Prof. Dr. Thomas Schmidt
HAW Hamburg, Dept. Informatik,
Raum 780, Tel.: 42875 - 8452
Email: schmidt@informatik.haw-hamburg.de
Web: http://inet.cpt.haw-hamburg.de/teaching

Verteilte Systeme

Aufgabe 2: Verteilte Primzahlfaktorisierung im Aktor-Modell

Ziele:

1. Message Passing im Aktor-Modell kennenlernen
2. Verteilungsszenarium für ein nebenläufiges Problem *begründet* Entwerfen
3. Konzipiertes Szenarium mittels asynchroner Nachrichten implementieren
4. Erzieltes Ergebnis mittels Performanzmessung evaluieren

Vorbemerkungen:

In dieser Aufgabe betrachten wir ein lose gekoppeltes Problem verteilter Rechenlast, dass im sog. Aktor-Modell (s. u.) gelöst werden soll. Dabei tauschen Worker (Aktoren) Nachrichten zur Koordinierung aus, um gemeinsam im Wettbewerb das gegebene Problem zu lösen.

Ihre Aufgabe ist es zunächst, ein *durchdachtes Konzep*t zu erstellen, in dem die Aufgabenverteilung und die Kommunikationsschritte passend zum Problem gewählt werden. Messen Sie die Qualität Ihrer Lösungsideen an den Qualitätseigenschaften verteilter Systeme und benutzen Sie diese Kriterien in der *Konzeptbegründung*. Diskutieren Sie insbesondere das *Skalierungsverhalten* und die *Fehlertoleranz*. Evaluieren Sie die tatsächliche Leistungsfähigkeit Ihrer Lösung mithilfe einer verteilten Laufzeitmessung.

Problemstellung:

Die Primfaktorenzerlegung großer Zahlen ist eines der numerisch "harten" Probleme. Public Key Security Verfahren (RSA) leiten z.B. ihre Schlüsselsicherheit davon ab, dass eine öffentlich bekannte große Zahl nicht in der notwendigen Zeit in ihre (unbekannten) Primfaktoren zerlegt werden kann. Aus umgekehrter Sicht ist es von Interesse, Rechenverfahren zu entwerfen, mit welchen die Primfaktorisierung möglichst beschleunigt werden kann.

Wie Sie sich leicht überlegen können, hat das naive Ausprobieren aller infrage kommenden Teiler einen Rechenaufwand von $\mathcal{O}(\sqrt{\mathcal{N}})$, wenn $\mathcal{N}$ die zu faktorisierende Zahl ist. Der nachfolgende Algorithmus, welchen wir verteilt implementieren wollen, geht auf Pollard zurück und findet einen Primfaktor p im Mittel nach $1{,}18\sqrt{p}$ Schritten. Seine zugrundeliegende Idee ist die des 'Geburtstagsproblems': Wie Sie mit einfachen Mitteln nachrechnen können, ist die Wahrscheinlichkeit überraschend groß, auf einer Party zufällig eine Person zu treffen, die am gleichen Tag Geburtstag hat wie Sie. [Randbemerkung: Pikanterweise ist gerade das Nichtbeachten dieses Geburtstagsproblems der Grund für die kryptographische Schwäche der WLAN Verschlüsselung WEP.]

Die Pollard Rho Methode zur Faktorisierung:

Die Rho Methode ist ein stochastischer Algorithmus, welcher nach zufälliger Zeit, aber zuverlässig Faktoren einer gegebenen *ungeraden* Zahl $\mathcal{N}$ aufspürt. Hierzu wird zunächst eine Pseudo-Zufallssequenz von Zahlen $x_i \leq \mathcal{N}$ erzeugt:

$$x_{i+1} = x_i^2 + a \bmod \mathcal{N},\ a \neq 0, -2 \text{ beliebig.}$$

Gesucht werden nun die Perioden der Sequenz x_i, also ein Index p, so dass $x_{i+p} = x_i$. p ist dann ein Teiler von $\mathcal{N}$.

