Workload-sensitive Timing Behavior Analysis for Automatic Software Fault Localization

Matthias Rohr

BTC Business Technology Consulting AG, Germany, Graduate School TrustSoft*, University of Oldenburg, Germany

August 24, 2010
Software Engineering Group, University of Kiel, Kiel, Germany

*This work is supported by the German Research Foundation (DFG), grant GRK 1076/1
Motivation

Complex Software System

Users

Complex Software System
Software systems contain almost always faults and fail.
Software systems contain almost always faults and fail.
Manual failure diagnosis is time-consuming and error-prone.
Software systems contain almost always faults and fail. Manual failure diagnosis is time-consuming and error-prone. Runtime behavior is indicative for failures and error-propagation.
Motivation

- **Approach**: Automatic fault localization
- **Contribution**: Workload-sensitive timing behavior analysis

Complex Software System with Monitoring

- Users
- Log of Runtime Behavior Measurements
- Failure Diagnosis
- Administrators
- Diagnosis Report
- Anomaly Graphs
Primary contribution:
Timing behavior analysis methods
Evaluation: Quantitative empirical evaluation in industry studies

Secondary contribution: Fault localization approach
Evaluation: Proof-of-concept in lab-studies and application of the monitoring infrastructure in industry systems
1. Instrumentation and Monitoring

- Recording of:
  - Response times of software operation executions
  - Execution sequences corresponding to user requests
  - Host identifier

- Reconstruction of Traces and Dependency Graphs

2. Trace-Context-Sensitive Timing Behavior Analysis

3. Workload-Intensity-Sensitive Timing Behavior Analysis

4. Anomaly Detection

5. Anomaly Correlation and Fault Localization
1. Instrumentation and Monitoring

2. Trace-Context-Sensitive Timing Behavior Analysis
   1. Identification of trace contexts
   2. Defining classes of observations based on the trace context
   3. (Re-)Merging classes

3. Workload-Intensity-Sensitive Timing Behavior Analysis

4. Anomaly Detection

5. Anomaly Correlation and Fault Localization
Workload-intensity-sensitive Timing Behavior Analysis

1. Instrumentation and Monitoring

2. Trace-Context-Sensitive Timing Behavior Analysis

3. Workload-Intensity-Sensitive Timing Behavior Analysis
   1. Definition of a workload metric by machine learning
   2. Splitting observations according to workload-intensity

4. Anomaly Detection

5. Anomaly Correlation and Fault Localization

Matthias Rohr, Workload-sensitive Timing Behavior Analysis
1. Instrumentation and Monitoring

2. Trace-Context-Sensitive Timing Behavior Analysis

3. Workload-Intensity-Sensitive Timing Behavior Analysis

4. Anomaly Detection
   - Evaluation of new observations in context of a profile.
   - How normal is a new observation?

5. Anomaly Correlation and Fault Localization
1. Instrumentation and Monitoring

2. Trace-Context-Sensitive Timing Behavior Analysis

3. Workload-Intensity-Sensitive Timing Behavior Analysis

4. Anomaly Detection

5. Anomaly Correlation and Fault Localization
   - Derivation of component ratings from execution ratings
   - Derivation of causes from symptoms
Example: Anomaly Correlation and Fault Localization

Motivation
Fault localization
TracSTA
TracSTA Evaluation
WiSTA
WiSTA Evaluation
Related Work
Summary
References

Matthias Rohr, Workload-sensitive Timing Behavior Analysis
Example: Anomaly Correlation and FaultLocalization

Virtual Machine 'klotz'
- ActionServlet
  - OrderBean
  - CatalogBean
  - CartBean
  - AccountBean

Virtual Machine 'tier'
- server.CatalogService
  - ItemSqlMapDao
- ProductSqlMapDao

Virtual Machine 'scooter'
- server.AccountService
  - AccountSqlMapDao

Virtual Machine 'puck'
- server.OrderService
  - OrderSqlMapDao

Matthias Rohr, Workload-sensitive Timing Behavior Analysis
Motivation

Fault Localization Approach

Trace-Context-Sensitive Timing Behavior Analysis
- Empirical evaluation

Workload-Intensity-Sensitive Timing Behavior Analysis

Related work

Summary
Trace-context-sensitive Timing Behavior Analysis: Motivation

**Motivation**

**Fault localization**

**TracSTA**

**Evaluation**

**WiSTA**

**Evaluation**

**Related Work**

**Summary**

**References**

**Matthias Rohr, Workload-sensitive Timing Behavior Analysis**
Goal

Derivation of a timing behavior model with lower variance and less multi-modality in timing behavior distributions
Trace-context-sensitive Timing Behavior Analysis

Goal
Derivation of a timing behavior model with lower variance and less multi-modality in timing behavior distributions

Definition: Calling-context of an operation call
Set of circumstances or facts that surround an operation call, in particular the sequence of surrounding operation executions.
Goal

Derivation of a timing behavior model with lower variance and less multi-modality in timing behavior distributions

Definition: Calling-context of an operation call

Set of circumstances or facts that surround an operation call, in particular the sequence of surrounding operation executions.

