Distributed Programming

Marcel Heinz & Ralf Lämmel
Software Languages Team
University of Koblenz-Landau

© 2016 Software Languages Team
Motivation

• How can we achieve better performance?
• How can we distribute computations?
• How can we realize communication between computation processes?
Relevant Programming Concepts

- Distributed Programming
- Concurrent Programming
- Parallel Programming
Relevant Programming Concepts

- Distributed Programming
- Concurrent Programming
- Parallel Programming

© 2016 Software Languages Team
Multi-Core Processing

- Moore's Law: Transistor density doubles approximately every two years and the number of cores correlates with it. [https://en.wikipedia.org/wiki/Moore's_law].
- How to advantage of this processing power?
Parallel computing is a form of computation in which many calculations are carried out simultaneously,

[https://en.wikipedia.org/wiki/Parallel_computing]
Parallel Programming

• There are several (possibly virtual) processors
• Many forms of parallelism
  – Task parallelism -- distribute execution processes
  – Data parallelism -- distribute data across multiple computing nodes
  – etc.
Multithreading in Java

- Thread
- Runnable
- Callable
- Future
- Thread Pool
Java Threads

• ... are objects

• ... model threads of execution

• ... can be started, interrupted, sent to sleep and made wait for notification, etc.

• Can be assigned an activity by passing a 'Runnable' implementation to Thread's constructor

• Behavior can also be added by creating a subclass and overwriting run(), where run initially does not implement an activity.
Thread implementation

java::lang::Thread
+ run() : void

java::lang::Runnable
+ run() : void

MyActivity
+ run() : void

Be careful about this choice!

Best practice to implement an activity that is supposed to be executed by a thread.

© 2016 Software Languages Team
Runnable

package concurrency;
public class PrintLengths implements Runnable {
    private final String s;
    public PrintLengths(String s) {
        this.s = s;
    }
    @Override
    public void run() {
        System.out.println(s.length());
    }
}

public static void main(String[] args) {
    String[] words = {"let","us","print"};
    for(String w : words){
        Thread t = new Thread(new PrintLengths(w));
        t.start();
    }
}
package concurrency;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class PrintLengthsTP implements Runnable{
    private final String s;
    public PrintLengthsTP(String s) {
        this.s = s;
    }
    @Override
    public void run() {
        System.out.println(s.length());
    }

    public static void main(String[] args) {
        String[] words = {"let","us","print"};
        ExecutorService p = Executors.newFixedThreadPool(2);
        for(String w : words){
            p.submit(new PrintLengthsTP(w));
        }
        p.shutdown();
    }
}
Threadpools

- **Single Thread Executor**: Uses a single thread
- **Cached Thread Pool**: Creates as many threads as necessary for a task. Old threads will be reused and removed if they idle.
- **Fixed Thread Pool**: Fixed number of threads
- **Scheduled Thread Pool**: Task scheduling capabilities
- **Single Thread Scheduled Pool**: One thread and scheduling capabilities
public class SumLength implements Callable<Integer>{
    private final String s;
    public SumLength(String s) {
        @Override
        public Integer call(){
            return s.length();
        }
    }
    public static void main(String[] args) throws InterruptedException, ExecutionException {
        String[] words = {"let","us","print"};
        ExecutorService p = Executors.newFixedThreadPool(2);
        Set<Future<Integer>> fs = new HashSet<>();
        for(String w : words)
            fs.add(p.submit(new SumLength(w)));
        int sum = 0;
        for(Future<Integer> f : fs){
            sum+= f.get();
        }
        System.out.println(sum);
        p.shutdown();
    }
}
Problem Description

• Is executing a repetitive task in parallel enough to boost up performance?

• What if there are multiple distinct tasks that have to be performed in parallel?

• How to support parallel execution on an architectural level?

• How to protect critical areas?
Relevant Programming Concepts

- Distributed Programming
- Concurrent Programming
- Parallel Programming
Concurrency

- According to concepts such as „Divide and conquer" a problem can be split into smaller parts.
- Dependent parts: Consider parts that have to be executed in an order.
- Independent parts: Parts that do not depend on another part to be executed first.
Concurrency

• Concurrency provides a way to structure a solution to a problem that may be parallelizable.

• Concurrency is about dealing with lots of things at once, while parallelism focuses on doing lots of things at once.

• Concurrent programming deals with some well-defined interaction (communication) between the independent parts.
Parallel vs Concurrent

- "Concurrency should not be confused with parallelism. Concurrency is a language concept and parallelism is a hardware concept."

