Performance Benchmarking of Application Monitoring Frameworks

PhD Thesis Defense
Kiel University, Software Engineering Group
Jan Waller — December 12, 2014
Motivation (Monitoring & Overhead)

Measure Everything

At Facebook we collect an enormous amount of [...] application level statistics [...] the really interesting things only show up in production.

—Robert Johnson, “Scaling Facebook to 500 Million Users and Beyond”

Measure influences the performance

Necessary trade-off [Reimer 2013]

• Detailed monitoring
• Monitoring overhead

Manage Overhead

• High overhead is common challenge [Plattner and Nievergelt 1981, Jeffery 1996, Shao et al. 2010]
• Customers expect minimal overhead [Siegl and Bouillet 2011]
• Especially important for frameworks [Bloch 2009, Kanstrén et al. 2011]

Research Questions & Methods

What is the **performance influence** an application-level monitoring framework has on the **monitored system**?

- **What are the causes for observed changes** in the **response time** of a monitored method?
  - *lab experiments*

- **How to develop a benchmark**?
  - *literature review proof-of-concept*

- **How to measure monitoring overhead**?
  - *lab experiments goal, question, metric*

Further reading: Chap. 5 and [Waller 2013]
Outline

Motivation

Monitoring Overhead

Benchmark Engineering Methodology

Benchmarks for Monitoring

Evaluation

Related Work

Outlook
Approach & Contributions
Causes of Monitoring Overhead

<table>
<thead>
<tr>
<th>Method</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>$I$</td>
</tr>
</tbody>
</table>

Further reading: Chap. 6 and [van Hoorn et al. 2009, Waller and Hasselbring 2012, Waller and Hasselbring 2013, Waller et al. 2014]

```java
public boolean method() {
    if (isMonitoringEnabled(...)) {
        r = collectDataBefore();
        writeMonitoringData(r);
    }

    retval = businessMethod();

    if (isMonitoringEnabled(...)) {
        r = collectDataAfter();
        writeMonitoringData(r);
    }

    return retval;
}
```

<table>
<thead>
<tr>
<th>Method &amp; Overhead Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
</tr>
<tr>
<td>$I$</td>
</tr>
<tr>
<td>$C$</td>
</tr>
<tr>
<td>$W$</td>
</tr>
</tbody>
</table>
Method & Overhead Costs

- \( T \) normal execution time
- \( I \) Instrumentation
- \( C \) Collection of data
- \( W \) Writing of data

Goal: \( T \gg I + C + W \)

Further reading: Chap. 6 and [van Hoorn et al. 2009, Waller and Hasselbring 2012, Waller and Hasselbring 2013, Waller et al. 2014]
There is no established methodology for benchmarks

Benchmark Engineering Methodology in three phases:

1. Design / Implementation
2. Execution
3. Analysis / Presentation

including a total of 18 different requirements and guidelines

Further reading: Chap. 7 and [Waller 2013, Waller and Hasselbring 2013, Waller et al. 2014]
### Requirements & Guidelines

<table>
<thead>
<tr>
<th>Requirement</th>
<th>1965 – 2003</th>
<th>2004 – 2014</th>
<th>∑ (49)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: Representative / Relevant</td>
<td>21</td>
<td>21</td>
<td>42</td>
</tr>
<tr>
<td>R2: Repeatable</td>
<td>9</td>
<td>16</td>
<td>25</td>
</tr>
<tr>
<td>R3: Robust</td>
<td>10</td>
<td>18</td>
<td>28</td>
</tr>
<tr>
<td>R4: Fair</td>
<td>4</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>R5: Simple</td>
<td>10</td>
<td>13</td>
<td>23</td>
</tr>
<tr>
<td>R6: Scalable</td>
<td>4</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>R7: Comprehensive</td>
<td>10</td>
<td>9</td>
<td>19</td>
</tr>
<tr>
<td>R8: Portable / Configurable</td>
<td>8</td>
<td>9</td>
<td>17</td>
</tr>
<tr>
<td>S1: Specific</td>
<td>6</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>S2: Accessible / Affordable</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Further reading: Chap. 7 and [Waller 2013, Waller and Hasselbring 2013, Waller et al. 2014]
## Benchmark Engineering (cont.)

