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

Matthias Rohr[1,2], André van Hoorn[2], Wilhelm Hasselbring[2,3]Marco Lübcke[4], and Sergej Alekseev[5,6]

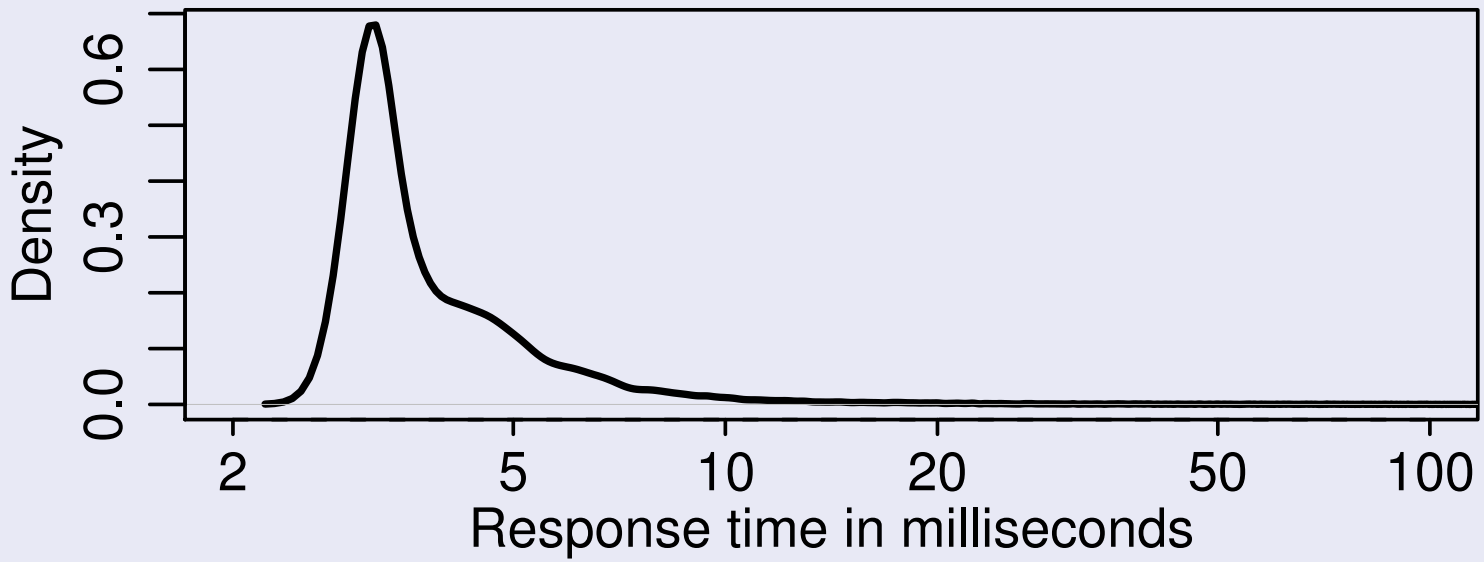
[1] BTC Business Technology Consulting AG, Germany,
[*], [2] Graduate School TrustSoft, University of Oldenburg, Germany,
[3] Software Engineering Group, University of Kiel, Germany,
[4] CeWe Color AG & Co. OHG, Oldenburg, Germany,
[5] Nokia Siemens Networks GmbH, Berlin, Germany,
[6] Hochschule Mittweida,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

