Refactoring Kieker's Monitoring Component to Further Reduce the Runtime Overhead

Symposium on Software Performance 2016

Hannes Strubel and Christian Wulf

08.11.2016

Software Engineering Group
Kiel University, Germany
Low monitoring overhead  [WallerSSP13]

Fast Pipe-and-Filter-based analyses  (migration completed soon)  [WulfSSP14]

Resolved performance anti-patterns [WulfSSP15]
Agenda

• Introduction

• Current Monitoring Component

• Refactored Monitoring Component

• Evaluation
  – Complexity
  – Runtime Overhead

• Conclusions
The Application threads atomically increase the number of transferred records.
The WriterController checks for the type of the incoming record.
The used implementation of the synchronized queues is very slow.
The string registry
  - is synchronized since it is used by two writer threads
  - is maintained by the registry controller
Each writer has to declare and to manage its own set of worker threads.
Refactored Monitoring Component

- Introduction of a new event-based system
  => replaces the atomic counter
- Removal of registry records
  => application threads do not check for record types anymore
- Avoidance of one of the two queues
- Introduction of a high-performance, lock-free queue based on JCTools\(^1\)
  => replaces the slow blocking queue
- No synchronous writer anymore

\(^1\): https://github.com/JCTools/JCTools
Refactored Monitoring Component

- Remaining writer thread first checks a record’s the string attributes
- String registry declaration on demand by each writer
- Faster, unsynchronized string registry
- No need for the registry controller anymore
- Writer controller now handles the thread management
Evaluation

• Complexity
  – Using the Hypergraph-based Software Evaluation-Plugin for Eclipse¹

• Runtime Overhead
  – Using MooBench²

¹ https://build.se.informatik.uni-kiel.de/eus/se/snapshot
² https://build.se.informatik.uni-kiel.de/kieker/moobench
## Complexity Evaluation

<table>
<thead>
<tr>
<th>Kieker Version</th>
<th>Lines of Code</th>
<th>Cyclomatic Complexity</th>
<th>Information Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>1092</td>
<td>2.48</td>
<td>242</td>
</tr>
<tr>
<td>Refactored</td>
<td>(60%) 654</td>
<td>(70%) 1.73</td>
<td>(73%) 176</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>Current</td>
<td>0.087</td>
<td>0.478</td>
<td>2.607</td>
</tr>
<tr>
<td>95%-ci (±)</td>
<td>0.000</td>
<td>0.005</td>
<td>0.007</td>
</tr>
<tr>
<td>Refactored*</td>
<td>0.088</td>
<td>0.468</td>
<td>2.335</td>
</tr>
<tr>
<td>95%-ci (±)</td>
<td>0.000</td>
<td>0.004</td>
<td>0.007</td>
</tr>
<tr>
<td>Refactored</td>
<td>0.083</td>
<td>0.472</td>
<td>2.384</td>
</tr>
<tr>
<td>95%-ci (±)</td>
<td>0.000</td>
<td>0.006</td>
<td>0.012</td>
</tr>
</tbody>
</table>
Conclusions

• Refactored architecture of the monitoring component
• Information complexity reduced to 73 %
• Runtime overhead reduced to 17 % (!)

Future work
• Record pooling to reduce pressure on GC
• Record data pre-filling to minimize data collection at runtime

Record Pooling

Lifetime of a record within the monitoring component

App → Probes → Monitoring Controller → Writer Controller → Writer

Application thread

Writer Registry

Writer thread

creation

sending & discard

pool
Record Data Pre-Filling

AspectJ limited: unable to intercept weaving process

=> No access to static code information at weaving, such as class and method name

Proposition: instrumentation via a low-level framework such as ASM/Javassist
