt werden. Beim Übergang in den Zustand **register** wird die Konfigurationsdatei des Koordinators erneut gelesen.
   - Es ist möglich jederzeit eine neue ggt-Berechnung zu starten.
6. Die Nachrichten **kill, reset** und **toggle** sollen in allen "aktiven" Zuständen des Koordinators möglich sein.
7. In seiner GUI (Textausgabe) und seiner Logdatei zeigt der Koordinator alle Nachrichten der ggT-Prozesse und Starter an.
8. Ist der Koordinator im **Zustand "beenden"** informiert er die ggT-Prozesse über die Beendigung (kill).
9. **Der Koordinator ist in Erlang/OTP** zu implementieren und muss auf jedem Rechner im Labor startbar sein!

Starter

1. Der Starter (mit eindeutiger Nummer) erfragt beim Koordinator die steuernden Werte (**get_ggt_vals**) asynchron und erwartet einen entsprechenden Rückruf (**ggt_vals**).
2. Der Starter liest aus der Datei **ggt.cfg** die folgenden Werte:
   - Erlang-Node des Namensdienstes
   - Name des Koordinators

- Nummer der Praktikumsgruppe
- Nummer des Teams

3. Der Starter startet die ggT-Prozesse mit den zugehörigen Werten:
   - **<ttw>**,
   - **<ttt>**,
   - Startnummer des ggT Prozesses
   - Kontaktdaten für Namensdienst und Koordinator
4. Beim Starten des Starters wird die Starternummer mitgegeben.
5. **Der Starter ist in Erlang/OTP** zu implementieren und muss auf jedem Rechner im Labor startbar sein!

ggT-Prozess

1. Ein ggT-Prozess hat den Namen, der eine Zahl ist, die sich wie folgt zusammensetzt:
   <PRGruppenID><TeamID><Nummer ggT-Prozess>_<Nummer des Starters>,
   Bsp.: **232_1** bezeichnet den 2'ten ggT Prozess des Starters mit Nummer 1 des 3'ten Teams der 2'ten Praktikumsgruppe **In der gesamten Kommunikation mit externen Prozessen wird ausschließlich dieser Name als atom verwendet!**
2. Zustand **initial**:
   - Der ggT-Prozess meldet sich beim Koordinator mit seinem Namen an (**check_in**) und registriert sich beim Namensdienst (**rebind**). Er registriert sich ebenfalls lokal auf der Erlang-Node mit seinem Namen (**register**).
   - Der ggT-Prozess erwartet dann die Informationen über seine Nachbarn (**setneighbours**). Wechsel in den Zustand **pre_process**
3. Zustand **pre_process**: Der ggT-Prozess erwartet vom Koordinator seine Zahl Mi (**set_pmi**). Wechsel in den Zustand **process**.
4. Der ggT-Prozess kann zu jeder Zeit zu einer neuen Berechnung aufgefordert werden!
5. Zustand **process**:
   - Wenn er eine Zahl erhält (**send**) führt er den ggT-Algorithmus aus. Ändert sich seine Zahl dadurch, informiert er zusätzlich den Koordinator (Nachricht **brief_mi**). Ändert sich seine Zahl dadurch nicht, macht der ggT-Prozess gar nichts und erwartet die nächste Nachricht.
   - Für eine ggT-Berechnung braucht er jedoch eine gewisse Zeit (**<ttw>**). Dies simuliert eine größere, Zeit intensivere Aufgabe. Der ggT-Prozess soll in dieser Zeit einfach nichts tun (**timer:sleep**). Während einer echten Berechnung ist der **<ttt>** Timer zu unterbrechen. Nach der Berechnung werden der **<ttt>** Timer und das letzte Empfangsereignis aktualisiert.
   - Der ggT-Prozess beobachtet die Zeit seit dem letzten Empfang einer Zahl (**send** oder **set_pmi**). Hat diese **<ttt>** Sekunden überschritten, startet er eine Terminierungsanfrage / Abstimmung (**vote**). Es wird (von ihm) nur genau eine Terminierungsanfrage gestartet. Eine weitere kann frühestens dann gestartet werden, wenn zwischenzeitlich eine Zahl (**send**, **set_pmi**) an ihn gesendet wurde!
   - Ist die Abstimmung erfolgreich (**vote** wird ihm mit seinem Namen gesendet), sendet er dem Koordinator eine Mitteilung über die Terminierung (**brief_term**) mit seinem Namen, dem errechneten ggT (sein aktuelles Mi) und seine aktuelle Systemzeit. Zudem zählt er seine erfolgreich gemeldeten Terminierungsmeldungen und notiert dies in seinem log.
   - Die Abstimmung arbeitet wie folgt:

    - Ein ggT-Prozess erhält die Anfrage nach der Terminierung (**vote**) und er ist nicht der Initiator: ist seit dem letzten Empfang einer Zahl mehr als **<ttt>**/2 Sekunden vergangen, dann leitet er die Anfrage an seinen linken Nachbarn weiter (implizites Zustimmen). Sonst ignoriert er die Nachricht (implizites ablehnen).
    - Erhält ein initiierender Prozess von seinem rechten Nachbarn die Anfrage nach der Terminierung (**vote**), meldet er die Terminierung dem Koordinator.
6. Der **ggT-Prozess ist in Erlang/OTP** zu implementieren und muss auf jedem Rechner im Labor startbar sein!