![Flow diagram showing the process of Design/Implementation, Execution, and Analysis/Presentation]

### Requirements & Guidelines

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R9: Robust Execution</td>
<td>8</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>R10: Repeated Executions</td>
<td>3</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>R11: Warm-up / Steady State</td>
<td>2</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>R12 Idle Environment</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>R13: Statistical Analysis</td>
<td>7</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>R14: Reporting</td>
<td>6</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>R15: Validation</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>S3: Public Results Database</td>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

Further reading: Chap. 7 and [Waller 2013, Waller and Hasselbring 2013, Waller et al. 2014]
Benchmarks for Overhead

Three Portions of Overhead

**Method & Overhead Costs**
- $T$: normal execution time
- $I$: Instrumentation
- $C$: Collection of data
- $W$: Writing of data

**Determine each portion (one at a time):**

1. Determine $T$ in the benchmark system $T$
2. Add instrumentation $I$ $T + I$
3. Add data collection $C$ $T + I + C$
4. Add writing $W$ $T + I + C + W$

Further reading: Chap. 8 and [Waller and Hasselbring 2012, Waller 2013, Waller and Hasselbring 2013, Waller et al. 2014]
Three evaluation steps

1. Micro-Benchmarks
   – MooBench

2. Macro-Benchmarks
   – Pet Store
   – SPECjvm2008
   – SPECjbb2013

3. Meta-Monitoring
   – Kicker for Kieker

Further reading: Chap. 8, 9, 10 and [Waller 2013]
• Measures the **three causes of overhead**

• *Monitored Application*
  – single class; single method; fixed timing; configurable

• *Benchmark Driver*
  – initializes; executes; collects; records

• *Designed/implemented, executed, and analyzed/presented* according to our benchmark engineering methodology

Further reading: Chap. 8 and [Waller and Hasselbring 2012, Waller 2013, Waller and Hasselbring 2013, Waller et al. 2014]
Evaluate **performance & scalability** of environments for *Java business applications*

- World-wide supermarket company IT infrastructure

Further reading: Chap. 8 and [Waller 2013]

[http://research.spec.org/](http://research.spec.org/)
Meta-Monitoring

Monitoring the Monitoring Framework
- based upon Kieker 1.10
- **Kicker** available as tagged version in git

Challenges
- Monitoring the monitoring
  - prevent endless loops
- Minimize perturbation
  - aka meta-monitoring overhead

Further reading: Chap. 10 and [Waller 2013]
Systems Under Test (SUTs)

Kieker
http://kieker-monitoring.net
- Monitoring framework
- Research project
  - Oldenburg Univ.
  - Kiel University
  - Univ. of Stuttgart
- Focus on traces

ExplorViz
http://explorviz.net
- Monitoring tool
- Research project
  - Kiel University
- Focus on performance under high load

inspectIT
http://inspectit.eu
- Monitoring tool
- Commercial tool
  - NovaTec GmbH
- Focus on APM
  - Integrated analysis

SPASS-meter
http://ssehub.github.com
- Monitoring tool
- Research project
  - Univ. of Hildesheim
- Focus on resources
  - Integrated analysis

Further reading: Chap. 4, 12 and [van Hoorn et al. 2012, Fittkau et al. 2013a, Siegl and Bouillet 2011, Eichelberger and Schmid 2014]
Warm-up vs. Steady State (Example)

Further reading: Chap. 11 and [Waller and Hasselbring 2013]

Mean response time of...
- Writing (W)
- Instrumentation (I)
- Collecting (C)
- Method time (T)

