Workload-Intensity-Sensitive Timing Behavior Analysis for Distributed Multi-User Software Systems

Matthias Rohr\textsuperscript{1,2}, André van Hoorn\textsuperscript{2}, Wilhelm Hasselbring\textsuperscript{2,3}, Marco Lübcke\textsuperscript{4}, and Sergej Alekseev\textsuperscript{5,6}

\textsuperscript{1} BTC Business Technology Consulting AG, Germany, \textsuperscript{*}, \textsuperscript{2} Graduate School TrustSoft, University of Oldenburg, Germany, \textsuperscript{3} Software Engineering Group, University of Kiel, Germany, \textsuperscript{4} CeWe Color AG & Co. OHG, Oldenburg, Germany, \textsuperscript{5} Nokia Siemens Networks GmbH, Berlin, Germany, \textsuperscript{6} Hochschule Mittweida, University of Applied Sciences, Mittweida, Germany

January 29, 2010
Joint WOSP/SIPEW International Conference on Performance Engineering, San Jose, California, USA

\textsuperscript{*} This work is supported by the German Research Foundation (DFG), grant GRK 1076/1
Workload-intensity can be a major influence to timing behavior in enterprise information systems.

Varying workload-intensity can cause high variance in timing behavior.

High variance can make it difficult to draw statistical conclusions.

E.g., proper threshold determination for anomaly detection.
Workload-intensity can be a major influence to timing behavior in enterprise information systems. Varying workload-intensity can cause high variance in timing behavior. High variance can make it difficult to draw statistical conclusions. E.g., proper threshold determination for anomaly detection.
Motivation 2/2 - Approach idea

Without considering workload intensity

Software system with monitoring instrumentation

Response time monitoring

Measurement data

Dynamic analysis, Trace reconstruction

Architecture models

Timing behavior model

Conclusions

Our approach

Goal: "Reduce" variation for statistical timing behavior analysis

Categorization based on workload-intensity levels

Requires only light-weight common monitoring infrastructure

Matthias Rohr, BTC AG, Workload-Intensity-Sensitive Timing Behavior
**Motivation 2/2 - Approach idea**

**With considering workload intensity**

Software system with monitoring instrumentation

Response time monitoring

Measurement data

**Workload intensity**

Dynamic analysis, Trace reconstruction

Architecture models

Timing behavior model

Conclusions

**Our approach**
Motivation 2/2 - Approach idea

With considering workload intensity

Software system with monitoring instrumentation

Response time monitoring

Measurement data

Dynamic analysis, Trace reconstruction

Architecture models

Workload intensity

Timing behavior model

Conclusions

Our approach

- Goal: “Reduce” variation for statistical timing behavior analysis
- Categorization based on workload-intensity levels
- Requires only light-weight common monitoring infrastructure

Matthias Rohr, BTC AG, Workload-Intensity-Sensitive Timing Behavior

Influences to Software Timing Behavior

- System architecture and implementation:
  - Hardware design
  - Software design
  - Middleware

- System usage:
  - Workload-intensity
    - Concurrent service requests [Happe et al. 2008]
    - Number of active users
  - Individual request characteristics
    - Parameter values and parameter size
    - Caller identity / stack content

- State:
  - Cache content
  - Load balancer state
  - Software application state
  - Other active processes on same platform
  - Database content
Response times and workload intensity

Relation between response times and workload intensity

(Schematic illustration based on ?)
Response times and execution times

**Figure:** Response times and execution times.
Motivation

Foundations

Workload-intensity-sensitive timing behavior analysis

Empirical evaluation

Related work

Conclusions and future work
1. Monitoring

- Recording of:
  - **Response times**: Time between start and end of software operation executions
  - **Execution sequences** corresponding to a user request
  - Host identifier

- Reconstruction of Traces and Dependency Graphs
- Kieker framework\(^a\) [?]