Steps of the approach

1. Integrate calling-context information into timing behavior model
2. Optimize timing behavior model; e.g. model size reduction
Calling-context equivalence

Three levels of abstraction for calling-context information:

**Equivalence relations on software operation executions**

Two executions of the same operation are

- **caller-context equivalent** (cp. Graham et al. [1982])
  := called from operations with the same name.

- **stack-context equivalent** (cp. Ammons et al. [1997])
  := equal paths from their tree nodes to root

- **trace-context equivalent** :=
  1) corresponding trees are equal
  2) tree nodes have same position in tree
Calling-context equivalence

Three levels of abstraction for calling-context information:

**Equivalence relations on software operation executions**

Two executions of the same operation are

- *caller-context equivalent* (cp. Graham et al. [1982])
  
  \[\text{called from operations with the same name.}\]

- *stack-context equivalent* (cp. Ammons et al. [1997])
  
  \[\text{equal paths from their tree nodes to root}\]

- *trace-context equivalent*:
  
  1) corresponding trees are equal
  2) tree nodes have same position in tree

- Stack-context equivalence \(\Rightarrow\) caller-context equivalence
- Trace-context equivalence \(\Rightarrow\) stack-context equivalence
Two executions of the same operation are **caller-context equivalent** if they are called from operations with the same name.

**Dynamic call tree for trace 1**

**Dynamic call tree for trace 2**

Caller–context equivalence for E.e

(Cp. Graham et al. [1982])
Two executions of the same operation are **stack-context equivalent** if the paths from the corresponding nodes to its root are equal.
Two executions of the same operation are \textit{trace-context equivalent} if the corresponding trees are equal and the both executions correspond to dynamic call tree nodes with the same position within the tree.

![Trace-context equivalence for A.a](image)

\begin{itemize}
  \item Dynamic call tree for trace 1
  \item Dynamic call tree for trace 2
\end{itemize}

For both traces together, number of trace-contexts = total number of nodes in non-equal trees = 19.
Calling-context sensitive timing behavior model derived from monitoring data:

- Complete partitioning of all response times based on trace-context, stack-context, or caller-context equivalence
Dealing with issues resulting from calling-context analysis:

- **Efficiency:**
  - Too many calling-contexts
  - Calling-contexts do not differ in timing behavior distributions

- **Applicability and robustness of statistical methods**
  - Calling-contexts with an insufficient number of measurements

Leaf nodes with similar distribution characteristics are merged.
Leaf nodes without a sufficient amount of observations are linked to an ancestor node.
Leaf nodes without siblings are removed.
Dealing with issues resulting from calling-context analysis:

- **Efficiency:**
  - Too many calling-contexts
  - Calling-contexts do not differ in timing behavior distributions

- **Applicability and robustness of statistical methods**
  - Calling-contexts with an insufficient number of measurements

Leaf nodes without similar distribution characteristics are **merged**

Leaf nodes without a sufficient amount of observations are **linked** to an ancestor node

Leaf nodes without siblings are **removed**
Goal

Derivation of a timing behavior model with lower variance and less multi-modality in timing behavior distributions
Goal
Derivation of a timing behavior model with lower variance and less multi-modality in timing behavior distributions

Definition: Calling-context of an operation call
Set of circumstances or facts that surround an operation call, in particular the sequence of surrounding operation executions.
**Goal**
Derivation of a timing behavior model with lower variance and less multi-modality in timing behavior distributions

**Definition: Calling-context of an operation call**
Set of circumstances or facts that surround an operation call, in particular the sequence of surrounding operation executions.