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

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

Motivation 1/2

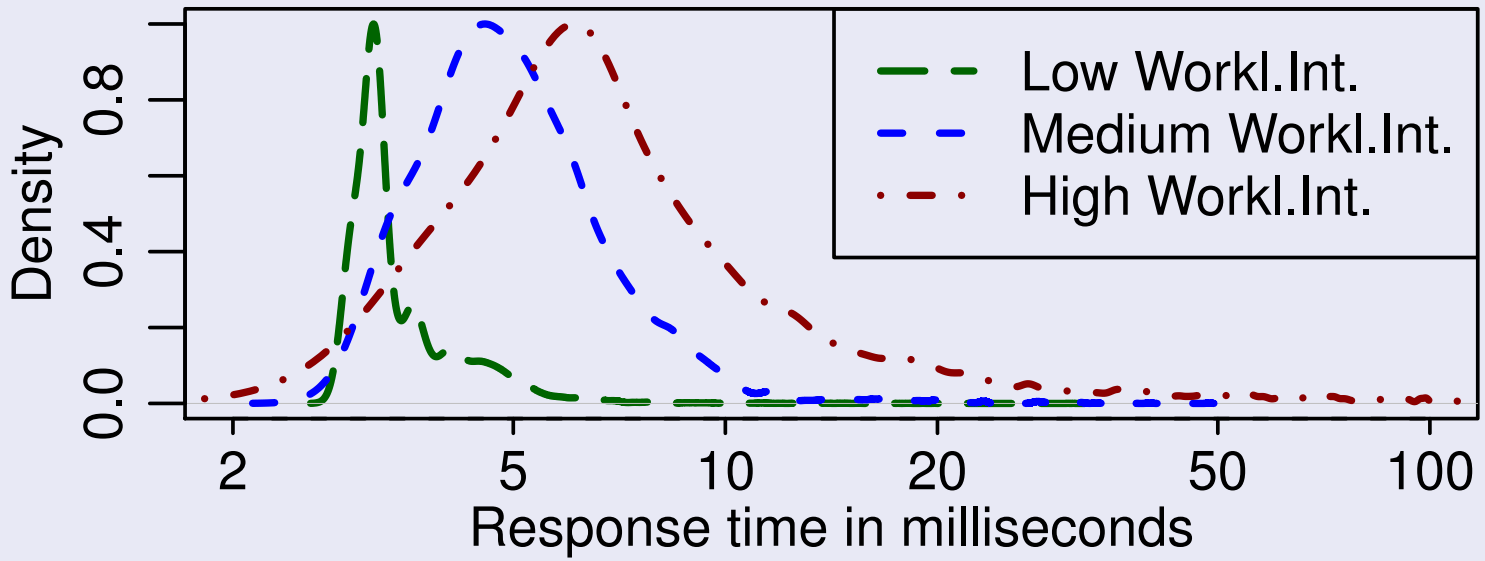
Software operation response times



- 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 1/2

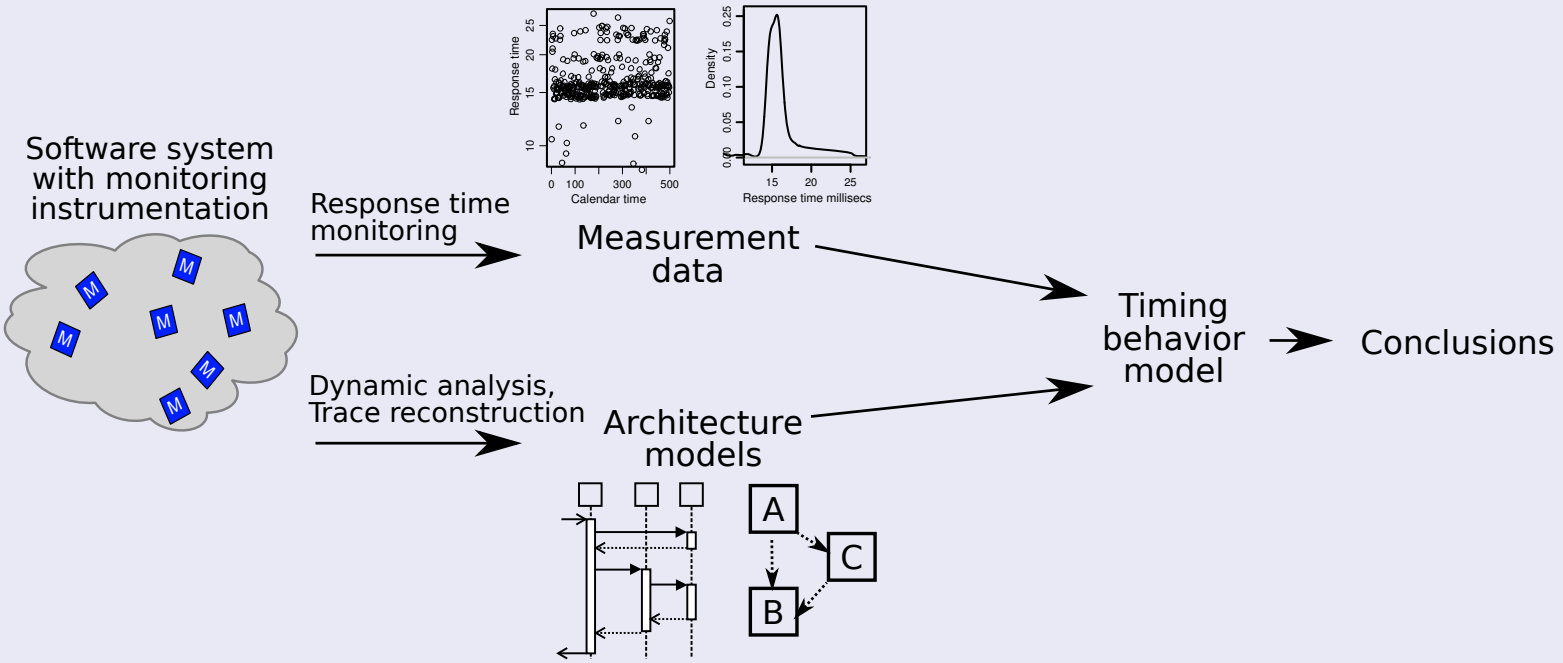
Software operation response times



- 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

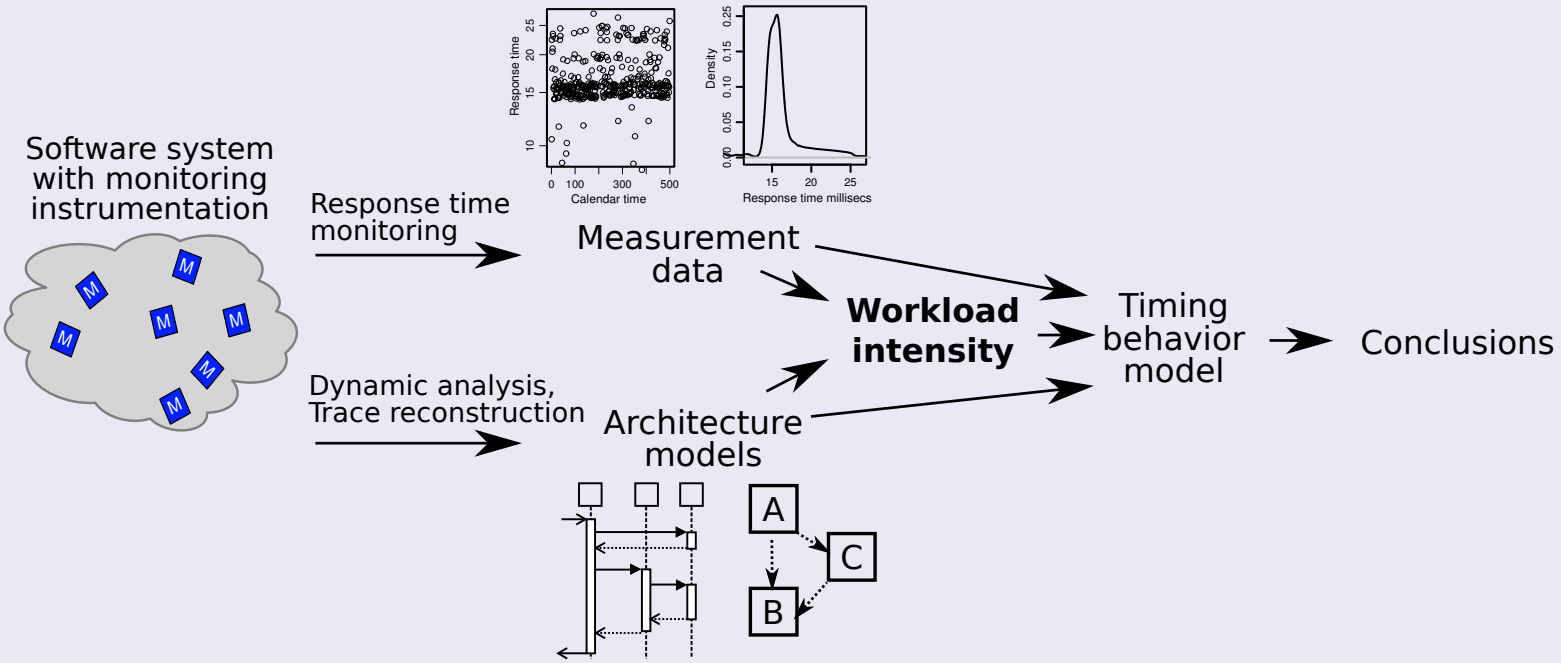
Without considering workload intensity