GUI

25. **Koordinator**-GUI:
    a. Eine einfache Textausgabe reicht.
    b. Alle ausgegebenen Informationen werden in der Datei Koordinator_Koordinatornode@Rechner.log mit protokolliert. Ein Beispiel liegt der Aufgabe bei.
    c. Die Ausgabe soll es ermöglichen, den Berechnungsverlauf nachzuvollziehen!
26. **Starter**-GUI:
    a. Eine einfache Textausgabe.
    b. Alle ausgegebenen Informationen werden in der Datei starter<nr>_Starternode@Rechner.log mit protokolliert. Ein Beispiel liegt der Aufgabe bei.
    c. Die Ausgabe soll es ermöglichen, den Berechnungsverlauf nachzuvollziehen!
27. **ggT-Prozess**-GUI:
    a. Eine einfache Textausgabe reicht.
    b. Alle ausgegebenen Informationen werden in der Datei GGTP_Name_GGTPNode@Rechner.log mit protokolliert. Ein Beispiel liegt der Aufgabe bei.
    c. Die Ausgabe soll es ermöglichen, den Berechnungsverlauf nachzuvollziehen!

Hinweise

- Die Vorgaben sind so gestaltet, dass bestimmte, lehrreiche Effekte während einer Berechnung entstehen können. Bevor Sie hier versuchen, "Fehler" zu vermeiden, fragen Sie bitte kurz nach, ob es sich um einen "Fehler" handelt. Sie würden sonst gegen die Vorgabe implementieren!
- Das System ist möglichst informativ zu gestalten.
- Für eine gemeinsame Vorführung sind die Vorgaben genau einzuhalten!
- Veränderungen am Kernalgorithmus (Berechnung des ggT) sind nicht zulässig!

Schnittstellen

Bei allen Schnittstellen sind bei synchroner Kommunikation die Antwortnachrichten bei dem Auftraggeber nicht aufgeführt, so sind z.B. die Antwortnachrichten des Namensdienstes nur beim Namensdienst aufgeführt!

Für den vorgegebenen **Namensdienst** nameservice:

- Nachdem ein ping auf den Knoten des Namensdienstes gemacht wurde (net_adm:ping(NameserviceNode)), erhält man die Adresse durch: Nameservice = global:whereis_name(nameservice).
- Rebinden eines Dienstes (erstmaliges oder wiederholtes binden): **Nameservice ! {self(),{rebind,meindienst,node()}}**,
- Lookup für einen Dienst: **Nameservice ! {self(),{lookup,meindienst}}**,
- Unbind eines Dienstes: **Nameservice ! {self(),{unbind,meindienst}}**,

Für den von Ihnen zu implementierenden **Koordinator Prozess**:

- **{get_ggt_vals,PID}**: Die Anfrage nach den steuernden Werten durch den Starter Prozess mit Nummer PID.
- **{check_in,ggtName}**: Ein ggT-Prozess meldet sich beim Koordinator mit Namen an (Name ist der beim Namensdienst registrierte Name).
- **{brief_mi,{ggtName,ggTMi,ggTZeit}}**: Ein ggT-Prozess mit Namen ggtName informiert über sein neues Mi ggTMi um ggTZeit Uhr.
- **{brief_term, {ggtName,ggTMi,ggTZeit},From}**: Ein ggT-Prozess mit Namen ggtName und PID From informiert über die Terminierung mit Ergebnis ggTMi um ggTZeit Uhr.
- **reset**: Der Koordinator sendet allen ggT-Prozessen das kill-Kommando und bringt sich selbst in den Zustand, indem sich Starter wieder melden können.
- **step**: Der Koordinator bildet den Ring. Er wartet nun auf den Start einer ggT-Berechnung.
- **prompt**: Der Koordinator erfragt bei allen ggT-Prozessen per **tell_mi** deren aktuelles Mi ab und zeigt es im log an.
- **whats_on**: Der Koordinator erfragt bei allen ggT-Prozessen per **whats_on** 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.

Für den von Ihnen zu implementierenden **Starter Prozess**:

- **{ggt_vals, TTW, TTT , GGTs}**: die steuernden Werte für die ggT-Prozesse werden im Starter Prozess gesetzt; TTW / TTT siehe Text, GGTs ist die Anzahl der zu startenden ggT-Prozesse.

Für den von Ihnen zu implementierenden **ggT-Prozess**:

- **{set_neighbours,LeftN,RightN}**: die (lokal auf deren Node registrierten und im Namensdienst registrierten) Namen des linken und rechten Nachbarn werden gesetzt. LeftN und RightN sind die Namen der Nachbarn.
- **{set_pmi,MiNeu}**: die von diesem Prozess zu berabeitenden Zahl für eine neue Berechnung wird gesetzt.
- **{send,Y}**: der rekursive Aufruf der ggT Berechnung.
- **{vote,Initiator}**: Wahlnachricht für die Terminierung der aktuellen Berechnung; Initiator ist der Initiator dieser Wahl (z.B. Name des ggT-Prozesses).

- **{tell_mi,From}**: Sendet das aktuelle Mi an From: From ! {mi,Mi}. Wird vom Koordinator z.B. genutzt, um bei einem Berechnungsstillstand die Mi-Situation im Ring anzuzeigen.
- **{whats_on,From}**: Sendet ein **{i_am, State}** an den Koordinator, in dem der aktuelle Zustand mitgeteilt wird. Wird vom Koordinator z.B. genutzt, um auf manuelle Anforderung hin die Lebendigkeit des Rings zu prüfen.
- **kill**: der ggT-Prozess wird beendet.