Solche Zyklenlängen p lassen sich leicht mithilfe von Floyd's Zyklenfindungsalgorithmus aufspüren:

Berechne $d = (x_{2i} - x_i) \bmod \mathcal{N}$, dann ist

$p = GGT(d, \mathcal{N})$, wobei GGT der größte gemeinsame Teiler ist.

Im Pseudocode sieht der Algorithmus von Pollard wie folgt aus:

rho (N,a) { N = zu faktorisierende Zahl; a = (worker-basiertes) Inkrement der Zufallssequenz;}

```
x = rand(1 … N);
y = x;
p = 1;
Repeat
        x = (x^2 + a) mod N;
        y = (y^2 + a) mod N;
        y = (y^2 + a) mod N;
        d = (y  -  x) mod N;
        p = ggt (d, N);
until (p != 1);
if (p != N) then  factor_found(p)
```

Hinweise: Die Rho-Methode findet nicht nur Primfaktoren, sondern manchmal auch das Produkt von mehreren Primfaktoren - **deshalb muss ein einmal gefundener Faktor noch 'weiterbearbeitet' werden**. Gefundene Faktoren können N zudem auch mehrfach teilen! Wenn die Rho-Methode terminiert, ohne einen echten Faktor gefunden zu haben (p = N), dann ist das untersuchte N entweder unteilbar, oder N wurde als Produkt seiner Primfaktoren entdeckt. Den erstgenannten Fall können Sie über einen Primalitätstest ausschließen, im letztgenannten Fall muss die Faktorisierung mit einer veränderten Zufallssequenz (Startwert und a) erneut durchgeführt werden.
Da es sich um ein zufallsgesteuertes Verfahren mit zufälliger Laufzeit handelt, können zu dem ungewöhnlich hohe Laufzeiten auftreten, ohne dass ein Faktor gefunden wird. Implementieren Sie ggf. eine Abbruchbedingung für die obige Schleife, nach welcher Sie den Algorithmus neu starten.

Das Aktor-Modell

Aktoren sind nebenläufige, unabhängige Softwarekomponenten, die keine gemeinsame Sicht einen Speicherbereich haben. Sie kommunizieren durch asynchronen Nachrichtenaustausch miteinander und können zu ihrer Laufzeit weitere Aktoren erschaffen. Da das Programmiermodell keine gemeinsame Sicht auf einen Speicherbereich vorsieht, werden zum einen Race Conditions ausgeschlossen und zum anderen eignet sich das Aktor-Modell auch zur Programmierung von im Netzwerk verteilter Anwendungen.

Aufgabenstellung

Teilaufgabe 1:

Konzipieren Sie ein Verteilungs- und Kommunikationsszenario im Aktor-Modell, in welchem die Rho-Methode auf nebenläufigen Workern 'im Wettbewerb' abgearbeitet wird (mit unterschiedlichen Inkrementen a).

Hierzu benötigen Sie:

1. Einen Master mit User-Interface für Start und Auswertung, welcher die zu faktorisierende Zahl entgegennimmt, an die Worker (Aktoren) verteilt und das Ergebnis (= die vollständige Primfaktorzerlegung sowie (a) die *tatsächlich aufgewendete CPU-Zeit, (b) die Summe der Rho Zyklendurchläufe* und (c) die *verstrichene Zeit* vom ersten Versenden bis zum Erhalt des letzten Faktors) ausgibt.
2. Kommunizierende Worker (Aktoren), die
    a. auf entfernten Rechnern durch en Master gesteuert werden,
    b. die Pollardmethode auf ihnen übergebene Zahlen anwenden,
    c. selbst gefundene Faktoren zusammen mit der aufgewendeten CPU-Zeit mitteilen
    d. und *asynchron* auf durch andere gefundene Faktoren lauschen.