Our approach

Motivation 2/2 - Approach idea

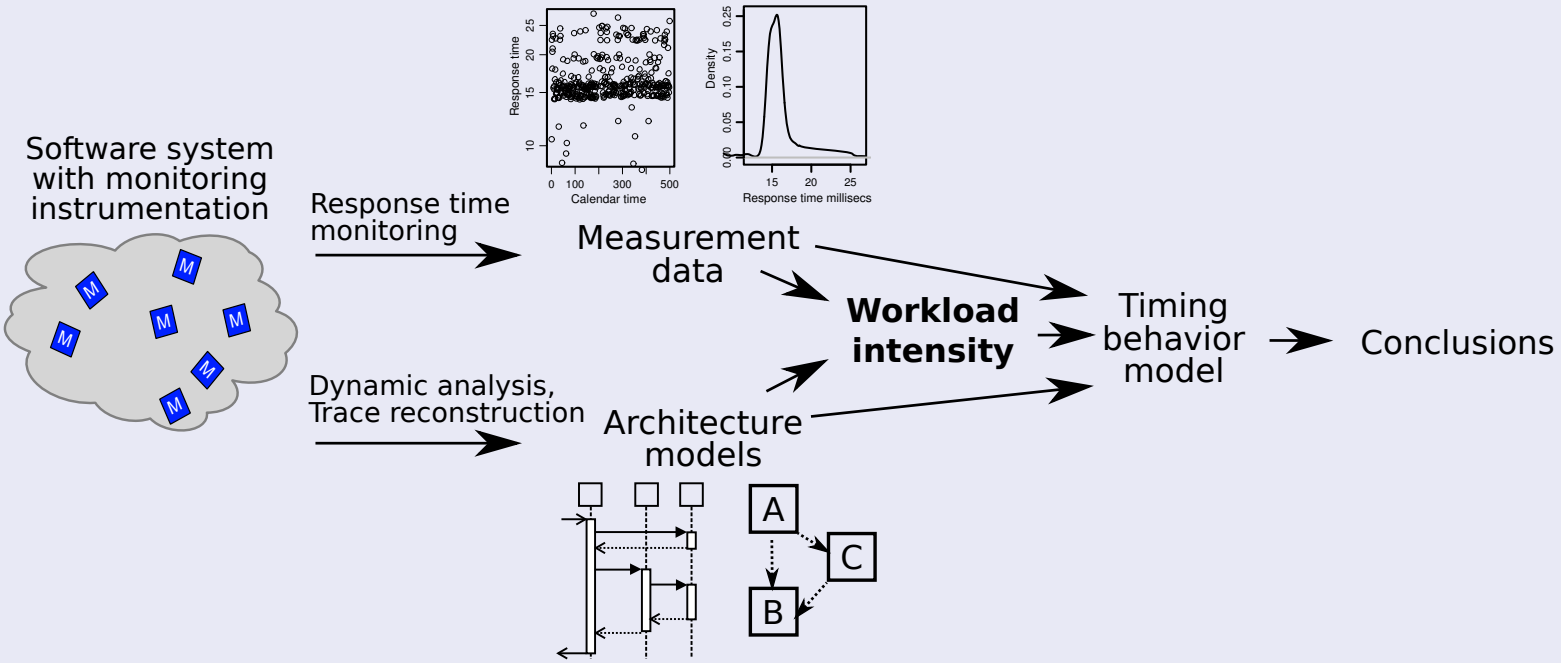
With considering workload intensity



Our approach

Motivation 2/2 - Approach idea

With considering workload intensity



Our approach

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

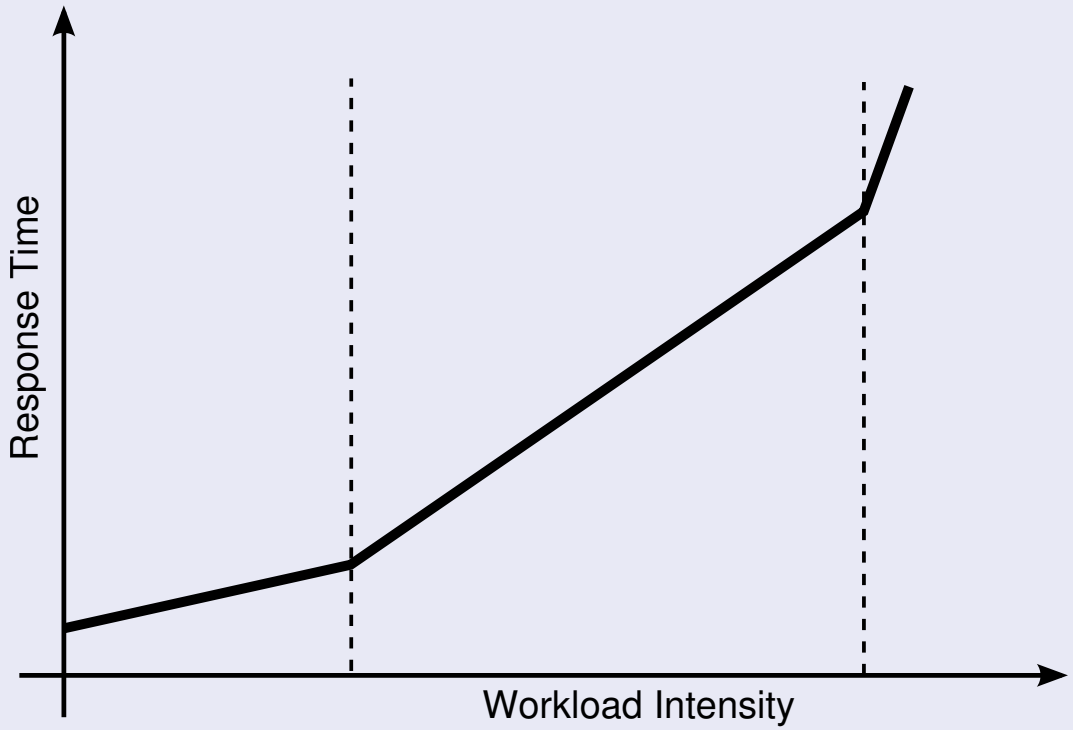
Agenda

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

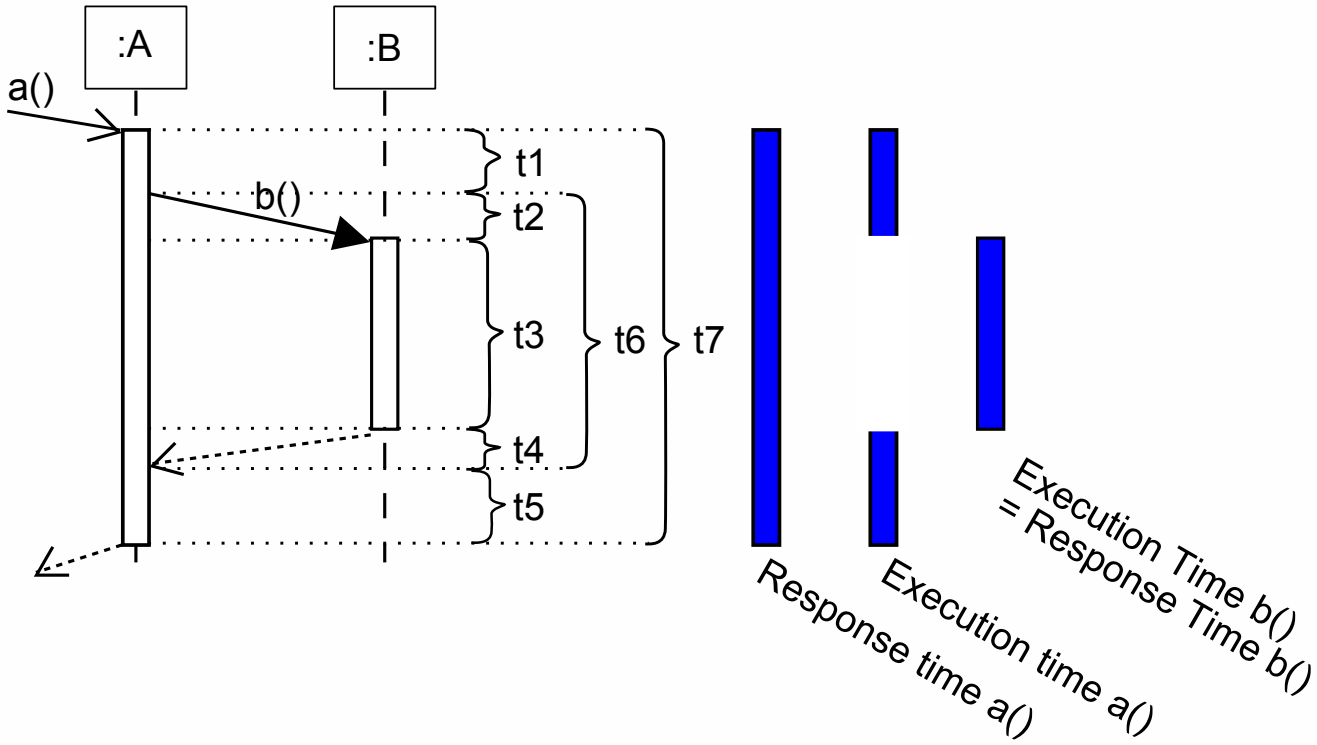


Figure: Response times and execution times.

