Evaluating the Prediction Accuracy of Generated Performance Models in Up- and Downscaling Scenarios

Symposium on Software Performance (SOSP) 2014

Andreas Brunnert¹, Stefan Neubig¹, Helmut Krcmar²
¹fortiss GmbH, ²Technische Universität München
Agenda

• Motivation and Vision
• Performance Model Generation
  – Data Collection
  – Data Aggregation
  – Model Generation
• Evaluation
  – SPECjEnterprise2010
  – Overhead Evaluation
  – Experiment Setup
  – Scenario Description
  – Scenario Results
    • Upscaling
    • Downscaling
• Future Work
Agenda

- Motivation and Vision
  - Performance Model Generation
    - Data Collection
    - Data Aggregation
    - Model Generation
  - Evaluation
    - SPECjEnterprise2010
    - Overhead Evaluation
    - Experiment Setup
    - Scenario Description
    - Scenario Results
      - Upscaling
      - Downscaling
- Future Work
Motivation & Vision

• Numerous performance modeling approaches are available to evaluate the performance (i.e., response time, throughput, resource utilization) of enterprise applications

• Performance models are especially useful for scenarios that cannot be tested on real systems, e.g.:
  – Scaling a system up or down in terms of the available hardware resources (e.g., number of CPU cores) during the capacity planning process.

• Creating a performance model requires considerable manual effort
  – → low adoption rates of performance models in practice
Motivation & Vision

• To increase the adoption rates of performance models in practice!
  – …For that purpose, we have proposed an automatic performance model generation approach for Java Enterprise Edition (EE) applications.
  – …This work improves the existing approach by further reducing the effort and time for the model generation.
  – … This work evaluates the prediction accuracy of these generated performance models in up- and downscaling scenarios, i.e.:
    • Increased and reduced number of CPU cores
    • Increased and reduced number of concurrent users
Agenda

- Motivation and Vision
- Performance Model Generation
  - Data Collection
  - Data Aggregation
  - Model Generation
- Evaluation
  - SPECjEnterprise2010
  - Overhead Evaluation
  - Experiment Setup
  - Scenario Description
  - Scenario Results
    - Upscaling
    - Downscaling
- Future Work
Performance Model Generation

Overview

1. Data Collection
2. Data Aggregation
3. Model Generation

(adapted from Willnecker et al. (2014))
Performance Model Generation

Data Collection

Java EE Application

Web Tier
- Servlet Filters
- Web Components (Servlets/JSPs)

Business Tier
- EJB Interceptors
- Enterprise JavaBeans (EJBs)
- JDBC Wrappers

Enterprise Information Systems Tier

Data collected:
- EJB and Web components
- EJB and Web component operations
- EJB and Web component relationships on the level of single component operations
- Resource demands for single component operations (CPU, Memory)
Performance Model Generation

Data Aggregation

BranchMetrics
- invocationCount : long(idl)
- totalCPUDemand : long(idl)
- totalResponseTime : long(idl)
- totalAllocatedHeapBytes : long(idl)

OperationCallLoopCount
- loopCount : long(idl)
- loopCountOccurrences : long(idl)

BranchDescriptor

ParentOperationBranch

JavaEEComponentOperationMBean

OperationIdentifier
- type : string(idl)
- componentName : string(idl)
- operationName : string(idl)

(Brunnert et al. 2014_1)
Performance Model Generation

Model Generation

• Except for the usage model, default models for all model layers of the Palladio Component Model (PCM) are generated automatically:

  – Repository model containing the components of an EA, their relationships and resource demands

  – System model containing the deployment units detected during the data collection (no single components)

  – A simple resource environment with one server and an allocation model that maps all deployment units to this server

(adapted from a presentation for Brunnert et al. (2014_2)
Agenda

• Motivation and Vision
• Performance Model Generation
  – Data Collection
  – Data Aggregation
  – Model Generation
• Evaluation
  – SPECjEnterprise2010
  – Overhead Evaluation
  – Experiment Setup
  – Scenario Description
  – Scenario Results
    • Upscaling
    • Downscaling
• Future Work
Evaluation

SPECjEnterprise 2010

• Benchmark Driver / Emulator
  – Simulates interactions with the SUT
  – Defines workload

• Automobile Manufacturer
  – Orders Domain (CRM)
  – Manufacturing Domain
  – Supplier Domain (SCM)

• Database Server

SPECjEnterprise 2010 Architecture (SPEC, 2009)
Evaluation

SPECjEnterprise2010

(Standard Performance Evaluation Corporation 2009)

• Workload defined by driver
• Business Transactions
  – Browse, Manage, Purchase
  – Predefined sequence of HTTP requests including probabilities

SPECjEnterprise 2010 Architecture
(SPEC, 2009)

Orders Domain Architecture
(Brunnert et al. 2013)
Evaluation

Overhead Evaluation 1/2

• Monitoring code is called before **and** after each monitored invocation
  → Considerable instrumentation overhead!