Legen Sie Ihr Vorgehen *begründet* **in einem kurzen Konzeptpapier dar.**

Teilaufgabe 2:

Implementieren Sie nun Ihre konzipierte Lösung in C++, Skala oder Java mit

- Worker-Aktoren, die die Rho-Methode BigInteger (Java), BigInt (Scala) oder int512_t (C++) Arithmetik realisieren.
- einem Starter, der mit ggf. vorgegebenen Workern kommuniziert und am Ende das Ergebnis gemeinsam mit einer Leistungsstatistik (CPU-Zeiten, verstrichene Wall-Clock Zeiten) ausgibt;
- ggf. weiteren Komponenten aus Ihrem Konzept sowie dem Kommunikationsablauf.

Teilaufgabe 3:

Testen Sie Ihr Programm unter Verteilung auf unterschiedliche Rechner mit den Zahlen:

Z1 = 8806715679 = 3 * 29 * 29 * 71 * 211 * 233

Z2 = 9398726230209357241 = 443 * 503 * 997 * 1511 * 3541 * 7907

Z3 = 113704728156282448422617157521937400432081248304 7

Z4 = 100060210614380659647872297427366695090390611213179474545733865926684244698502207679211230917397524350696971050 3

Analysieren Sie das Laufzeitverhalten Ihres Programmes: CPU-Zeit versus Wall-Clock Zeit, vergleichen Sie mit anderen Lösungen.

Anhang: Hilfestellungen und Erläuterungen

Aktoren in Scala und Java

In der *Scala/Java-Bibliothek Akka*, die sich grob am Aktor-Modell orientiert, sind Aktoren eventbasierte Software-Komponenten, die von der Klasse Actor/UntypedActor erben. Empfängt ein Aktor eine Nachricht, wird der "receive"-Handler des Aktors (in Java die onReceive Methode) aufgerufen. Um netzwerktransparent programmieren zu können, muss Akka entsprechend konfiguriert werden. Aktoren, die über Netzwerk erreichbar sein sollen brauchen zudem einen Namen, über den sie entfernt angesprochen werden können.

Aktoren in C++

Die C++11 Bibliothek *libcppa* implementiert ein Aktoren-System, in dem ein Aktor sowohl Funkions- als auch Klassenbasiert implementiert werden kann. In *libcppa* können Aktoren mithilfe der Funktion *publish* an einen Port gebunden werden. Andere Knoten im verteilten System können diesen Aktor dann mit der Funktion *remote_actor* erreichen.

Programmierbeispiel: Verteilte Aktoren

Der Master vergibt eine Berechnungsaufgabe an einen Worker, den er per Netzwerk erreicht, weiter und empfängt dessen berechnetes Ergebnis. Der Worker wird hierzu *vor* dem Master gestartet. Auf den folgenden Seiten finden Sie das Beispielprogramm für Scala, C++ und Java implementiert. Weitere Erklärungen finden Sie in den Quellcode-Kommentaren.

