Graph-Based Performance Analysis at System- and Application-Level

Leipzig, November 13, 2020
Richard Müller and Tom Strempel
KIEKER

- The Kieker framework provides
  - monitoring,
  - analysis,
  - and visualization support
  for
  - application and system performance analysis as well as
  - reverse engineering

KIEKER PLUGIN

- Transforms monitored log data into graphs
- Supports software engineers with performance analysis and architecture discovery

KIEKER PLUGIN ISSUES

- No support for system-level information, such as CPU and system memory utilization
- High disk usage due to redundant information in the graph schema
- Long scan times due to implementation flaws
- No evaluation with regard to scalability
CONTRIBUTIONS

– Kieker plugin was extended and improved to solve the aforementioned issues
– Correctness and scalability of the revised plugin were evaluated by
  – processing data and
  – reproducing analysis results of two recent experiments
– A reproduction package is provided to replicate the evaluation: https://github.com/softvis-research/SSP2020
OLD KIEKER GRAPH SCHEMA

controllerName: String
experimentId: int
fileName: String
hostname: String
loggingTimestamp: long
numberOfRecords: int
timeOffset: int
timeUnit: String
version: String

traceId: long
threadId: long
hostName: String
loggingTimestamp: long
sessionId: String

beforeTimestamp: long
afterTimestamp: long
beforeOrderIndex: int
afterOrderIndex: int

name: String
signature: String
duration: long
incomingCalls: int
outgoingCalls: int

calledBy: CALLS
executes: EXECUTES
declares: DECLARES
dependsOn: DEPENDS_ON

[Müller and Fischer 2019]
REVISED KIEKER GRAPH SCHEMA
1ST EXPERIMENT - HORA: ARCHITECTURE-AWARE ONLINE FAILURE PREDICTION

- Combine component failure predictors with architectural knowledge to improve failure prediction

Fig. 1. Running example: high-level three-tier architecture and selected measurements.

[Pitakrat et al. 2018]
PERFORMANCE ANALYSIS AT SYSTEM-LEVEL

Reproduce two line charts showing the system-level measures CPU and system memory utilization of the second business-tier instance from the first experiment [Pitakrat et al. 2018]
CYPHPER QUERY FOR CPU UTILIZATION

MATCH (r:Record)-[:CONTAINS]->(c:CpuUtilization)
WHERE r.fileName =~ '.*/1-MemoryLeak-5/kieker-logs\nkieker-20150820-064855519-UTC-middletier2-KIEKER'
RETURN c.timestamp AS timestamp, c.cpuID AS cpuID, c.totalUtilization * 100 AS cpuUtilization
ORDER BY timestamp

<table>
<thead>
<tr>
<th>timestamp</th>
<th>cpuID</th>
<th>cpuUtilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1440053336119231206</td>
<td>&quot;0&quot;</td>
<td>89.99999999999999</td>
</tr>
<tr>
<td>1440053336119479386</td>
<td>&quot;1&quot;</td>
<td>98.03921568627452</td>
</tr>
<tr>
<td>1440053345612040829</td>
<td>&quot;0&quot;</td>
<td>12.882787750791975</td>
</tr>
<tr>
<td>1440053345612101780</td>
<td>&quot;1&quot;</td>
<td>24.26160337552743</td>
</tr>
<tr>
<td>1440053355611996675</td>
<td>&quot;0&quot;</td>
<td>3.60000000000000005</td>
</tr>
</tbody>
</table>
2ND EXPERIMENT - COMPARING STATIC AND DYNAMIC WEIGHTED SOFTWARE COUPLING METRICS

Investigate how weighted dynamic coupling measurements can support software engineers to evaluate the architectural quality of software systems.

Table 1. Numbers of users and monitored calls.

<table>
<thead>
<tr>
<th>#</th>
<th>Date</th>
<th>Users</th>
<th>Method Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>February 2017</td>
<td>19</td>
<td>196,442,044</td>
</tr>
<tr>
<td>2</td>
<td>September 2017</td>
<td>48</td>
<td>854,657,027</td>
</tr>
<tr>
<td>3</td>
<td>February 2018</td>
<td>16</td>
<td>475,357,185</td>
</tr>
<tr>
<td>4</td>
<td>September 2018</td>
<td>58</td>
<td>2,409,688,701</td>
</tr>
</tbody>
</table>

Table 10. Average Coupling Degrees in our four Experiments.