10 JVMs
2,000,000 calls
10 stack depth
50% warm-up
minimal time T
Regression Benchmarks (Kieker)

Benchmark capabilities:
- Benchmark all versions of Kieker
- Compare releases with each other
- Detect performance regressions

Response time of...
- Writing (W)
- Collecting (C)
- Instrumentation (I)
- Method time (T)
(mean mean with 95% CI)

Further reading: Chap. 11 and [Waller and Hasselbring 2013]
Linear Increase of Overhead

**Execution time (median in µs)**

- **Writing (W)**
- **Collecting (C)**
- **Instrumentation (I)**
- **Method Time (T)**

**Recursion depth (number of successive method calls)**

**Slope (m)**
- $m = 0.1$
- $m = 0.9$
- $m = 0.1$

**Benchmark capabilities:**
- ✓ Benchmark with **scaling workloads**

Further reading: Chap. 11 and [Waller and Hasselbring 2012]
Multi-Core Architectures

Benchmark capabilities:
✓ Benchmark and compare different environments

Execution time (median in µs)

<table>
<thead>
<tr>
<th></th>
<th>X6270 Blade Server</th>
<th>X6240 Blade Server</th>
<th>T6330 Blade Server</th>
<th>T6340 Blade Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>synch</td>
<td>7,3</td>
<td>8,0</td>
<td>4,6</td>
<td>78,8</td>
</tr>
<tr>
<td>asynch</td>
<td>0,1</td>
<td>0,1</td>
<td>0,1</td>
<td>0,1</td>
</tr>
<tr>
<td>synch</td>
<td>2,7</td>
<td>2,5</td>
<td>2,6</td>
<td>9,4</td>
</tr>
<tr>
<td>asynch</td>
<td>1,0</td>
<td>1,0</td>
<td>1,4</td>
<td>1,4</td>
</tr>
<tr>
<td>synch</td>
<td>0,7</td>
<td>0,7</td>
<td>1,4</td>
<td>1,4</td>
</tr>
<tr>
<td>asynch</td>
<td>0,1</td>
<td>0,1</td>
<td>1,4</td>
<td>1,4</td>
</tr>
</tbody>
</table>

Further reading: Chap. 11 and [Waller and Hasselbring 2012]
Performance Tuning (with MooBench)

Benchmark capabilities:

✓ Benchmark to **guide a structured performance tuning** approach
✓ Benchmark **other tools** (ExplorViz monitoring)

Response time of ...
- Writing (W)
- Collecting (C)
- Instrumentation (I)
- Method time (T)

(mean with 95% CI)

Further reading: Chap. 11 and [Waller et al. 2014]
Compare high and low workloads

Response time (median in μs)

<table>
<thead>
<tr>
<th></th>
<th>Deactiv.</th>
<th>Collecting</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td>high load</td>
<td>2.4</td>
<td>25.0</td>
<td>2.4</td>
</tr>
<tr>
<td>low load</td>
<td>3.4</td>
<td>23.9</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Compare local and remote analysis (under high workload)

Response time (median in μs)

<table>
<thead>
<tr>
<th></th>
<th>Deactiv.</th>
<th>Collecting</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>2.4</td>
<td>25.0</td>
<td>2.4</td>
</tr>
<tr>
<td>remote</td>
<td>2.4</td>
<td>24.9</td>
<td>5.2</td>
</tr>
</tbody>
</table>

Benchmark capabilities:
- Additional commercial monitoring tools (inspectIT)
- Only minor adjustments required

Further reading: Chap. 12 and [Siegl and Bouillet 2011]
Benchmark capabilities:
- Additional **open-source monitoring tools** (SPASS-meter)
- Only **very minor adjustments** required

Further reading: Chap. 12 and [Eichelberger and Schmid 2014]
Determine capacity of system (workload as benchmark score)

<table>
<thead>
<tr>
<th></th>
<th>No instr.</th>
<th>Deactiv.</th>
<th>Collect.</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td>jOPS</td>
<td>268705</td>
<td>19490</td>
<td>2013</td>
<td>303</td>
</tr>
</tbody>
</table>