Agenda

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

Workload-intensity-sensitive Timing Behavior Analysis

1. Monitoring

2. Computation of workload-intensity from monitoring data:

→ next slides

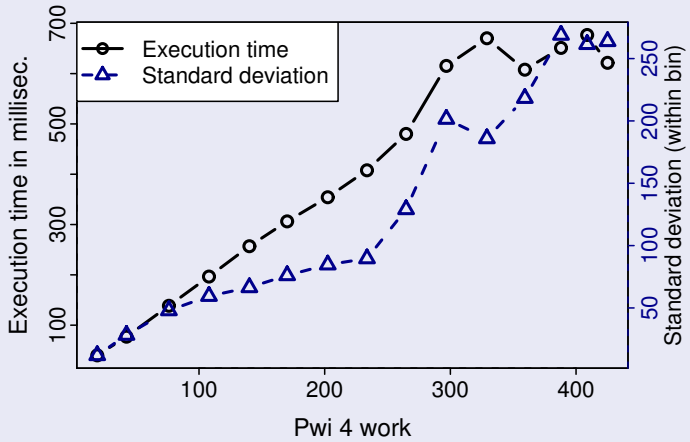
3. Categorization based on workload-intensity levels

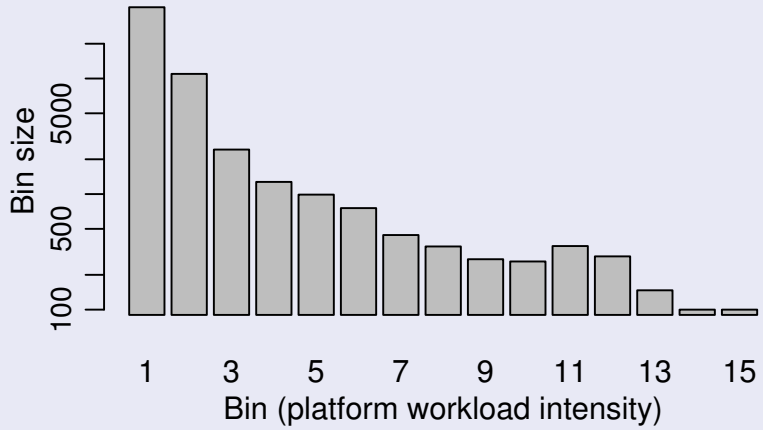
Workload-intensity-sensitive Timing Behavior Analysis

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)