• Overhead Evaluation: CPU & Heap vs. CPU only
  – 4 CPU cores
  – 20 GB RAM
  – 600 Users
  – Only steady state data is collected
Evaluation

Overhead Evaluation 2/2

<table>
<thead>
<tr>
<th>Component Operation</th>
<th>CPU &amp; Heap</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>CPU only</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All levels</td>
<td>Top level</td>
<td>All levels</td>
<td>Top level</td>
<td>All levels</td>
<td>Top level</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPU</td>
<td>Heap</td>
<td>CPU</td>
<td>Heap</td>
<td>CPU</td>
<td>CPU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 app.sellinventory</td>
<td>1.023 ms</td>
<td>33,650 B</td>
<td>3.001 ms</td>
<td>225,390 B</td>
<td>0.756 ms</td>
<td>3.003 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 CustomerSession.sellInventory</td>
<td>0.785 ms</td>
<td>60,450 B</td>
<td>-</td>
<td>-</td>
<td>0.731 ms</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 CustomerSession.getInventories</td>
<td>0.594 ms</td>
<td>49,540 B</td>
<td>-</td>
<td>-</td>
<td>0.548 ms</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 OrderSession.getOpenOrders</td>
<td>0.954 ms</td>
<td>70,600 B</td>
<td>-</td>
<td>-</td>
<td>0.878 ms</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 dealerinventory.jsp.sellinventory</td>
<td>0.108 ms</td>
<td>16,660 B</td>
<td>-</td>
<td>-</td>
<td>0.103 ms</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total Resource Demand</strong></td>
<td>3.464 ms</td>
<td>230,900 B</td>
<td>3.001 ms</td>
<td>225,390 B</td>
<td>3.015 ms</td>
<td>3.003 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mean Data Collection Overhead</strong></td>
<td>0.116 ms</td>
<td>1378 B</td>
<td></td>
<td></td>
<td>0.003 ms</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Brunnert et al. 2014_1)
## Evaluation

### Overhead Evaluation 2/2

<table>
<thead>
<tr>
<th>Component Operation</th>
<th>CPU &amp; Heap</th>
<th></th>
<th>CPU only</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All levels</td>
<td>Top level</td>
<td>All levels</td>
<td>Top level</td>
</tr>
<tr>
<td></td>
<td>CPU</td>
<td>Heap</td>
<td>CPU</td>
<td>Heap</td>
</tr>
<tr>
<td></td>
<td>Top level</td>
<td></td>
<td>Top level</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Top level</td>
<td></td>
<td>CPU</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 app.sellinventory</td>
<td>1.023 ms</td>
<td>33,650 B</td>
<td>3.001 ms</td>
<td>225,390 B</td>
</tr>
<tr>
<td>2 CustomerSession.sellInventory</td>
<td>0.785 ms</td>
<td>60,450 B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 CustomerSession.getInventories</td>
<td>0.594 ms</td>
<td>49,540 B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 OrderSession.getOpenOrders</td>
<td>0.954 ms</td>
<td>70,600 B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 dealerinventory.jsp.sellinventory</td>
<td>0.108 ms</td>
<td>16,660 B</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total Resource Demand</strong></td>
<td>3.464 ms</td>
<td>230,900 B</td>
<td>3.001 ms</td>
<td>225,390 B</td>
</tr>
<tr>
<td><strong>Mean Data Collection Overhead</strong></td>
<td>0.116 ms</td>
<td>1378 B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Brunnert et al. 2014_1)
Evaluation
Overhead Evaluation 2/2

<table>
<thead>
<tr>
<th>Component Operation</th>
<th>CPU &amp; Heap</th>
<th></th>
<th>CPU only</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU</td>
<td>Heap</td>
<td>CPU</td>
</tr>
<tr>
<td>1 app.sellinventory</td>
<td>1.023 ms</td>
<td>33,650 B</td>
<td>3.001 ms</td>
</tr>
<tr>
<td>2 CustomerSession.sellInventory</td>
<td>0.785 ms</td>
<td>60,450 B</td>
<td>-</td>
</tr>
<tr>
<td>3 CustomerSession.getInventories</td>
<td>0.594 ms</td>
<td>49,540 B</td>
<td>-</td>
</tr>
<tr>
<td>4 OrderSession.getOpenOrders</td>
<td>0.954 ms</td>
<td>70,600 B</td>
<td>-</td>
</tr>
<tr>
<td>5 dealerinventory.jsp.sellinventory</td>
<td>0.108 ms</td>
<td>16,660 B</td>
<td>-</td>
</tr>
<tr>
<td>Total Resource Demand</td>
<td>3.464 ms</td>
<td>230,900 B</td>
<td>3.001 ms</td>
</tr>
<tr>
<td>Mean Data Collection Overhead</td>
<td>0.116 ms</td>
<td>1378 B</td>
<td>0.003 ms</td>
</tr>
</tbody>
</table>

Without heap monitoring, the Mean Data Collection Overhead drops dramatically!

