Application Performance Monitoring: Trade-Off between Overhead Reduction and Maintainability

Jan Waller, Florian Fittkau, and Wilhelm Hasselbring

2014-11-27
1. Introduction

2. Foundation

3. Performance Benchmark

4. Overhead Reduction and its Impact on Maintainability

5. Related Work

6. Future Work and Conclusions

7. References
Application level monitoring introduces monitoring overhead
Live trace processing approaches rely on high throughput
How to achieve?
Application level monitoring introduces monitoring overhead
Live trace processing approaches rely on high throughput
How to achieve?
Structured process for performance tunings utilizing benchmarks
Figure 1: UML component diagram of a top-level view on the Kieker framework architecture
Causes of Monitoring Overhead

Performance Benchmark

Figure 2: UML sequence diagram for method monitoring with the Kieker framework [WH13]
Figure 3: Benchmark engineering phases [WH13]
Figure 4: Time series diagram of measured timings
Overhead Reduction and its Impact on Maintainability

- Four performance tunings (PT1 to PT4)
- Used the benchmark for structured performance optimizations
- **Goal:** Low monitoring overhead and high throughput
- Every tuning is evaluated by the benchmark
- We will see whether usable in Kieker or not
Experimental Setup

Overhead Reduction and its Impact on Maintainability

- Modifying Kieker 1.8
- X6270 Blade Server with
  - 2x Intel Xeon 2.53 GHz E5540 Quadcore processors,
  - 24 GiB RAM, and
  - Solaris 10
### Overhead Reduction and its Impact on Maintainability

**Table 1:** Throughput for basis (traces per second)

<table>
<thead>
<tr>
<th></th>
<th>No instr.</th>
<th>Deactiv.</th>
<th>Collecting</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>1 176.5k</td>
<td>757.6k</td>
<td><strong>63.2k</strong></td>
<td><strong>16.6k</strong></td>
</tr>
<tr>
<td><strong>95% CI</strong></td>
<td>± 25.9k</td>
<td>± 5.5k</td>
<td>± 0.1k</td>
<td>± 0.02k</td>
</tr>
<tr>
<td><strong>Q1</strong></td>
<td>1 189.2k</td>
<td>756.6k</td>
<td>63.0k</td>
<td>16.2k</td>
</tr>
<tr>
<td><strong>Median</strong></td>
<td>1 191.2k</td>
<td>765.9k</td>
<td>63.6k</td>
<td>16.8k</td>
</tr>
<tr>
<td><strong>Q3</strong></td>
<td>1 194.6k</td>
<td>769.8k</td>
<td>63.9k</td>
<td>17.2k</td>
</tr>
</tbody>
</table>
Analysis

Overhead Reduction and its Impact on Maintainability

- High monitoring overhead in:
  - Collection of data and
  - actually writing the gathered data
- Expensive Reflection API calls
- Reuse of signature of operations
### PT1: Caching & Cloning

**Overhead Reduction and its Impact on Maintainability**

<table>
<thead>
<tr>
<th></th>
<th>No instr.</th>
<th>Deactiv.</th>
<th>Collecting</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>1 176.5k</td>
<td>757.6k</td>
<td>63.2k</td>
<td>16.6k</td>
</tr>
</tbody>
</table>

**Table 2:** Throughput for basis (traces per second)

<table>
<thead>
<tr>
<th></th>
<th>No instr.</th>
<th>Deactiv.</th>
<th>Collecting</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>1 190.5k</td>
<td>746.3k</td>
<td><strong>78.2k</strong></td>
<td><strong>31.6k</strong></td>
</tr>
<tr>
<td><strong>95% CI</strong></td>
<td>± 4.1k</td>
<td>± 4.1k</td>
<td>± 0.1k</td>
<td>± 0.1k</td>
</tr>
</tbody>
</table>

**Table 3:** Throughput for PT1 (traces per second)
Overhead Reduction and its Impact on Maintainability

- Will be used in Kieker since not impacting interfaces
From PT1: Queue is saturated and the monitoring thread waits for a free space in the queue

Target: Decrease the synchronization impact of writing data

Optimize the communication between monitoring and writer thread

**Disruptor** instead of Java’s **ArrayBlockingQueue**
### PT2: Inter-Thread Communication

Overhead Reduction and its Impact on Maintainability

<table>
<thead>
<tr>
<th></th>
<th>No instr.</th>
<th>Deactiv.</th>
<th>Collecting</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>1,190.5k</td>
<td>746.3k</td>
<td>78.2k</td>
<td>31.6k</td>
</tr>
</tbody>
</table>