Workload intensity metrics

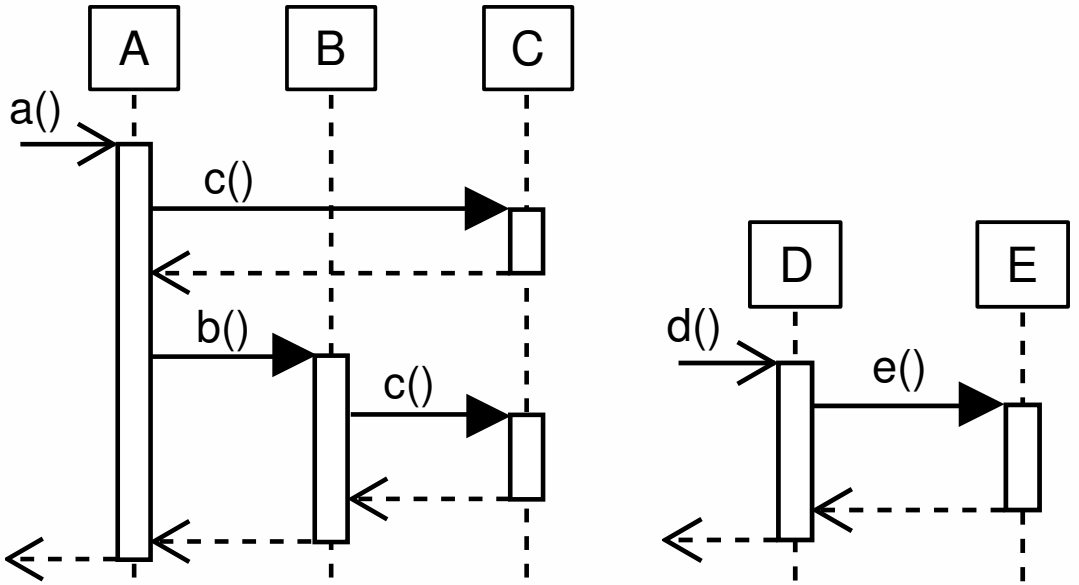
- Key element of our approach: Four alternative workload-intensity metrics, denoted *pwi* (Platform Workload Intensity):

Metric	Time metric	Execution environment	Operation weighting
pwi_1	Response times	Non-distributed	No weighting
pwi_2	Execution times	Non-distributed	No weighting
pwi_3	Execution times	Distributed	No weighting
pwi_4	Execution times	Distributed	Learned

pwi_1

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.



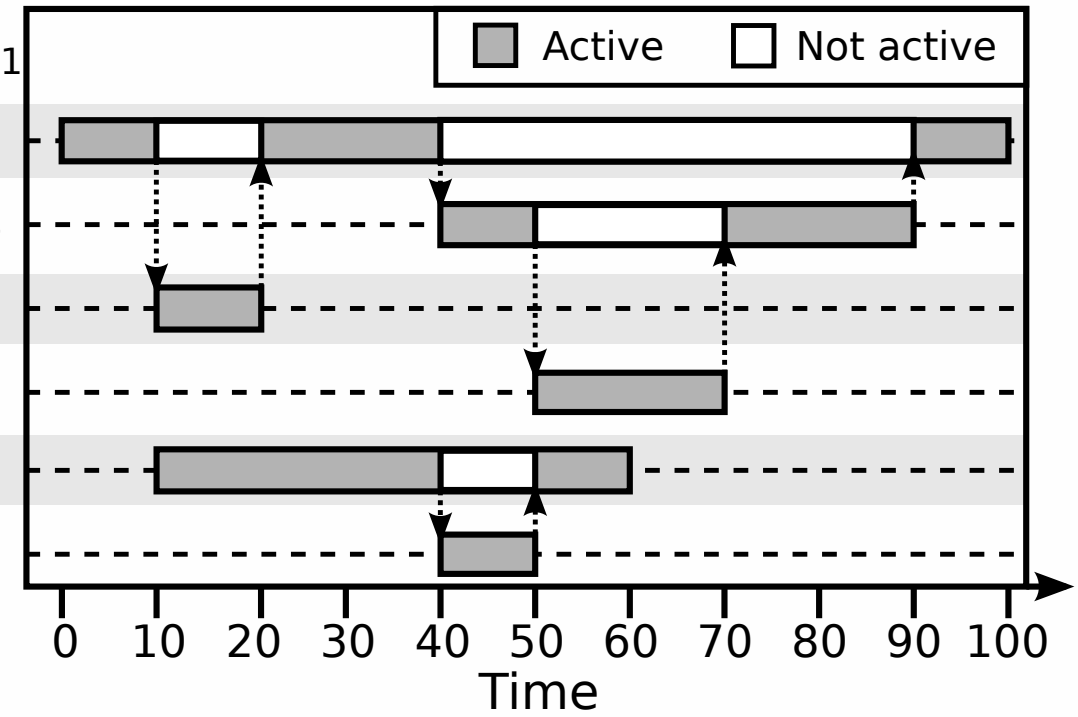
(a) Trace 1 **(b)** Trace 2

Figure: Example traces: UML Sequence Diagrams

pwi_1

Average number of concurrent traces during the time period between the start (call action) and the end of an operation execution.