→ We focus on collecting CPU demand only!
Evaluation

Experiment Setup

• Model generation
  – Collect **only** CPU demands using Servlet Filters and EJB Interceptors
  – Only steady state data is collected
  – SUT under moderate load (~ 40-60 %)

• Model simulation
  – Generated PCM models are taken as input for SimuCom
  – Results compared with benchmark runs

• Benchmark runs
  – Servlet Filter at front controller to measure response times and throughput
  – JMX connection to measure CPU utilization of JVM
  – Only steady state data is collected
  – Performed three times (same weight)
Evaluation

Scenario Description

• Upscaling: *How many resources do I need for an increasing workload?*
  – Workload: 600 Users → 900 Users → 1200 Users
  – CPU: 4 Cores → 6 Cores → 8 Cores
  – Model generation: 4 cores (~52 % load)

• Downscaling: *How many resources do I need for the given workload?*
  – Workload: 800 Users
  – CPU: 8 Cores → 6 Cores → 4 Cores
  – Model generation: 8 cores (~39 % load)
## Evaluation

### Scenario Results: Upscaling 1/2

<table>
<thead>
<tr>
<th>CPU Cores</th>
<th>Number of Users</th>
<th>Business Transaction</th>
<th>Measured RT</th>
<th>Simulated RT</th>
<th>Prediction Error</th>
<th>Measured Throughput</th>
<th>Simulated Throughput</th>
<th>Prediction Error</th>
<th>Measured CPU</th>
<th>Simulated CPU</th>
<th>Prediction Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>600</td>
<td>Browse</td>
<td>63.23 ms</td>
<td>65.06 ms</td>
<td>2.91 %</td>
<td>1820.6</td>
<td>1813.1</td>
<td>0.41 %</td>
<td></td>
<td></td>
<td>3.88 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage</td>
<td>11.58 ms</td>
<td>13.28 ms</td>
<td>14.71 %</td>
<td>906.8</td>
<td>917.3</td>
<td>1.16 %</td>
<td>48.76 %</td>
<td>46.87 %</td>
<td>3.88 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Purchase</td>
<td>8.27 ms</td>
<td>9.73 ms</td>
<td>17.67 %</td>
<td>904.9</td>
<td>900.3</td>
<td>0.50 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>900</td>
<td>Browse</td>
<td>69.25 ms</td>
<td>57.56 ms</td>
<td>16.89 %</td>
<td>2708.3</td>
<td>2721.5</td>
<td>0.49 %</td>
<td></td>
<td></td>
<td>9.42 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage</td>
<td>12.54 ms</td>
<td>11.95 ms</td>
<td>4.69 %</td>
<td>1354.3</td>
<td>1354.4</td>
<td>0.01 %</td>
<td>51.72 %</td>
<td>46.85 %</td>
<td>9.42 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Purchase</td>
<td>8.95 ms</td>
<td>8.72 ms</td>
<td>2.60 %</td>
<td>1352.4</td>
<td>1368.1</td>
<td>1.16 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1200</td>
<td>Browse</td>
<td>88.82 ms</td>
<td>56.25 ms</td>
<td>36.66 %</td>
<td>3617.8</td>
<td>3641.9</td>
<td>0.67 %</td>
<td></td>
<td></td>
<td>18.09 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage</td>
<td>14.13 ms</td>
<td>11.64 ms</td>
<td>17.67 %</td>
<td>1806.4</td>
<td>1795.0</td>
<td>0.63 %</td>
<td>57.34 %</td>
<td>46.97 %</td>
<td>18.09 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Purchase</td>
<td>9.31 ms</td>
<td>8.46 ms</td>
<td>9.15 %</td>
<td>1811.6</td>
<td>1819.2</td>
<td>0.42 %</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Brunnert et al. 2014_1)
## Evaluation