**Steps of the approach**
1. Integrate calling-context information into timing behavior model
2. Optimize timing behavior model; e.g. model size reduction
Calling-context equivalence

Three levels of abstraction for calling-context information:

**Equivalence relations on software operation executions**

Two executions of the same operation are

- **caller-context equivalent** (cp. Graham et al. [1982])
  \[\text{:=} \text{called from operations with the same name.}\]

- **stack-context equivalent** (cp. Ammons et al. [1997])
  \[\text{:= equal paths from their tree nodes to root}\]

- **trace-context equivalent**
  \[1) \text{corresponding trees are equal}\]
  \[2) \text{tree nodes have same position in tree}\]
Three levels of abstraction for calling-context information:

**Equivalence relations on software operation executions**

Two executions of the same operation are

- **caller-context equivalent** (cp. Graham et al. [1982])
  
  \[\text{called from operations with the same name.}\]

- **stack-context equivalent** (cp. Ammons et al. [1997])
  
  \[\text{equal paths from their tree nodes to root}\]

- **trace-context equivalent**
  
  1) corresponding trees are equal
  2) tree nodes have same position in tree

**Stack-context equivalence \(\Rightarrow\) caller-context equivalence**

**Trace-context equivalence \(\Rightarrow\) stack-context equivalence**
Two executions of the same operation are **caller-context equivalent** if they are called from operations with the same name.

Calling-context equivalence for E.e

Dynamic call tree for trace 1

Dynamic call tree for trace 2

Motivation
Fault localization
TracSTA
TracSTA Evaluation
WISTA
WISTA Evaluation
Related Work
Summary
References
Two executions of the same operation are **stack-context equivalent** if the paths from the corresponding nodes to its root are equal.

**Stack–context equivalence for E.e**

Dynamic call tree for trace 1

Dynamic call tree for trace 2
Two executions of the same operation are **trace-context equivalent** if the corresponding trees are equal and the both executions correspond to dynamic call tree nodes with the same position within the tree.

For both traces together, number of trace-contexts = total number of nodes in non-equal trees = 19.
Monitored response times of all instrumented software operations

RT = [ ... ]

Partitioning based on operation name equality

Partitioning based on caller-context equivalence

Partitioning based on stack-context equivalence

Partitioning based on trace-context equivalence

Calling-context sensitive timing behavior model derived from monitoring data:

- Complete partitioning of all response times based on trace-context, stack-context, or caller-context equivalence
Dealing with issues resulting from calling-context analysis:

- **Efficiency:**
  - Too many calling-contexts
  - Calling-contexts do not differ in timing behavior distributions

- **Applicability and robustness of statistical methods:**
  - Calling-contexts with an insufficient number of measurements

Leaf nodes with similar distribution characteristics are merged.
Leaf nodes without a sufficient amount of observations are linked to an ancestor node.
Leaf nodes without siblings are removed.
Dealing with issues resulting from calling-context analysis:

- **Efficiency:**
  - Too many calling-contexts
  - Calling-contexts do not differ in timing behavior distributions

- **Applicability and robustness of statistical methods**
  - Calling-contexts with an insufficient number of measurements

Leaf nodes with similar distribution characteristics are **merged**
Leaf nodes without a sufficient amount of observations are **linked** to an ancestor node
Leaf nodes without siblings are **removed**
Motivation

Fault Localization Approach

**Trace-Context-Sensitive Timing Behavior Analysis**
- Empirical evaluation

Workload-Intensity-Sensitive Timing Behavior Analysis

Related work

Summary
Case study 1/2 - 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
## Results 1/2

<table>
<thead>
<tr>
<th>Calling-context type</th>
<th>Average st.dev. decrease in %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>18 mon.pts. scenario</td>
</tr>
<tr>
<td>Caller-context analysis</td>
<td>0.2</td>
</tr>
<tr>
<td>Stack-context analysis</td>
<td>0.6</td>
</tr>
<tr>
<td>Trace-context analysis</td>
<td>3.3</td>
</tr>
</tbody>
</table>

### (a) 18 mon.pts. scenario

### (b) Full instrumentation

A significant part of the variance is connected to trace-context information!
Motivation
Fault localization
TracSTA
TracSTA Evaluation
Evaluation
WiSTA
WiSTA Evaluation
Related Work
Summary
References

Case study 1/2 - Distributed Web Shop

Results 2/2

Figure: Average decrease in standard deviation for different numbers of calling-contexts.
Case study 2/2 - 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
Figure: Probability density distribution for an operation.
Results 2/2

(a) PDF for trace-context 1 and 3.

(b) PDF for trace-context 2.