\(^a\)http://kieker.sourceforge.org

2. Computation of workload-intensity from monitoring data:

3. Categorization based on workload-intensity levels
1. Monitoring

2. Computation of workload-intensity from monitoring data:
   → next slides

3. Categorization based on workload-intensity levels
1. Monitoring

2. Computation of workload-intensity from monitoring data:

3. Categorization based on workload-intensity levels

1. The $pwi$ range is divided into intervals (e.g., 15) of equal length
2. Bins are extended to minimum size (e.g., 100 observations)
Key element of our approach: Four alternative workload-intensity metrics, denoted \( pwi \) (Platform Workload Intensity):

<table>
<thead>
<tr>
<th>Metric ( pwi_1 )</th>
<th>Time metric</th>
<th>Execution environment</th>
<th>Operation weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Response times</td>
<td>Non-distributed</td>
<td>No weighting</td>
</tr>
<tr>
<td>( pwi_2 )</td>
<td>Execution times</td>
<td>Non-distributed</td>
<td>No weighting</td>
</tr>
<tr>
<td>( pwi_3 )</td>
<td>Execution times</td>
<td>Distributed</td>
<td>No weighting</td>
</tr>
<tr>
<td>( pwi_4 )</td>
<td>Execution times</td>
<td>Distributed</td>
<td>Learned</td>
</tr>
</tbody>
</table>
Average number of concurrent traces during the time period between the start (call action) and the end of an operation execution.
Average number of concurrent traces during the time period between the start (call action) and the end of an operation execution.

Figure: Example traces: UML Sequence Diagrams
Average number of concurrent traces during the time period between the start (call action) and the end of an operation execution.

<table>
<thead>
<tr>
<th>Operation, execution</th>
<th>Trace</th>
<th>Execution environment</th>
<th>pwi</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>a()</td>
<td>1</td>
<td>1</td>
<td>1.5</td>
<td>0</td>
</tr>
<tr>
<td>b()</td>
<td>1</td>
<td>1</td>
<td>1.4</td>
<td>10</td>
</tr>
<tr>
<td>c(),1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>c(),2</td>
<td>1</td>
<td>1</td>
<td>1.5</td>
<td>30</td>
</tr>
<tr>
<td>d()</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>e()</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>50</td>
</tr>
</tbody>
</table>

Legend:
- **Active**
- **Not active**

Graphical representation:
- The chart illustrates the active and not active state of operations over time.
- Each operation (a(), b(), c(), d(), e()) is represented with a horizontal bar indicating the duration of its execution.
- The pwi values (1.5, 1.4, 2) correspond to the average number of concurrent traces for each operation during a specific time period.
Average number of concurrent traces during the time period between the start (call action) and the end of an operation execution.
**Average number of concurrent traces during the time period between the start (call action) and the end of an operation execution.**

<table>
<thead>
<tr>
<th>operation, execution</th>
<th>trace</th>
<th>exec. env.</th>
<th>$pwi_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a()</td>
<td>1</td>
<td>1</td>
<td>1.5</td>
</tr>
<tr>
<td>b()</td>
<td>1</td>
<td>1</td>
<td>1.4</td>
</tr>
<tr>
<td>c(),1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>c(),2</td>
<td>1</td>
<td>1</td>
<td>1.5</td>
</tr>
<tr>
<td>d()</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>e()</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

![Graph](image)
An operation execution’s $pwi_2$ is the average number of concurrent traces during its execution time period.

- Difference to $pwi_1$: Execution time period instead of response time period
- No competition for resources during waiting for sub-calls
An operation execution’s $pwi_2$ is the average number of concurrent traces during its execution time period.

- **Difference to $pwi_1$**: Execution time period instead of response time period
- **No competition for resources during waiting for sub-calls**

### Operation Execution Table

<table>
<thead>
<tr>
<th>Operation</th>
<th>Trace</th>
<th>Execution Environment</th>
<th>$pwi_1$</th>
<th>$pwi_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a()</td>
<td>1</td>
<td>1</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>b()</td>
<td>1</td>
<td>1</td>
<td>1.4</td>
<td>1.33</td>
</tr>
<tr>
<td>c(),1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>c(),2</td>
<td>1</td>
<td>1</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>d()</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>e()</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

### Diagram

- **Active** operation executions are shaded.
- **Not active** operation executions are not shaded.

Matthias Rohr, BTC AG, Workload-Intensity-Sensitive Timing Behavior
An operation execution’s $pwi_3$ is the average number of concurrent **active executions within the same execution environment** during its execution time period.

- $pwi_3$ extends $pwi_2$ for distributed systems.
- Assumption: Execution contexts have own hardware platform
- Hypothesis: Little competition for resources with executions in other execution environments.
An operation execution’s $pw_i^3$ is the average number of concurrent active executions within the same execution environment during its execution time period.
An operation execution’s $pw_{i3}$ is the average number of concurrent active executions within the same execution environment during its execution time period.