C++ Beispielprogramm:

```
#include <string>
#include <vector>
#include <iostream>
#include "cppa/cppa.hpp"
#include <boost/multiprecision/cpp_int.hpp>

using namespace std;
using namespace cppa;
using boost::multiprecision::int512_t;

void worker(int state_id) {
  become (
    on(atom("calculate"), arg_match) >> [=](int512_t a, int512_t b) {
      // send a continuation message to self
      send(self, atom("ccont"), state_id + 1, self->last_sender(), a, b);
      // change state to accept continuation
      worker(state_id + 1);
    },
    on(atom("ccont"), arg_match)
    >> [=](int state, actor_ptr server, int512_t a, int512_t b) {
      if (state != state_id) return; // discard old messages
      // calculate and send result
      send(server, atom("result"), a + b);
    }
  );
}
```

```
void master(vector<actor_ptr> worker_nodes) {
  become (
    on(atom("calculate"), arg_match) >> [=](int512_t a, int512_t b) {
      // send calculate message to each worker
      for (auto& w : worker_nodes) {
        send(w, atom("calculate"), a, b);
      }
    },
    on(atom("result"), arg_match) >> [](int512_t r) {
      // print result and quit actor
      cout << "result = " << r << endl;
      self->quit();
    },
    others() >> [] {
      cout << "received: " << to_string(self->last_dequeued()) << endl;
    }
  );
}



int main(int argc, char** argv) {
  // tell libcppa how to serialize an int512_t
  struct meta_int512_t : util::abstract_uniform_type_info<int512_t> {
    void serialize(const void* ptr, serializer* sink) const {
      ostringstream oss;
      oss << *reinterpret_cast<const int512_t*>(ptr);
      *sink << oss.str();
    }
    void deserialize(void* ptr, deserializer* source) const {
      istringstream iss{get<string>(source->read_value(pt_u8string))};
      iss >> *reinterpret_cast<int512_t*>(ptr);
    }
  };
  announce(typeid(int512_t), create_unique<meta_int512_t>());
  // parse arguments
  match (vector<string>(argv + 1, argv + argc)) (
    on("-s") >> [] {
      // connect to a worker running on localhost:4242 and spawn master
      auto m = spawn(master, vector<actor_ptr>{remote_actor("localhost", 4242)});
      send(m, atom("calculate"), int512_t{10}, int512_t{20});
    },
    on("-w") >> [] {
      // spawn worker and publish it to port 4242
      publish(spawn(worker, 0), 4242);
    },
    others() >> [] {
      cout << "usage:" << endl
           << " '-s'    start in server mode" << endl
           << " '-w'    start as worker node" << endl;
    }
  );
  // wait until are actors are done
  await_all_others_done();
}
```

Scala Beispielprogramm:

```
package de.haw.inet.vs

import scala.math.BigInt
import akka.actor._
import com.typesafe.config.ConfigFactory

// message types
case class CalculateMessage(a: BigInt, b: BigInt)
case class ContinuationMessage(stateId: Int, server: ActorRef,
                               a: BigInt, b: BigInt)
case class ResultMessage(result: BigInt)

class Worker extends Actor {
  def bhvr(stateId: Int): Receive = {
    case CalculateMessage(a, b) => {
      // send a continuation message to self
      self ! ContinuationMessage(stateId + 1, sender, a, b)
      // change state to accept continuation
      context.become(bhvr(stateId + 1))
    }
    case ContinuationMessage(sId, server, a, b) => {
      if (sId == stateId) {
        // calculate and send result
        server ! ResultMessage(a + b)
      }
    }
  }
  // set default message handler
  def receive = bhvr(0)
}

class Master(system: ActorSystem) extends Actor {
  def bhvr(nodes: List[String]): Receive = {
    case CalculateMessage(a, b) => {
      // send CalculateMessage to each worker
      nodes foreach (n => {
        val w = system.actorSelection(n);
        w ! CalculateMessage(a, b)
      })
    }
    case ResultMessage(result) => {
      // print result and shutdown the actor system
      println("result = " + result)
      self ! PoisonPill
      system shutdown
    }
  }
  // set default message handler
  def receive =
bhvr(List[String]("akka.tcp://Lab2Node@127.0.0.1:2553/user/worker"))
}
```

```
object Lab2 {

  // Akka configuration for Master
  val lab2Config = """
    akka {
      loglevel = DEBUG
      actor.provider = "akka.remote.RemoteActorRefProvider"
      remote {
        secure-cookie = "0009090D040C030E03070D0509020F050B080400"
        require-cookie = on
             transport = "akka.remote.netty.NettyRemoteTransport"
      }
    }
  """

  // Akka configuration for Worker
  val lab2NodeConfig = """
    akka {
      loglevel = DEBUG
      actor.provider = "akka.remote.RemoteActorRefProvider"
      remote {
        secure-cookie = "0009090D040C030E03070D0509020F050B080400"
        require-cookie = on
        transport = "akka.remote.netty.NettyRemoteTransport"
        netty.tcp {
          hostname = "127.0.0.1"
          port = 2553
        }
      }
    }
  """

  def main(args: Array[String]): Unit = args match {
    case Array("-s") => {
               val system = ActorSystem("Lab2",
                                        ConfigFactory.parseString(lab2Config))
               val master = system.actorOf(Props(classOf[Master], system))
               master ! CalculateMessage(10, 10)
         }
         case Array("-w") => {
               val system = ActorSystem("Lab2Node",
                                        ConfigFactory.parseString(lab2NodeConfig))
               val node = system.actorOf(Props[Worker], "worker")
         }
         case _ => {
           println("usage:\n" +
                    " '-s'   start in server mode\n" +
                    " '-w'   start in worker mode")
         }
       }

}
```