### Scenario Results: Upscaling 1/2

<table>
<thead>
<tr>
<th>CPU Cores</th>
<th>Number of Users</th>
<th>Business Transaction</th>
<th>Measured RT</th>
<th>Simulated RT</th>
<th>Prediction Error</th>
<th>Measured Throughput</th>
<th>Simulated Throughput</th>
<th>Prediction Error</th>
<th>Measured CPU</th>
<th>Simulated CPU</th>
<th>Prediction Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>600</td>
<td>Browse</td>
<td>63.23 ms</td>
<td>65.06 ms</td>
<td>2.91 %</td>
<td>1820.6</td>
<td>1813.1</td>
<td>0.41 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage</td>
<td>11.58 ms</td>
<td>13.28 ms</td>
<td>14.71 %</td>
<td>906.8</td>
<td>917.3</td>
<td>1.16 %</td>
<td>48.76 %</td>
<td>46.87 %</td>
<td>3.88 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Purchase</td>
<td>8.27 ms</td>
<td>9.73 ms</td>
<td>17.67 %</td>
<td>904.9</td>
<td>900.3</td>
<td>0.50 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>900</td>
<td>Browse</td>
<td>69.25 ms</td>
<td>57.56 ms</td>
<td>16.89 %</td>
<td>2708.3</td>
<td>2721.5</td>
<td>0.49 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage</td>
<td>12.54 ms</td>
<td>11.95 ms</td>
<td>4.69 %</td>
<td>1354.3</td>
<td>1354.4</td>
<td>0.01 %</td>
<td>51.72 %</td>
<td>46.85 %</td>
<td>9.42 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Purchase</td>
<td>8.95 ms</td>
<td>8.72 ms</td>
<td>2.60 %</td>
<td>1352.4</td>
<td>1368.1</td>
<td>1.16 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1200</td>
<td>Browse</td>
<td>88.82 ms</td>
<td>56.25 ms</td>
<td>36.66 %</td>
<td>3617.8</td>
<td>3641.9</td>
<td>0.67 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage</td>
<td>14.13 ms</td>
<td>11.64 ms</td>
<td>17.67 %</td>
<td>1806.4</td>
<td>1795.0</td>
<td>0.63 %</td>
<td>57.34 %</td>
<td>46.97 %</td>
<td>18.09 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Purchase</td>
<td>9.31 ms</td>
<td>8.46 ms</td>
<td>9.15 %</td>
<td>1811.6</td>
<td>1819.2</td>
<td>0.42 %</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Brunnert et al. 2014_1)
Evaluation
Scenario Results: Upscaling 2/2

4 CPU cores, 600 users

6 CPU cores, 900 users

8 CPU cores, 1200 users

(Brunnert et al. 2014_1)
## Evaluation

### Scenario Results: Downscaling 1/2

<table>
<thead>
<tr>
<th>CPU Cores</th>
<th>Number of Users</th>
<th>Business Transaction</th>
<th>Measured RT</th>
<th>Simulated RT</th>
<th>Prediction Error</th>
<th>Measured Throughput</th>
<th>Simulated Throughput</th>
<th>Prediction Error</th>
<th>Measured CPU</th>
<th>Simulated CPU</th>
<th>Prediction Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>800</td>
<td>Browse</td>
<td>71.54 ms</td>
<td>64.03 ms</td>
<td><strong>10.50 %</strong></td>
<td>2413.9</td>
<td>2415.8</td>
<td><strong>0.08 %</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage</td>
<td>12.96 ms</td>
<td>12.64 ms</td>
<td><strong>2.49 %</strong></td>
<td>1203.5</td>
<td>1209.2</td>
<td><strong>0.48 %</strong></td>
<td>37.41 %</td>
<td>35.17 %</td>
<td><strong>5.99 %</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Purchase</td>
<td>9.36 ms</td>
<td>9.33 ms</td>
<td><strong>0.25 %</strong></td>
<td>1215.9</td>
<td>1228.7</td>
<td><strong>1.05 %</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>800</td>
<td>Browse</td>
<td>67.62 ms</td>
<td>66.03 ms</td>
<td><strong>2.35 %</strong></td>
<td>2413.9</td>
<td>2425.4</td>
<td><strong>0.48 %</strong></td>
<td>46.38 %</td>
<td>46.94 %</td>
<td><strong>1.21 %</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage</td>
<td>12.52 ms</td>
<td>13.08 ms</td>
<td><strong>4.45 %</strong></td>
<td>1202.0</td>
<td>1196.6</td>
<td><strong>0.45 %</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Purchase</td>
<td>9.05 ms</td>
<td>9.64 ms</td>
<td><strong>6.57 %</strong></td>
<td>1208.2</td>
<td>1215.0</td>
<td><strong>0.56 %</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>800</td>
<td>Browse</td>
<td>71.15 ms</td>
<td>87.46 ms</td>
<td><strong>22.92 %</strong></td>
<td>2437.0</td>
<td>2420.8</td>
<td><strong>0.66 %</strong></td>
<td>65.60 %</td>
<td>70.27 %</td>
<td><strong>7.12 %</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage</td>
<td>12.98 ms</td>
<td>17.04 ms</td>
<td><strong>31.29 %</strong></td>
<td>1199.7</td>
<td>1193.5</td>
<td><strong>0.51 %</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Purchase</td>
<td>8.93 ms</td>
<td>12.88 ms</td>
<td><strong>44.33 %</strong></td>
<td>1211.6</td>
<td>1212.1</td>
<td><strong>0.04 %</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Brunnert et al. 2014_1)
## Evaluation