<table>
<thead>
<tr>
<th>operation, execution</th>
<th>trace</th>
<th>exec. env.</th>
<th>$pw_{i1}$</th>
<th>$pw_{i2}$</th>
<th>$pw_{i3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a()</td>
<td>1</td>
<td>1</td>
<td>2.2</td>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>b()</td>
<td>1</td>
<td>1</td>
<td>2.2</td>
<td>2</td>
<td>1.66</td>
</tr>
<tr>
<td>c(),1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>c(),2</td>
<td>1</td>
<td>1</td>
<td>2.5</td>
<td>2.5</td>
<td>1.5</td>
</tr>
<tr>
<td>d()</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2.25</td>
</tr>
<tr>
<td>e()</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>g()</td>
<td>3</td>
<td>1</td>
<td>2.71</td>
<td>2.5</td>
<td>2.5</td>
</tr>
<tr>
<td>h()</td>
<td>3</td>
<td>2</td>
<td>2.8</td>
<td>2.8</td>
<td>1</td>
</tr>
</tbody>
</table>
*pwi*$_4$ extends *pwi*$_3$ by using the weight $w_{o,p} \in W$ for considering concurrent executions of $p$ for evaluating $o$.

- *pwi*$_1$-*pwi*$_3$ equally consider different (local) operations
- Resource competition leads to high weights.

**Computation of weight matrix $W$**

- $W$ is determined via machine learning from historical monitoring data
- Learning goal: maximum standard deviation reduction
- High computational costs if many operations are instrumented
- Convention: $w_{o,p}$ is 0, if $o$ and $p$ are not in the same execution environment
- Heuristic: Correlation matrix provides good starting values
Example 1/2

**Software system with 2 operations:**

- *Wait*: Non-busy waiting for 300 ms.
- *Work*: CPU-intensive number crunching.
Software system with 2 operations:

- *Wait*: Non-busy waiting for 300 ms.
- *Work*: CPU-intensive number crunching.

Experiment setting:

- 120,000 random execution of *wait* and *work*
- 1-24 parallel executions
Software system with 2 operations:

- **Wait**: Non-busy waiting for 300 ms.
- **Work**: CPU-intensive number crunching.

Experiment setting:

- 120,000 random execution of *wait* and *work*
- 1-24 parallel executions

Results:

Weight matrix:

<table>
<thead>
<tr>
<th></th>
<th>work</th>
<th>wait</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>work</strong></td>
<td>2.01</td>
<td>-0.05</td>
</tr>
<tr>
<td><strong>wait</strong></td>
<td>1.03</td>
<td>0.05</td>
</tr>
</tbody>
</table>
Software system with 2 operations:

- **Wait**: Non-busy waiting for 300 ms.
- **Work**: CPU-intensive number crunching.

Experiment setting:

- 120,000 random execution of *wait* and *work*
- 1-24 parallel executions

Results:

**Weight matrix:**

<table>
<thead>
<tr>
<th></th>
<th>work</th>
<th>wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>work</td>
<td>2.01</td>
<td>-0.05</td>
</tr>
<tr>
<td>wait</td>
<td>1.03</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**Standard dev. reduction (%):**

<table>
<thead>
<tr>
<th></th>
<th><em>pwi</em>&lt;sub&gt;4&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>work</td>
<td>72.5 ± 2</td>
</tr>
<tr>
<td>wait</td>
<td>18.8 ± 9</td>
</tr>
</tbody>
</table>
**pwi_4 Example 2/2**

![Boxplot Diagrams](image)

- **(a) wait**
- **(b) work**

**Figure:** Example boxplots: Relation between $pwi_4$ and execution times.

Matthias Rohr, BTC AG, Workload-Intensity-Sensitive Timing Behavior
<table>
<thead>
<tr>
<th></th>
<th>Agenda</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Motivation</td>
</tr>
<tr>
<td>2</td>
<td>Foundations</td>
</tr>
<tr>
<td>3</td>
<td>Workload-intensity-sensitive timing behavior analysis</td>
</tr>
<tr>
<td>4</td>
<td>Empirical evaluation</td>
</tr>
<tr>
<td>5</td>
<td>Related work</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions and future work</td>
</tr>
</tbody>
</table>
Evaluation methodology

**Evaluation Metric**

Reduction of standard deviation (in percent) in relation to the original dataset for each operation and in total weighted by the number of observations per operation.