Run experiments using determined capacity

Response times (mean in ms with 95% CI)

- **Kieker**
  - No instr. 0.18
  - Deactiv. Collecting 1.02
  - Writing 3.57

- **MooBench**
  - Deactiv. 0.13
  - Collecting 1.00
  - Writing 3.05

- **SPECjbb**
  - Deactiv. 0.17
  - Collecting 1.00
  - Writing 3.49

Macro-benchmarks confirm findings of micro-benchmarks.

Further reading: Chap. 13
findings for development of Kieker

results for causes comparable to MooBench

Further reading: Chap. 14
Related Work & Outlook
Related Work

Benchmark Engineering Methodology

- **No encompassing methodology**
  - see [Hinnant 1988, Price 1989, Sachs 2011]

- Only **15 of 50 publications** on benchmark engineering

- **Execution of benchmarks mostly ignored in literature!**
  - Only recognized in more recent publication!
Measuring Monitoring Overhead

– Basic analysis (27 publications)
  • E.g., [Parsons et al. 2006, AppDynamics 2010]

– Causes of Overhead (13 publications)
  • E.g., [Kanstrén et al. 2011]

– Adaptive Monitoring (5 publications)
  • E.g., [Reiss 2008]

– Performance Evaluations of Kieker (4 publications)
  • E.g., [Focke 2006, Eichelberger and Schmid 2014]

Further reading: Chap. 15
Replication and Validation

- MooBench as open-source software
- All results available online
  - Raw results and generated diagrams
  - Prepared experiments for all Kieker versions
  - Detailed description of experiments

Publication of Benchmark Results

1. Publication only
2. Publication with code
3. Publication with source code and data
4. Full replication
5. Gold standard

Further reading: Chap. 16
Outlook & Future Work (cont.)

Benchmark Experiments

• Additional **monitoring frameworks/tools**

• Further **environments** besides Java

• Evaluation of **analysis overhead**
  – TeeTime [Wulf et al. 2014]

• Establish benchmarks in **community**
  – inspectIT

• Use in **continuous integration**
  – Automated regression benchmarks for Kieker [Waller et al. 2015]

Further reading: Chap. 16
Regression Benchmarks (Continuous Integration)

Benchmark capabilities:

✓ Automated benchmarks in **continuous integration**

Further reading: Chap. 11 and [Waller et al. 2015]
Bibliography


Backup Slides
**Goal, Question, Metric (GQM)**

<table>
<thead>
<tr>
<th>Goal</th>
<th>G</th>
<th>Measure and quantify the performance overhead of a monitoring framework with the MooBench micro-benchmark.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question</td>
<td>Q1</td>
<td>Which monitoring tools can be benchmarked?</td>
</tr>
<tr>
<td>Metrics</td>
<td>M1</td>
<td>tools or frameworks</td>
</tr>
<tr>
<td></td>
<td>M2</td>
<td>required changes for simple benchmarks</td>
</tr>
<tr>
<td></td>
<td>M3</td>
<td>required changes for cause analysis</td>
</tr>
<tr>
<td>Question</td>
<td>Q2</td>
<td>What effort is required to benchmark?</td>
</tr>
<tr>
<td>Metrics</td>
<td>M2</td>
<td>required changes for simple benchmarks</td>
</tr>
<tr>
<td></td>
<td>M3</td>
<td>required changes for cause analysis</td>
</tr>
<tr>
<td></td>
<td>M4</td>
<td>required run-time of the benchmark</td>
</tr>
<tr>
<td>Question</td>
<td>Q3</td>
<td>Can the monitoring overhead be quantified?</td>
</tr>
<tr>
<td>Metrics</td>
<td>M5</td>
<td>different scenarios</td>
</tr>
<tr>
<td></td>
<td>M6</td>
<td>configurability of the benchmark</td>
</tr>
<tr>
<td></td>
<td>M7</td>
<td>reproducibility of benchmark results</td>
</tr>
<tr>
<td>Question</td>
<td>Q4</td>
<td>Are the benchmark results representative?</td>
</tr>
<tr>
<td>Metrics</td>
<td>M7</td>
<td>reproducibility of benchmark results</td>
</tr>
<tr>
<td></td>
<td>M8</td>
<td>differences to other benchmarks</td>
</tr>
</tbody>
</table>
Micro-Benchmarks

