Distributed Pipe-and-Filter Architectures with TeeTime

Master’s thesis
Florian Echternkamp – 21.04.2017
Motivation: Performance

Single node: Parallelization limited by CPU cores
Motivation: Performance

Multiple nodes: parallelization no more limited by CPU
Motivation: Locality

• Data locality
  – Big Data
  – Stages process data close to the corresponding data source
Outline

- Motivation
- Goals
- Foundations
- Developed Approach
- Evaluation
- Related Work
- Conclusion
- Future Work
• Motivation
• **Goals**
• Foundations
• Developed Approach
• Evaluation
• Related Work
• Conclusion
• Future Work
Goals

• G1: Evaluation of Java Frameworks for Distributed Application Development
  – G1.1: Providing Efficient Distributed Communication
  – G1.2: Providing Fault Tolerance
  – G1.3: Providing Remote Deployment and Execution
  – G1.3: Providing Encrypted Data Transmission
• G2: Implementation of a Distributed Pipe-and-Filter Architecture
• G3: Adding Support for Distributed Configurations in the TeeTime DSL
• G4: Evaluation of Our Approach
  – G4.1: Feasibility
  – G4.2: Performance
• Motivation
• Goals
• **Foundations**
• Developed Approach
• Evaluation
• Related Work
• Conclusion
• Future Work
• The Pipe-and-Filter Architectural Style [Sommerville 2012]

• The Pipe & Filter Framework TeeTime [Wulf et al. 2014]

• The Actor Model [Agha 1985]

• The Actor Framework Akka [http://akka.io]
• TeeTime Domain-specific Language (DSL) [Zloch 2016]

• Communication Patterns for Distributed Systems
  [Silcock and Goscinski 1995]
  – Message Passing
  – Remote Procedure Call
  – Distributed Shared Memory
• Motivation
• Goals
• Foundations
• **Developed Approach**
• Evaluation
• Related Work
• Conclusion
• Future Work
Evaluation of Java Frameworks

• Build Infrastructure

<table>
<thead>
<tr>
<th>License</th>
<th>Open Source</th>
<th>Min. JDK Version</th>
<th>Latest activity</th>
<th>Latest release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 2.0</td>
<td>Yes</td>
<td>1.8</td>
<td>12.06.2016</td>
<td>03.03.2016 (2.3.0)</td>
</tr>
</tbody>
</table>

• Features

<table>
<thead>
<tr>
<th>Communication Pattern</th>
<th>Transport Protocol</th>
<th>Fault Tolerance</th>
<th>Remote Deployment</th>
<th>Custom Serializer</th>
<th>Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Passing</td>
<td>TCP, UDP</td>
<td>Supervisor, ...</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Evaluation of Java Frameworks
Evaluation of Java Frameworks

- Apache Zookeeper
- Apache River
- JGroups
- Copycat
- Orbit
- Akka
- Atomix
- Hystrix
- Quasar
- Apache Spark
- Apache Hadoop
- JP

- Communication via stages
- Communication via ports without a pipe
- Communication via a pipe
Distributed Pipe

Distributed Configuration

Node A

Stage

Node B

Stage

Distributed Pipe

Node A

Stage

Distributed Pipe

Sender Actor

Node B

Receiver Actor

Distributed Pipe

Stage
public class DistributedConfiguration extends AbstractDistributedConfiguration {
    public DistributedConfiguration()
    {
        RandomStringGeneratorStage stage1 = new RandomStringGeneratorStage(...);
        CPULoadGeneratorStage stage2 = new CPULoadGeneratorStage(...);
        CPULoadGeneratorStage stage3 = new CPULoadGeneratorStage(...);
        CPULoadGeneratorStage stage4 = new CPULoadGeneratorStage(...);
        EndStage stage5 = new EndStage();
        createNodeForStages("Node1", stage1, stage2);
        createNodeForStages(stage3);
        createNodeForStages(stage4, stage5);
        connectPorts(stage1.getOutputPort(), stage2.getInputPort());
        connectPorts(stage2.getOutputPort(), stage3.getInputPort(), TransportProtocol.TCP);
        connectPorts(stage3.getOutputPort(), stage4.getInputPort());
        connectPorts(stage4.getOutputPort(), stage5.getInputPort());
    }
}
DSL for the Distributed Config

```java
DistributedConfig(){
    RandomStringGeneratorStage stage1(...)
    CPULoadGeneratorStage stage2(...)
    CPULoadGeneratorStage stage3(...)
    CPULoadGeneratorStage stage4(...)
    EndStage stage5()

    Node(Node1) stage1 stage2
    Node stage3
    Node stage4 stage5

    stage1->stage2-TCP->stage3->stage4->stage5
}
```
Remote Deployment (1 of 6)