### Scenario Results: Downscaling 1/2

<table>
<thead>
<tr>
<th>CPU Cores</th>
<th>Number of Users</th>
<th>Business Transaction</th>
<th>Measured RT</th>
<th>Simulated RT</th>
<th>Prediction Error</th>
<th>Measured Throughput</th>
<th>Simulated Throughput</th>
<th>Prediction Error</th>
<th>Measured CPU</th>
<th>Simulated CPU</th>
<th>Prediction Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>800</td>
<td>Browse</td>
<td>71.54 ms</td>
<td>64.03 ms</td>
<td>10.50 %</td>
<td>2413.9</td>
<td>2415.8</td>
<td>0.08 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage</td>
<td>12.96 ms</td>
<td>12.64 ms</td>
<td>2.49 %</td>
<td>1203.5</td>
<td>1209.2</td>
<td>0.48 %</td>
<td>37.41 %</td>
<td>35.17 %</td>
<td>5.99 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Purchase</td>
<td>9.36 ms</td>
<td>9.33 ms</td>
<td>0.25 %</td>
<td>1215.9</td>
<td>1228.7</td>
<td>1.05 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>800</td>
<td>Browse</td>
<td>67.62 ms</td>
<td>66.03 ms</td>
<td>2.35 %</td>
<td>2413.9</td>
<td>2425.4</td>
<td>0.48 %</td>
<td>46.38 %</td>
<td>46.94 %</td>
<td>1.21 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage</td>
<td>12.52 ms</td>
<td>13.08 ms</td>
<td>4.45 %</td>
<td>1202.0</td>
<td>1196.6</td>
<td>0.45 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Purchase</td>
<td>9.05 ms</td>
<td>9.64 ms</td>
<td>6.57 %</td>
<td>1208.2</td>
<td>1215.0</td>
<td>0.56 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>800</td>
<td>Browse</td>
<td>71.15 ms</td>
<td>87.46 ms</td>
<td>22.92 %</td>
<td>2437.0</td>
<td>2420.8</td>
<td>0.66 %</td>
<td>65.60 %</td>
<td>70.27 %</td>
<td>7.12 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage</td>
<td>12.98 ms</td>
<td>17.04 ms</td>
<td>31.29 %</td>
<td>1199.7</td>
<td>1193.5</td>
<td>0.51 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Purchase</td>
<td>8.93 ms</td>
<td>12.88 ms</td>
<td>44.33 %</td>
<td>1211.6</td>
<td>1212.1</td>
<td>0.04 %</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Brunnert et al. 2014_1)
Evaluation
Scenario Results: Downscaling 2/2

8 CPU cores, 800 users

6 CPU cores, 800 users

4 CPU cores, 800 users
Agenda

• Motivation and Vision
• Performance Model Generation
  – Data Collection
  – Data Aggregation
  – Model Generation
• Evaluation
  – SPECjEnterprise2010
  – Overhead Evaluation
  – Experiment Setup
  – Scenario Description
  – Scenario Results
  • Upscaling
  • Downscaling
• Future Work
Future Work

• Usage model generation
  – Master Thesis recently finished based on session traces
  – Christian Vögele currently extends this approach

• Representing distributed systems
  – Distributing the transaction ID within distributed Java EE environments using the Application Response Measurement (ARM) standard or similar means
  – Aggregating data from multiple Java EE instances
    • Recently finished a prototype for distributed deployments communicating over SOAP
  – Setting thread limits for each tier (Web, EJB, JDBC Conn. Pools)

• Supporting new component types:
  – JSF, web-services, Message driven beans, Other Java EE 7.0 enhancements, e.g. Batch Jobs, ..
Future Work

• Using APM data for the model generation (collaboration with Dynatrace)
Bibliography


Andreas Brunnert, Stefan Neubig
performancegroup@fortiss.org
pmw.fortiss.org