- Evaluation and simulation techniques can benefit from “reduction” of standard deviation, e.g.,
  - in terms of requiring less observations,
  - providing tighter confidence intervals,
  - requiring less or shorter simulation runs.

**Evaluation method:**

- Results for $pwi_1 - 3$ can directly be computed
- Evaluation of $pwi_4$ requires two separate data sets for training, and one for cross-validation
- Operations with less than 600 observations are accounted 0% reduction
Case study 1/3 - Distributed Web Shop

Setting

- 5-node distributed variant\(^1\) of the iBATIS JPetStore
- 34 software operation instrumented

- Probabilistic, multi-user workload using Markov4JMeter
- Real workload intensity curve, scaled to max. 80% capacity utilization

\(^1\) Instrumented sources available at http://sourceforge.net/projects/kieker
Standard deviation is reduced in average from 35% for $pwi_1$ up to 56% for $pwi_4$.

Log-transforming the $pwi$ values, before defining bins additionally improves standard deviation reduction by 29% in average.

For $pwi_4$, this results in a standard deviation reduction of 65%.

For some operations, there is no benefit.
Case study 2/3 - Telecommunication System

**Setting**

- Telecommunication signaling system of Nokia Siemens Networks
- 8 instrumented operations on two clustered nodes
- Test workload using the companies own workload simulator
- Less than 15% of CPU utilization peak

![Graph showing BHCA (tsd.) over time in seconds](image)
Case study 2/3 - Telecommunication System

Results

- \( pwi_4 \) performs best in the comparison.
- For all \( pwi \) metrics, standard deviation reduction additionally increases by more than 30% if the logarithm of the \( pwi \) values are used for defining timing behavior classes.
- Traces do not cross execution environments \( \Rightarrow pwi_2 = pwi_3 \).
Case study 3/3 - Photo Shopping and Service Portal

**Setting**

- Customer portal for ordering photo prints and other photo products of CeWe Color AG, Europe’s largest digital photo service provider.
- Large number of monitoring points: 161
- Low utilization: CPU utilization (averaged) stays below 15%
- Real workload - Kieker monitoring framework used in production environment:

![Graph showing CPU utilization over time]

Wednesday Thursday Friday Saturday Sunday

<table>
<thead>
<tr>
<th>Time</th>
<th>12:00</th>
<th>22:00</th>
<th>8:00</th>
<th>18:00</th>
<th>4:00</th>
<th>14:00</th>
<th>0:00</th>
<th>10:00</th>
<th>20:00</th>
<th>6:00</th>
<th>16:00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilization (%)</td>
<td>50%</td>
<td>100%</td>
<td>200%</td>
<td>50%</td>
<td>100%</td>
<td>200%</td>
<td>50%</td>
<td>100%</td>
<td>200%</td>
<td>50%</td>
<td>100%</td>
</tr>
</tbody>
</table>
Case study 3/3 - Photo Shopping and Service Portal

Results

- $pw_i_4$ performs best in the comparison of the four alternative methods (26.46%, 29.15% for log.).
- Single execution environment monitored $\Rightarrow pw_i_2 = pw_i_3$.
- 0% benefit was accounted for several operations with too few observations.
Related Work

- Requests are grouped by request complexity.
- Workload intensity changes related to the day time are used in network data analysis.
- Requests are grouped according to resource usage.
- Control-flow (Caller context).
- Control-flow (Stack content).
- Control-flow (Trace context).
Conclusions

Approach summary
- Goal: “Reduce” variance for statistical measurement analysis
- Workload-intensity metrics $pwi_1 - pwi_4$
- Categorization based on workload-intensity
- No additional monitoring requirements

Empirical evaluation results
- Applicability in real, distributed, enterprise software systems
- Observation: A significant part of the variance in timing behavior could be controlled by considering workload intensity.
- $pwi_4$ (operation specific weights) performed best.
- No big difference between $pwi_1$ (response times), and $pwi_2$ (execution times) in the case studies.
Future Work

- Application in the context of anomaly detection.

- Comparison of the standard deviation reduction with the $pwi$ workload-intensity metrics with that resulting from other timing behavior influences, such as parameter values, request types, and control flow context, in standard deviation reduction.

- Comparison of the $pwi$ workload-intensity metrics with other workload intensity metrics, such as CPU utilization, load average, and arrival rate.