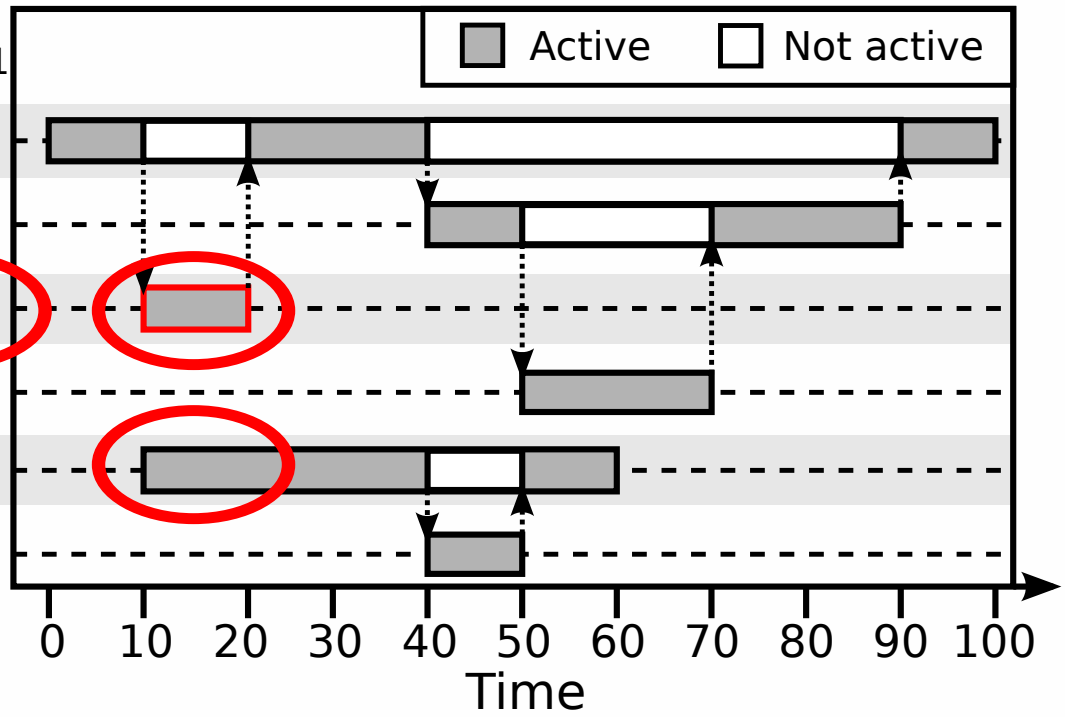
operation, execution	trace	exec. env.	pwi_1
a()	1	1	1.5
b()	1	1	1.4
c(),1	1	1	2
c(),2	1	1	1.5
d()	2	1	2
e()	2	1	2



pwi_1

Average number of concurrent traces during the time period between the start (call action) and the end of an operation execution.

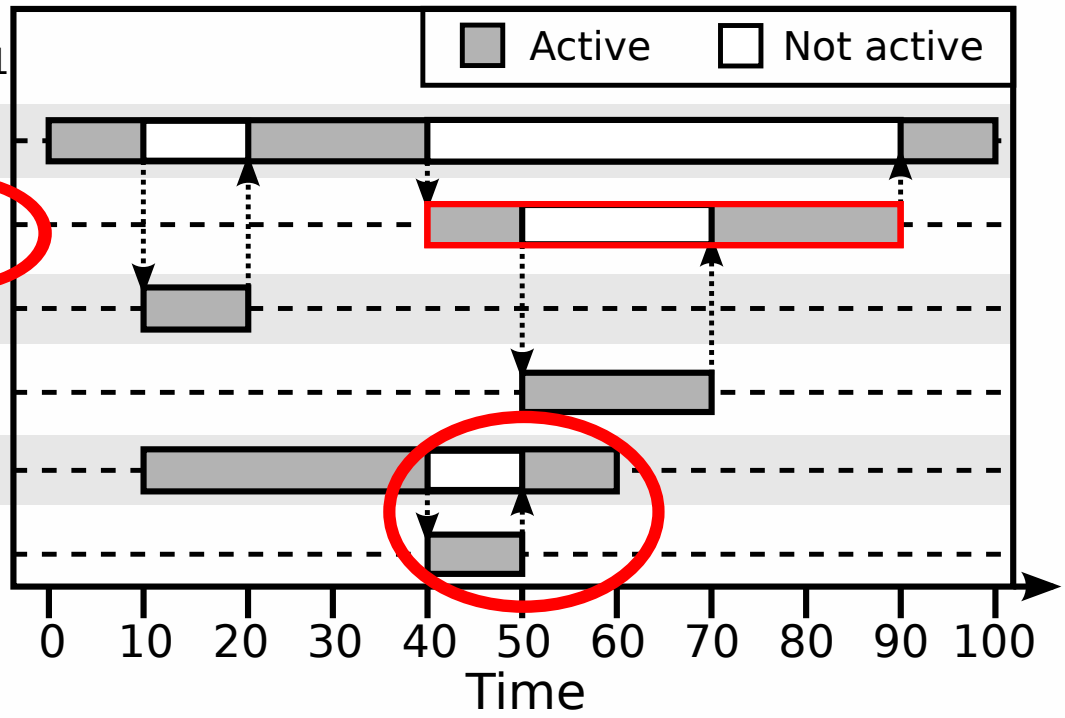
operation, execution	trace	exec. env.	pwi_1
a()	1	1	1.5
b()	1	1	1.4
c(),1	1	1	**2**
c(),2	1	1	1.5
d()	2	1	2
e()	2	1	2



pwi_1

Average number of concurrent traces during the time period between the start (call action) and the end of an operation execution.

operation, execution	trace	exec. env.	pwi_1
a()	1	1	1.5
b()	1	1	**1.4**
c(),1	1	1	2
c(),2	1	1	1.5
d()	2	1	2
e()	2	1	2



pwi_2

An operation execution's pwi_2 is the average number of concurrent traces during its execution time period.

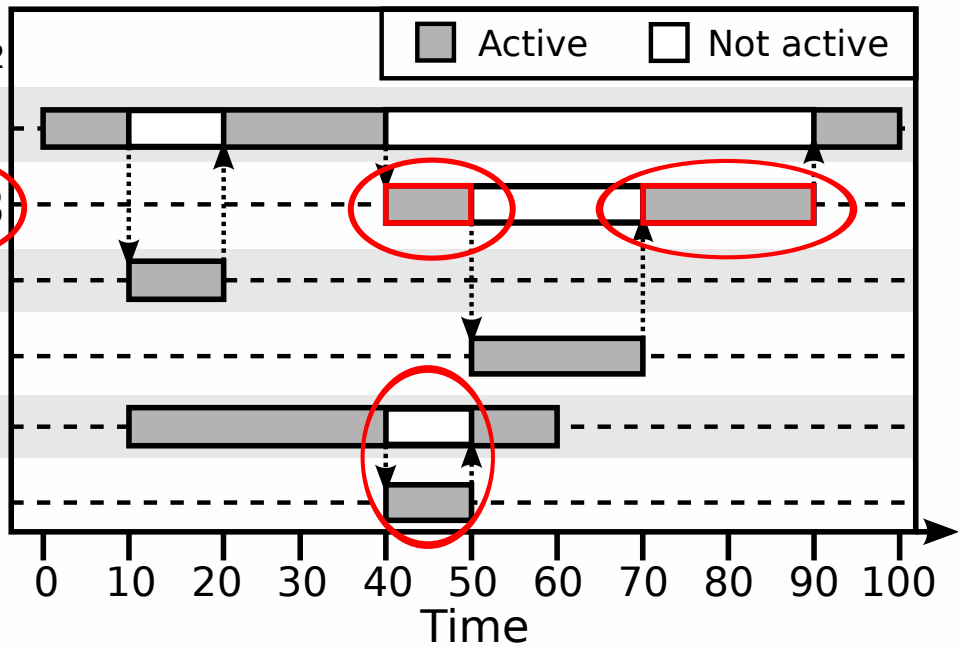
- Difference to pwi_1: Execution time period instead of response time period
- No competition for resources during waiting for sub-calls