Java Beispielprogramm:

CalculateMessage.java:

```
package de.haw.inet.vs.lab2;

import java.io.Serializable;
import java.math.BigInteger;

public class CalculateMessage implements Serializable {

  private static final long serialVersionUID = 840244832287440949L;

  private BigInteger a;

  private BigInteger b;

  public BigInteger getA() {
    return a;
  }

  public BigInteger getB() {
    return b;
  }

  public CalculateMessage(BigInteger a, BigInteger b) {
    this.b = b;
    this.a = a;
  }

}
```

ContinuationMessage.java:

```
package de.haw.inet.vs.lab2;

import java.io.Serializable;
import akka.actor.ActorRef;

public class ContinuationMessage implements Serializable {

  private static final long serialVersionUID = 1L;

  private int stateId;

  private CalculateMessage calculateMessage;

  private ActorRef server;

  public int getStateId() {
    return stateId;
  }

  public CalculateMessage getCalculateMessage() {
    return calculateMessage;
  }

  public ContinuationMessage(ActorRef server, int stateId, CalculateMessage calcu-
lateMessage) {
    this.server = server;
    this.stateId = stateId;
    this.calculateMessage = calculateMessage;
  }

  public ActorRef getServer() {
    return server;
  }
}
```

ResultMessage.java:

```
package de.haw.inet.vs.lab2;

import java.io.Serializable;
import java.math.BigInteger;

public class ResultMessage implements Serializable {

  private static final long serialVersionUID = -6065578273626197783L;

  private BigInteger result;

  public BigInteger getResult(){
    return this.result;
  }

  public ResultMessage(BigInteger result){
    this.result = result;
  }
  }
```

Worker.java:

```
package de.haw.inet.vs.lab2;

import java.math.BigInteger;
import akka.actor.UntypedActor;

public class Worker extends UntypedActor {

  private int stateId = 0;

  public void onReceive(Object message) {
    if (message instanceof CalculateMessage) {
      CalculateMessage calculateMessage = (CalculateMessage) message;
      // send a continuation message to self
      self().tell(new ContinuationMessage(sender(), ++stateId, calculateMessage),
self());
    } else if (message instanceof ContinuationMessage) {
      ContinuationMessage continuationMessage = (ContinuationMessage) message;
      // discard old message
      if (continuationMessage.getStateId() != stateId) return;
      // calculate and send result
      BigInteger a = continuationMessage.getCalculateMessage().getA();
      BigInteger b = continuationMessage.getCalculateMessage().getB();
      BigInteger result = a.add(b);
      continuationMessage.getServer().tell(new ResultMessage(result), self());
    } else {
      throw new IllegalArgumentException("Unknown message [" + message + "]");
    }
  }

}
```

Master.java:

```
package de.haw.inet.vs.lab2;

import java.util.ArrayList;
import java.util.List;
import akka.actor.*;

public class Master extends UntypedActor {

  @Override
  public void onReceive(Object message) throws Exception {
    if (message instanceof CalculateMessage) {
      CalculateMessage calculate = (CalculateMessage) message;
      // send  CalculateMessage to each worker
      for (String path : workerNodes) {
        ActorSelection worker = system.actorSelection(path);
        worker.tell(calculate, self());
      }
    } else if (message instanceof ResultMessage) {
      // print result and shutdown the actor system
      self().tell(PoisonPill.getInstance(), self());
      system.shutdown();
    } else {
      throw new IllegalArgumentException("Unknown message [" + message + "]");
    }
  }

  private List<String> workerNodes = new ArrayList<String>() {{
              add("akka.tcp://Lab2Node@127.0.0.1:2553/user/worker");
          }};

  private ActorSystem system;

  public Master(ActorSystem system) {
    this.system = system;
  }

}
```