- "Concurrency and parallelism are orthogonal: it is possible to run concurrent programs on a single processor (using preemptive scheduling and time slices) [...]"

Peter Van Roy: "Programming Paradigms for Dummies: What Every Programmer Should Know"
Three Levels of Concurrency

- **Distributed System**: Computing nodes connected through a network.

- **Operating System**: Managing one computing node. One concurrent activity is called a process and has independent memory.

- **Activities inside one process**: Threads are concurrent activities that execute independently but share the same memory space.
Problem Description

• How to protect critical areas?
The banking problem

https://github.com/101companies/101repo/blob/master/technologies/Java_platform/samples/javaThreadsSamples/banking/
Concurrent Datastructures

- There are a few datastructures that already support concurrency natively
  - ConcurrentHashMap
  - ConcurrentLinkedQueue
  - And more
    https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html
Problem Description

• How can activities tell each other that they finished or share information in a style that resembles the Observer pattern?

• How can processes communicate over a network?
Relevant Programming Concepts

- Distributed Programming
- Concurrent Programming
- Parallel Programming
Messaging-based Concurrency
Messaging-based Concurrency

- Message delivery is preferred instead of data sharing.
- Tony Hoare formulated a formal language for describing patterns of interaction in concurrent systems called `Communicating Sequential Processes (CSP)'.
- Languages such as Occam or Go were influenced by CSP.
Messaging Service

- Message-Oriented-Middleware
- Create, edit, read and send messages
- Send messages to destinations
- Publish messages to all subscribers
- An application server provides resources to support messaging capabilities
- Clients may not have any knowledge of each other's existence

© 2016 Software Languages Team
Publish-Subscribe Messaging

- Multiple activities need to receive the same messages
Point-To-Point Messaging

• An activity needs to send a message to a specific other activity
Java Message Service (JMS) API

- Supports Publish-Subscribe, Point-To-Point messaging, asynchronous- and synchronous communication.
- Messages contain additional information such as a timestamp or user assigned properties.
Java Message Service (JMS) API

- A JMS implementation is ActiveMQ from Apache.
- The approach offers high robustness and guaranteed delivery.
- JMS is used frequently in JavaEE Applications.
The JMS API Programming Model

:Connection Factory
  \( \downarrow \) creates
  :Connection
  \( \downarrow \) creates
  :Session
  \( \downarrow \) creates
  :Message Producer
  \( \downarrow \) sends to
  Destination
  \( \leftarrow \) creates
  :Message
  \( \downarrow \) creates
  :Message Consumer
  \( \downarrow \) receives from
  Destination

© 2016 Software Languages Team
101Companies
A JMS based implementation using Point-To-Point Messaging
101Companies
A JMS based implementation using Publish-Subscribe Messaging
Akka

- Uses the actor model to define independent activities.
- Communication between actors is realized via definable messages.
- Supports synchronous communication at a local level.
- Supports Point-To-Point Messaging through so-called Mailboxes.
- Supports Publish-Subscribe Messaging through Routing.
Actor Model

- Excessively uses `divide and conquer'
- An actor can be envisioned as a human worker.
- Communication based on messages.
- Messages can be placed in an actor's mailbox.
- A hierarchy of supervision has to be set up.
- If an actor does not know how to handle a certain situation, it might send a message to a supervising actor.
Actor Model Guidelines

- A manager supervises its workers that are assigned to a subtask by it.
- If an actor has `critical' data, it should assign subtasks to children to enable appropriate recovery from a failure.
- One actor may simply watch out for another's liveness, if it depends on its work.
- For more: See the documentation http://doc.akka.io/docs/akka/2.4.7/general/actor-systems.html
101Companies
An Akka based implementation using Publish-Subscribe Messaging
Akka – beyond local communication

- Akka Cluster, a fault-tolerant decentralized peer-to-peer based cluster membership service.
- Based on gossip protocols (randomly communicating the cluster's state).
- Cluster membership used in Akka is based on Amazon's Dynamo system.
- One node is defined by a (hostname:port:uid) tuple and is part of a cluster, where a single node acts as a `team-leader`.

© 2016 Software Languages Team
Hadoop

- Allows for distributed processing of large datasets across computing node clusters
- Scale from single node to thousands
- See documentation: http://hadoop.apache.org/
- See 101companies implementation using Hadoop: https://github.com/101companies/101repo/tree/master/contributions/hadoop
Summary

- Concurrency is about dealing and parallelism is about doing many things at once.

- Messaging is a concept to realize communication between activities. Concrete implementations are JMS and Akka.

- Akka and Hadoop enable the creation of large computing node clusters to deal with huge processing loads.