pwi_2

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	trace	exec. env.	pwi_1	pwi_2
a()	1	1	1.5	1.5
b()	1	1	**1.4**	**1.33**
c(),1	1	1	2	2
c(),2	1	1	1.5	1.5
d()	2	1	2	2
e()	2	1	2	2



pwi_3

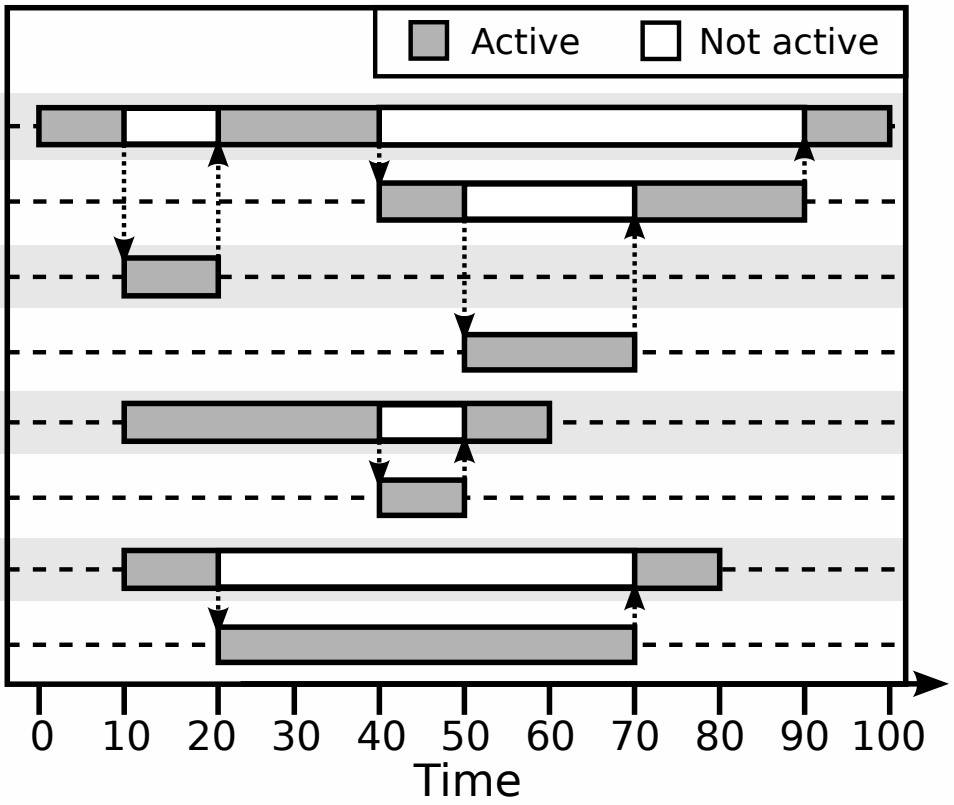
An operation execution's pwi_3 is the average number of concurrent **active executions within the same execution environment** during its execution time period.

- 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.

pwi_3

An operation execution's pwi_3 is the average number of concurrent **active executions within the same execution environment** during its execution time period.

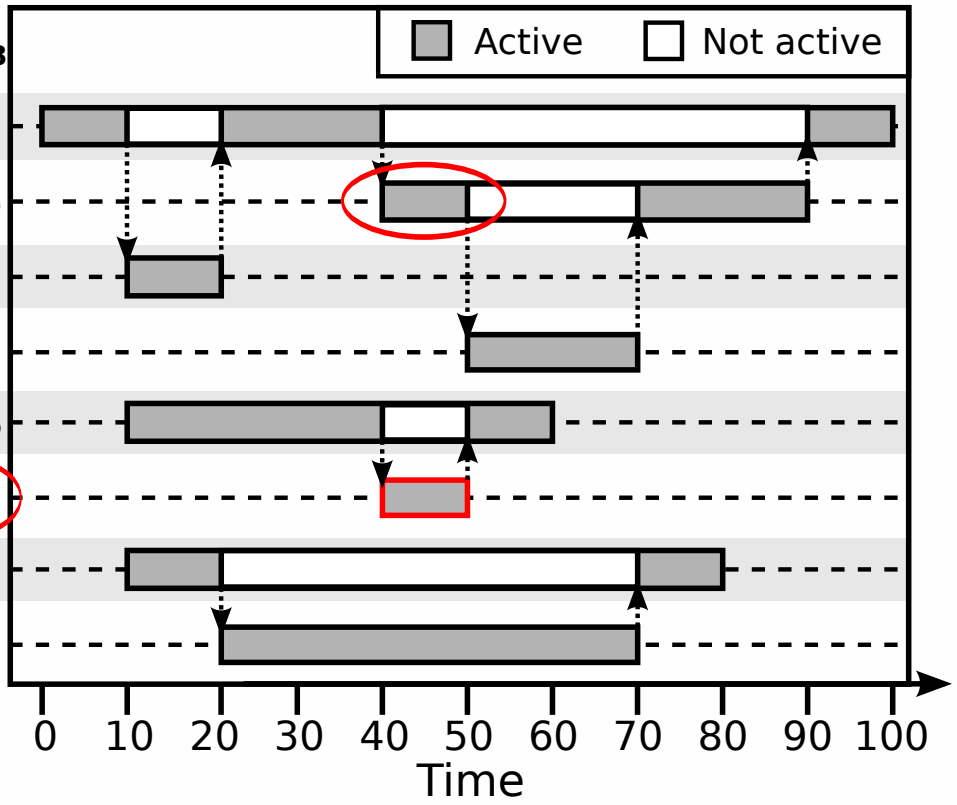
operation, execution	trace	exec. env.	pwi_1	pwi_2	pwi_3
a()	1	1	2.2	2	1.5
b()	1	1	2.2	2	1.66
c(),1	1	1	3	3	3
c(),2	1	1	2.5	2.5	1.5
d()	2	1	3	3	2.25
e()	2	1	3	3	2
g()	3	1	2.71	2.5	2.5
h()	3	**2**	2.8	2.8	1



pwi_3

An operation execution's pwi_3 is the average number of concurrent **active executions within the same execution environment** during its execution time period.