Main.java:

```
package de.haw.inet.vs.lab2;

import akka.actor.*;
import java.math.BigInteger;
import com.typesafe.config.ConfigFactory;

public class Main {

  // Akka configuration for Master
  public static final String lab2Config =
      "akka {" +
      "  loglevel = DEBUG\n" +
      "  actor.provider = \"akka.remote.RemoteActorRefProvider\"\n" +
      "  remote {\n" +
      "    secure-cookie = \"0009090D040C030E03070D0509020F050B080400\"\n" +
      "    require-cookie = on\n" +
      "    transport = \"akka.remote.netty.NettyRemoteTransport\"\n" +
      "  }\n" +
      "}";

  // Akka configuration for Worker
  private static final String lab2NodeConfig = "akka {\n" +
          "  loglevel = DEBUG\n" +
          "  actor.provider = \"akka.remote.RemoteActorRefProvider\"\n" +
          "  remote {\n" +
          "    secure-cookie = \"0009090D040C030E03070D0509020F050B080400\"\n" +
          "    require-cookie = on\n" +
          "    transport = \"akka.remote.netty.NettyRemoteTransport\"\n" +
          "    netty.tcp {\n" +
          "      hostname = \"127.0.0.1\"\n" +
          "      port = 2553\n" +
          "    }\n" +
          "  }\n" +
          "}";


  public static void main(String[] args) {
     if (args.length == 1 && args[0].equals("-s")) {
      ActorSystem system = ActorSystem.create("Lab2", ConfigFacto-
ry.parseString(lab2Config));
      ActorRef master = system.actorOf(Props.create(Master.class, system));
      master.tell(new CalculateMessage(new BigInteger("10"), new BigInte-
ger("10")), null);
    }
    else if (args.length == 1 && args[0].equals("-w")) {
      ActorSystem system = ActorSystem.create("Lab2Node", ConfigFacto-
ry.parseString(lab2NodeConfig));
      system.actorOf(Props.create(Worker.class), "worker");
    }
    else {
      System.out.println("usage:\n" +
                  " '-s'   start in server mode\n" +
                  " '-w'   start in worker mode");
    }
  }
}
```

Um das Scala oder Java Beispiel kompilieren und ausführen zu können, benötigen Sie die Akka-Library in der Version 2.2.1 (http://downloads.typesafe.com/akka/akka-2.2.1.zip). Legen Sie ein Eclispe-Projekt an, in dem Sie die Jars aus dem "lib/akka" Verzeichnis hinzufügen. Bei einem Java-Projekt müssen Sie zudem noch die scala-library.jar einbinden.

Um das C++ Beispiel kompilieren zu können benötigen Sie libcppa in der Version 0.8.1 (https://github.com/Neverlord/libcppa/releases/tag/V0.8.1), die Boost-Library und einen C++11-fähigen Compiler (z.B. GCC mindestens in Version 4.7 oder Clang). Beim kompilieren müssen Sie das Flag "-std=c++11" benutzen, bei Clang zusätzlich "-stdlib=libc++".

In allen Fällen starten Sie das Programm mit dem Kommandozeilenargument "-w" um einen Worker zu starten und "-s" um einen Master zu starten. **Wichtiger Hinweis** zu Akka: in der Konfiguration für den Worker muss der Konfigurationsparameter "hostname" auf die tatsächliche IP geändert werden, damit eine Verbindung über Netzwerk hergestellt werden kann. Die Konfiguration lässt sich in eine Datei auslagern, um ohne Neukompilierung die Parameter ändern zu können.