<table>
<thead>
<tr>
<th>#</th>
<th>static classes</th>
<th>packages</th>
<th>dynamic classes</th>
<th>packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>730</td>
<td>8742</td>
<td>40,058</td>
<td>143,483</td>
</tr>
<tr>
<td>2</td>
<td>586</td>
<td>6922</td>
<td>144,403</td>
<td>592,232</td>
</tr>
<tr>
<td>3</td>
<td>580</td>
<td>6554</td>
<td>80,698</td>
<td>375,121</td>
</tr>
<tr>
<td>4</td>
<td>580</td>
<td>6554</td>
<td>370,821</td>
<td>1,868,664</td>
</tr>
</tbody>
</table>
PERFORMANCE ANALYSIS AT APPLICATION-LEVEL

- Plugin processes 2,409,688,701 method calls and reproduces the weighted dynamic dependency graphs at class and package level from the second experiment [Schnoor and Hasselbring 2020]
- Disk usage
  - Original tar.xz file: 8.89 GB
  - Graph database: 110 MB*
- Scan and graph creation time
  - 1h 38min 29s

* This reduction is mainly due to omitting the node types Event and Trace including their properties.
**CYPER QUERY FOR METHOD CALLS**

MATCH (:Method:Kieker)-[calls:CALLS]->(:Method:Kieker)
RETURN SUM(calls.weight) AS methodCalls

<table>
<thead>
<tr>
<th>#</th>
<th>Date</th>
<th>Users</th>
<th>Method Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>February 2017</td>
<td>19</td>
<td>196,442,044</td>
</tr>
<tr>
<td>2</td>
<td>September 2017</td>
<td>48</td>
<td>854,657,027</td>
</tr>
<tr>
<td>3</td>
<td>February 2018</td>
<td>16</td>
<td>475,357,185</td>
</tr>
<tr>
<td>4</td>
<td>September 2018</td>
<td>58</td>
<td><strong>2,409,688,701</strong></td>
</tr>
</tbody>
</table>
CYPHER QUERY FOR AVERAGE EXPORT COUPLING DEGREE ON CLASS LEVEL

MATCH (t:Type:Kieker)
WHERE (t)-[:DEPENDS_ON]->() OR ()-[:DEPENDS_ON]->(t)
WITH t
OPTIONAL MATCH (t)-[:out:DEPENDS_ON]->()
WITH t, SUM(out.weight) AS import
OPTIONAL MATCH ()-[:in:DEPENDS_ON]->(t)
WITH t, import, SUM(in.weight) AS export
RETURN ROUND(AVG(export)) AS averageExport

<table>
<thead>
<tr>
<th>averageExport</th>
<th>=</th>
</tr>
</thead>
<tbody>
<tr>
<td>370821.0</td>
<td></td>
</tr>
</tbody>
</table>

Table 10. Average Coupling Degrees in our four Experiments.
REPRODUCTION PACKAGE

SSP2020

Reproduction package for the paper "Graph-Based Performance Analysis at System- and Application-Level"

Please, click on the binder badge to start the mybinder environment. Then you can run the jupyter notebooks (1. Performance analysis at system-level.ipynb and 2. Performance analysis at application-level.ipynb) and replicate the analyses.

External Credits

- Software Analytics with Python
- Binder and Neo4j integration

[https://github.com/softvis-research/SSP2020]
FUTURE WORK

- Replicate the complete experiment from [Schnoor and Hasselbring 2020]
- Kieker plugin will be used to generate dynamic dependency graphs
- Java bytecode scanner plugin will be used to generate static dependency graphs

[https://github.com/jQAssistant/jqa-java-plugin]
REFERENCES

- R. Müller and M. Fischer. "Graph-Based Analysis and Visualization of Software Traces". In: 10th Symposium on Software Performance: Joint Developer and Community Meeting of Descartes/Kieker/Palladio. Würzburg, Germany, 2019.
THANK YOU.

Richard Müller  
Information Systems Institute, Chair of Software Engineering, Leipzig University  
Tom Strempel  
Master student in Computer Science, Leipzig University

rmueller@wifa.uni-leipzig.de  
@rimllr  
https://github.com/softvis-research  
http://softvis.wifa.uni-leipzig.de