**Table 4**: Throughput for PT1 (traces per second)

<table>
<thead>
<tr>
<th></th>
<th>No instr.</th>
<th>Deactiv.</th>
<th>Collecting</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>1,190.5k</td>
<td>757.6k</td>
<td>78.2k</td>
<td>56.0k</td>
</tr>
<tr>
<td><strong>95% CI</strong></td>
<td>± 3.6k</td>
<td>± 6.2k</td>
<td>± 0.1k</td>
<td>± 0.2k</td>
</tr>
</tbody>
</table>

**Table 5**: Throughput for PT2 (traces per second)
Will be used in Kieker since only impacting communication between MonitoringController and Writers
Analysis

Overhead Reduction and its Impact on Maintainability

- From PT2: Monitoring thread is waiting for the writer thread to finish
- Target: Decrease the writing time
- Reduce the conducted work of the writer thread
- Flat record model (ByteBuffer)
# PT3: Flat Record Model

Overhead Reduction and its Impact on Maintainability

<table>
<thead>
<tr>
<th></th>
<th>No instr.</th>
<th>Deactiv.</th>
<th>Collecting</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>1 190.5k</td>
<td>757.6k</td>
<td>78.2k</td>
<td>56.0k</td>
</tr>
</tbody>
</table>

*Table 6: Throughput for PT2 (traces per second)*

<table>
<thead>
<tr>
<th></th>
<th>No instr.</th>
<th>Deactiv.</th>
<th>Collecting</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>1 176.5k</td>
<td>729.9k</td>
<td>115.7k</td>
<td>113.2k</td>
</tr>
<tr>
<td><strong>95% CI</strong></td>
<td>± 2.1k</td>
<td>± 4.4k</td>
<td>± 0.2k</td>
<td>± 0.5k</td>
</tr>
</tbody>
</table>

*Table 7: Throughput for PT3 (traces per second)*
Discussion

Overhead Reduction and its Impact on Maintainability

- Will not be used in Kieker since monitoring records now writing bytes directly to buffers (less maintainable)
From PT3: About 80% spent time in collecting phase

Target: Decrease the collecting time

Remove interface definitions, configurability, and consistence checks

Five hard coded types of MonitoringRecords
## PT4: Minimal Monitoring Code

Overhead Reduction and its Impact on Maintainability

<table>
<thead>
<tr>
<th></th>
<th>No instr.</th>
<th>Deactiv.</th>
<th>Collecting</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>1 176.5k</td>
<td>729.9k</td>
<td>115.7k</td>
<td>113.2k</td>
</tr>
</tbody>
</table>

**Table 8:** Throughput for PT3 (traces per second)

<table>
<thead>
<tr>
<th></th>
<th>No instr.</th>
<th>Deactiv.</th>
<th>Collecting</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>1 190.5k</td>
<td>763.3k</td>
<td><strong>145.1k</strong></td>
<td><strong>141.2k</strong></td>
</tr>
<tr>
<td><strong>95% CI</strong></td>
<td>± 2.0k</td>
<td>± 4.0k</td>
<td>± 0.2k</td>
<td>± 0.3k</td>
</tr>
</tbody>
</table>

**Table 9:** Throughput for PT4 (traces per second)
Results and Discussion

Overhead Reduction and its Impact on Maintainability

▶ Will not be used in Kieker since breaking the framework idea
At least one core was available for the monitoring

Common threats of micro-benchmarks (relevance and systematic errors)

Different memory layouts of programs or JIT compilation paths
Figure 5: Overview of the tuning results in response time
Related Work

- Dapper
- Magpie
- X-Trace
- SPASS-meter
Future Work

Future Work and Conclusions

- Reduce the impact of deactivated probes by, for instance, DiSL
- Generator handling the monitoring record byte serialization
- Multi-threaded versions of our monitoring benchmark
- Compare to other benchmarks
Conclusions

Future Work and Conclusions

▶ Proposed micro-benchmark for monitoring frameworks

MooBench

▶ Tunings show an upper limit for the monitoring overhead

▶ Useful for live trace processing in the context of ExplorViz¹

¹http://www.explorviz.net
Jan Waller and Wilhelm Hasselbring.  
A benchmark engineering methodology to measure the overhead of application-level monitoring.  