Diagram of a distributed system with a client, a Java application, a distributed configuration, a master actor, and worker actors on various nodes.
Remote Deployment (3 of 6)

Client

Java Application

Distributed Configuration

Master Actor

Node 1

Worker Actor

TeeTime Execution

Sender Actor

Node ...

Worker Actor

TeeTime Execution

Receiver Actor

Sender Actor

Node n

Worker Actor

TeeTime Execution

Receiver Actor

Request Receiver Path

Registers to Mediator

Mediator
Remote Deployment (4 of 6)

Diagram showing a distributed system architecture with a client, Java application, distributed configuration, master actor, worker ready nodes 1, ..., n, and associated actors (TeeTime Execution, Sender Actor, Receiver Actor).
• Motivation
• Goals
• Foundations
• Developed Approach
• Evaluation
• Related Work
• Conclusion
• Future Work
Evaluation of Our Approach

• Feasibility
  – Remote deployment and remote execution
  – Distributed communication
  – Fault tolerance

• Performance
  – Communication overhead
  – Execution time
Feasibility Test Scenario

Resulting P&F architecture
Demo time
Test setup
- 3 Cloud Nodes
  - 2x Intel Xeon (2.8 GHz, 8 cores)
  - 128 GB RAM
- Software
  - Ubuntu 14.04.5 LTS
  - Java JDK 1.8.121 64bit
- 10GBit/s network
Comm. Overhead Results

Communication Overhead per Transport Protocol

Test objects: 5000x 1 Megabyte large Strings | 20 test iterations
CI 95% <= ±00:00,840
• Configuration Changes
  – Multiple CPU Load Generator Stages in a row
  – Synched pipes
  – Increased CPU load by the factor 10
### Execution Time Results

#### Execution Time per Transport Protocol

<table>
<thead>
<tr>
<th>Protocol</th>
<th>non-distributed</th>
<th>tcp</th>
<th>udp</th>
<th>ssl</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>02:39,480</td>
<td>02:39,462</td>
<td>02:40,609</td>
<td>02:40,752</td>
</tr>
<tr>
<td>distributed</td>
<td>01:35,943</td>
<td>01:35,948</td>
<td>01:36,307</td>
<td>01:36,307</td>
</tr>
</tbody>
</table>

Test objects: 1000x 1 Megabyte large Strings | 20 test iterations
Local CI 95% <= ±00:01,386 | Distributed CI 95% <= ±00:00,250
• Motivation
• Goals
• Foundations
• Developed Approach
• Evaluation

• Related Work
• Conclusion
• Future Work
Related Work

- Apache Hadoop¹: MapReduce on a big problem

  ![Apache Hadoop Logo]

- Apache Spark² and Storm³: only acyclic graphs

  ![Apache Spark Logo]  ![Apache Storm Logo]

- Akka⁴: message box is untyped

  ![Akka Logo]

• Motivation
• Goals
• Foundations
• Developed Approach
• Evaluation
• Related Work
• Conclusion
• Future Work
• Achieved all implementation goals
  – Distributed communication
    • TCP and UDP
    • Encrypted
  – Remote deployment and remote execution
  – Fault tolerance
  – Distributed configurations in the TeeTime DSL
• Feasibility and performance advantages shown
• Motivation
• Goals
• Foundations
• Developed Approach
• Evaluation
• Related Work
• Conclusion
• Future Work
Future Work

- Access stage attributes via the client
- SSL connection between master and worker
- Providing a master system similar to the remote system
  - No need to embed the execution of the distributed configuration
  - Similar to the non-distributed config
References