Benchmark designed to measure individual portions

- **External Controller** configures **Monitoring** and **Driver**
- **Monitored Application** provides fixed $T$
- **Benchmark Threads**
  - call *monitored method*
    - #totalCalls
    - #recodedCalls
- Run **4 times** to measure

---

complete source code available at:
http://kieker-monitoring.net
Listing 8.2. Required Java interface for the Monitored Application

```java
public interface MonitoredClass {
    public long monitoredMethod(long methodTime, int recDepth);
}
```

Listing 8.3. Basic implementation of the MonitoredClass interface

```java
public final class MonitoredClassSimple implements MonitoredClass {
    public final long monitoredMethod(long methodTime, int recDepth) {
        if (recDepth > 1) {
            return this.monitoredMethod(methodTime, recDepth - 1);
        } else {
            final long exitTime = System.nanoTime() + methodTime;
            long currentTime;
            do {
                currentTime = System.nanoTime();
            } while (currentTime < exitTime);
            return currentTime;
        }
    }
}
```

Further reading: Chap. 8 and [Waller and Hasselbring 2013, Waller et al. 2014]
Listing 8.4. Excerpt of the Benchmark Thread’s run method

```java
public final void run() {
    long start_ns;
    long stop_ns;
    for (int i = 0; i < totalCalls; i++) {
        start_ns = System.nanoTime();
        monitoredClass.monitoredMethod(methodTime, recursionDepth);
        stop_ns = System.nanoTime();
        timings[i] = stop_ns - start_ns;
    }
}
```
Cause of Defect

• Supermarkets running out of wares under extremely high load

• Usual goal
  – generate high load
  – compare systems

• Our approach
  – utilizes rather low load
  – might be affected by defect
  – however, experiment easily repeatable, as soon as bug fix is released

http://www.spec.org/jbb2013/defectnotice.html
Performance comparison with MooBench
Experimental Setup (usually)

• X6270 Blade Server
  – 2x Intel Xeon 2.53 GHz
  – 24 GiB RAM
  – Solaris 10
  – Java 1.5 – 1.7 (64bit)
  – AspectJ

• X6240 AMD

• T6330/6340 SUN

Further reading: Chap. 11 and [Waller and Hasselbring 2012, Waller and Hasselbring 2013, Waller 2013, Waller et al. 2014]
Regression Benchmarks (Kieker)

Further reading: Chap. 11 and [Waller and Hasselbring 2013]...
Multi-Core Experiments

<table>
<thead>
<tr>
<th>Exp</th>
<th>Writer</th>
<th>Cores</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>SyncFS</td>
<td>1</td>
<td>single physical core</td>
</tr>
<tr>
<td>S2</td>
<td>SyncFS</td>
<td>2</td>
<td>two logical cores on the same physical core</td>
</tr>
</tbody>
</table>

X6270 Blade Server
2x Intel Xeon 2.53 GHz E5540 Quadcore / 24 GB RAM
Solaris 10 / Oracle Java x64 Server VM 1.6.0_26 (1 GB heap)
Meta-Monitoring (deactivated probe)
Benchmarking the Analysis

Benchmark capabilities:
- Additional **benchmark scenarios** (online analysis)
- Additional **environments** (private cloud)
- Additional **monitoring tools** (ExplorViz monitoring)

Further reading: Chap. 11 and [Fittkau et al. 2013b]