Refactoring Kieker’s I/O Infrastructure to Improve Scalability and Extensibility

Holger Knoche

University of Kiel

November 10, 2017 @ SSP
Agenda

1. Motivation

2. Kieker’s I/O Infrastructure Today

3. Kieker’s I/O Infrastructure Tomorrow

4. Performance Evaluation

5. Conclusions and Future Directions
Need for Scalability

Motivation

- Large, scalable software systems produce large amounts of monitoring data
- Modern messaging solutions provide the basis for scalable, distributed data processing
- But: Kieker’s I/O infrastructure currently does not leverage the scalability of such platforms
Anatomy of a Reader-Writer Pair

Kieker’s I/O Infrastructure Today

- Reader and writer reside in separate components
- No place to put specific dependencies and test cases
- Data format hard-coded into the readers and writers

ConcreteReader
AbstractReaderPlugin
ConcreteWriter
AbstractMonitoringWriter
writeMonitoringRecord(record)
Kieker’s default binary protocol uses string tables to avoid redundantly transferring string values.

1. All strings from a monitoring record are replaced by numeric IDs from the table
2. If a new string is encountered, a new ID is assigned and a special record is sent to update the consumer’s table
3. The producer sends the encoded record
4. The consumer decodes the record using its own copy of the string table

**Problem:** The string table is stateful.
1. Stateful transfer protocol
   ▶ Does not work with multiple record consumers
   ▶ Prevents scalability

2. Data format is hard-coded into the writers
   ▶ Little flexibility in terms of data format

3. Only one record is processed at a time
   ▶ Increased overhead in messaging systems

4. Cumbersome separation of readers and writers
   ▶ Leads to global dependencies
   ▶ No place to put reader-writer tests
Collectors collect multiple monitoring records and orchestrate the process of encoding and transferring the data

Serializers and Deserializers encode monitoring records in a particular format and vice versa

Raw Data Readers and Writers are responsible for transferring the data using a particular medium
1. Stateful transfer protocol
   ✓ Fixed by a new, stateless container format

2. Data format is hard-coded into the writers
   ✓ (De-)Serializers can be selected independent of the reader / writer

3. Only one record is processed at a time
   ✓ Collectors allow to collect and batch-process multiple records

4. Cumbersome separation of readers and writers
   ✓ Raw readers and writers (and their dependencies) can now be put into the same component
* Mission Accomplished *

But...
Major changes were made to Kieker’s I/O Infrastructure, which may affect performance.

**Performance Evaluation Questions:**

**EQ1** Does the refactoring have a (negative) performance impact on the monitored application?

**EQ2** To what extent does the chunking affect the overall resource consumption when using a messaging technology?
We...

- used MooBench
- used “null” writers that only did serialization
- ran everything on a Raspberry Pi 3
- evaluated four configurations

<table>
<thead>
<tr>
<th>Configuration</th>
<th>95% CI (in $\mu$s)</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>[106.0;106.2]</td>
<td>21.0</td>
</tr>
<tr>
<td>Current infrastructure</td>
<td>[152.8;152.9]</td>
<td>28.3</td>
</tr>
<tr>
<td>Collector (no bypass)</td>
<td>[163.3;163.6]</td>
<td>64.9</td>
</tr>
<tr>
<td>Collector (bypass)</td>
<td>[141.5;141.6]</td>
<td>40.3</td>
</tr>
</tbody>
</table>
We...

- used a test harness issuing records at a constant rate
- measured CPU utilization by the harness process and overall network utilization
- ran the benchmarks on an otherwise idle system
- evaluated different chunk sizes

<table>
<thead>
<tr>
<th>Chunk size</th>
<th>95% CI CPU in CPU sec. / sec.</th>
<th>95% CI net in KiB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old writer</td>
<td>[0.612;0.620]</td>
<td>[1,913.7;1,914.2]</td>
</tr>
<tr>
<td>1</td>
<td>[0.768;0.780]</td>
<td>[2,706.4;2,710.3]</td>
</tr>
<tr>
<td>16</td>
<td>[0.460;0.477]</td>
<td>[684.2;684.3]</td>
</tr>
<tr>
<td>32</td>
<td>[0.273;0.275]</td>
<td>[639.9;640.1]</td>
</tr>
<tr>
<td>128</td>
<td>[0.340;0.356]</td>
<td>[593.8;594.2]</td>
</tr>
<tr>
<td>1024</td>
<td>[0.338;0.352]</td>
<td>[577.3;581.1]</td>
</tr>
</tbody>
</table>
* Mission Really Accomplished *
We...

▶ ...prepared Kieker to leverage the scalability of modern messaging infrastructures
▶ ...improved extensibility and flexibility along the way
▶ ...did not break anything in terms of performance
▶ ...even achieved performance improvements for message writers
Distributed Trace Processing

Conclusions and Future Directions

Monitored Nodes

Trace Preprocessing / Filtering Nodes

Trace Processing Nodes

Holger Knoche  Refactoring Kieker's I/O Infrastructure  November 10, 2017 @ SSP