Total average decrease in standard deviation

136.47 → 53.83 ; 2.20; 49.74 → 35.94 → 73.66% reduction of standard deviation
Agenda

1. Motivation
2. Fault Localization Approach
3. Trace-Context-Sensitive Timing Behavior Analysis
4. Workload-Intensity-Sensitive Timing Behavior Analysis
   - Empirical evaluation
5. Related work
6. Summary
Hypothesis of WIA: Considering a multi-user system’s workload intensity reduces uncertainty in anomaly detection.

**Figure:** Anomaly detection in systems with changing workload intensity.
1. Computation of workload intensity

- Computation of workload intensity for each execution

→ next slides

2. Categorization based on workload intensity ranges
1. Computation of workload intensity

- Computation of workload intensity for each execution

→ next slides

2. Categorization based on workload intensity ranges

1. The \( pw_i \) 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</th>
<th>Time metric</th>
<th>Execution environment</th>
<th>Operation weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pwi_1$</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.

<table>
<thead>
<tr>
<th>Operation, execution</th>
<th>Trace</th>
<th>Execution 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>
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>pwi₁</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>
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 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>

Matthias Rohr, Workload-sensitive Timing Behavior Analysis
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 $pwi_1$ for distributed systems.
- Assumption: Execution contexts have own hardware platform
- Hypothesis: Little competition for resources with executions in other execution environments.
- $pwi_3$ uses execution times instead of response times
An operation execution’s $pwi_3$ is the average number of concurrent active executions within the same execution environment during its execution time period.

### Table 1: Operation Execution Details

<table>
<thead>
<tr>
<th>Operation, Execution</th>
<th>Trace</th>
<th>Execution Environment</th>
<th>$pwi_1$</th>
<th>$pwi_2$</th>
<th>$pwi_3$</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>

### Diagram 1: Execution Timeline

- **Active**: Solid bars
- **Not active**: Dashed bars

---

Matthias Rohr, Workload-sensitive Timing Behavior Analysis
An operation execution’s $pwi_3$ is the average number of concurrent active executions within the same execution environment during its execution time period.
$pwi_4$ 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
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><em>work</em></td>
<td>2.01</td>
<td>-0.05</td>
</tr>
<tr>
<td><em>wait</em></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><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>

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

<table>
<thead>
<tr>
<th></th>
<th>(p\text{wi}_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>work</strong></td>
<td>72.5 ± 2</td>
</tr>
<tr>
<td><strong>wait</strong></td>
<td>18.8 ± 9</td>
</tr>
</tbody>
</table>
Motivation
Fault localization
TracSTA
Evaluation
WiSTA
Evaluation
Related Work
Summary
References

$pwi_4$ Example 2/2

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

Fault Localization Approach

Trace-Context-Sensitive Timing Behavior Analysis

Workload-Intensity-Sensitive Timing Behavior Analysis
  - Empirical evaluation

Related work

Summary
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 $pwi_4$. 
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

Nokia Siemens Networks

BHCA (tsd.)

Calls active at a time

Time in seconds

Matthias Rohr, Workload-sensitive Timing Behavior Analysis
**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 $\Rightarrow pwi_2 = pwi_3$. 

---

Matthias Rohr, Workload-sensitive Timing Behavior Analysis
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 workload distribution]

- 50%
- 100%
- 200%

- Wednesday
- Thursday
- Friday
- Saturday
- Sunday

12:00 22:00 8:00 18:00 4:00 14:00 0:00 10:00 20:00 6:00 16:00
Case study 3/3 - Photo Shopping and Service Portal

Results

- $pwi_4$ performs best in the comparison of the four alternative methods (26.46%, 29.15% for log.).
- Single execution environment monitored $⇒ pwi_2 = pwi_3$.
- 0% benefit was accounted for several operations with too few observations.
1. Motivation

2. Fault Localization Approach

3. Trace-Context-Sensitive Timing Behavior Analysis

4. Workload-Intensity-Sensitive Timing Behavior Analysis

5. Related work

6. Summary
Summary

Trace-context-sensitive timing behavior analysis
- Goal: “Reduce” variance for statistical measurement analysis
- Categorization of operation executions based on trace context information
- Evaluated in field and lab studies

Workload-intensity-sensitive timing behavior analysis
- Goal: “Reduce” variance for statistical measurement analysis
- Workload-intensity metrics $pwi_1 - pwi_4$
- Categorization based on workload-intensity
- Evaluated in field and lab studies

Additional contributions
- Workload-sensitive fault localization approach
- Contributions to the Kieker monitoring framework