operation, execution	trace	exec. env.	pwi_1	pwi_2	pwi_3
a()	1	1	2.2	2	1.5
b()	1	1	2.2	2	1.66
c(),1	1	1	3	3	3
c(),2	1	1	2.5	2.5	1.5
d()	2	1	3	3	2.25
e()	2	1	**3**	**3**	**2**
g()	3	1	2.71	2.5	2.5
h()	3	**2**	2.8	2.8	1



pwi_4

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

pwi_4 Example 1/2

Software system with 2 operations:

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

pwi_4 Example 1/2

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

pwi$_4$ Example 1/2

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:

	work	*wait*
work	2.01	-0.05
wait	1.03	0.05

pwi_4 Example 1/2

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:

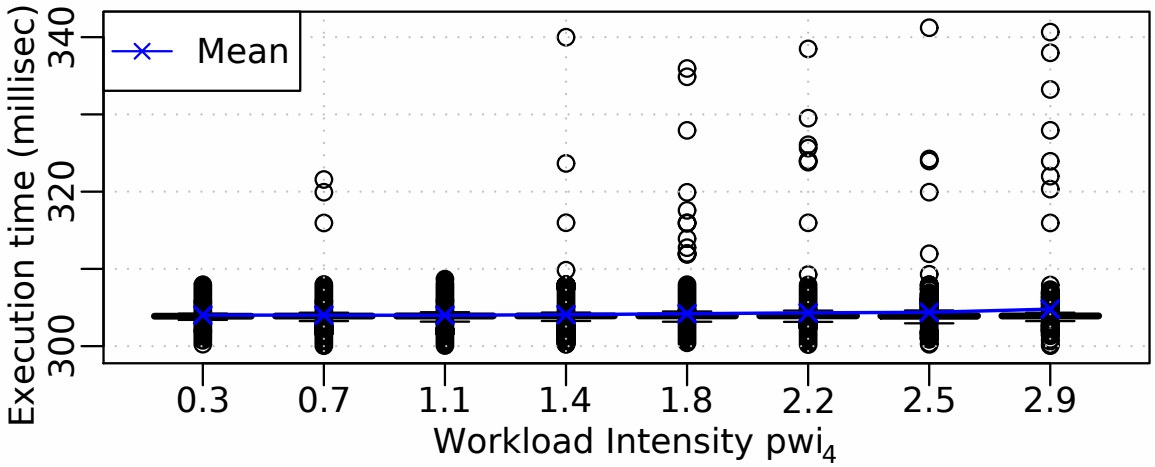
Weight matrix:

	work	*wait*
work	2.01	-0.05
wait	1.03	0.05

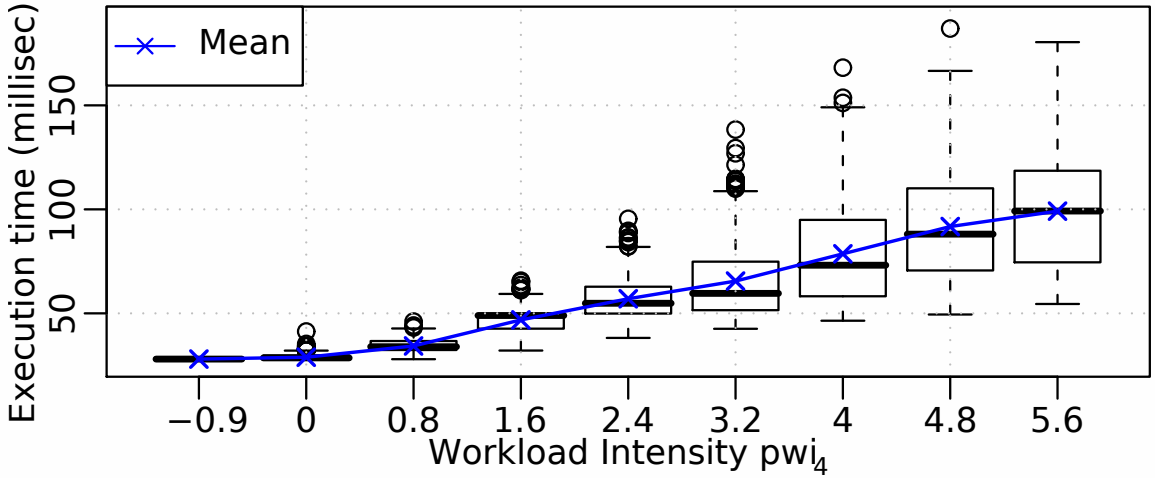
Standard dev. reduction (%):

	pwi_4
work	72.5 ± 2
wait	18.8 ± 9

pwi_4 Example 2/2



(a) wait



(b) work

Agenda

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 [**?**].

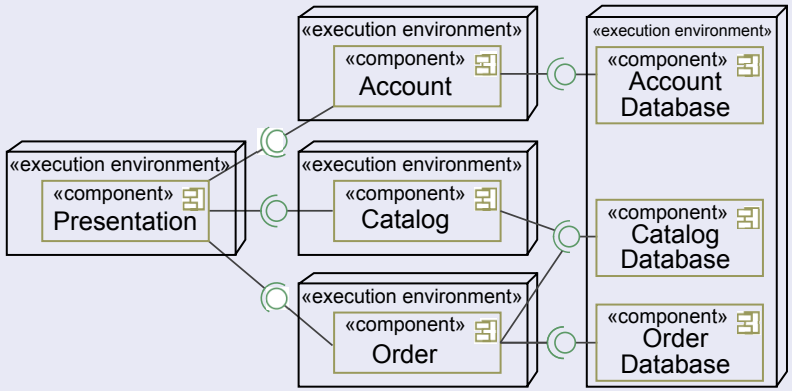
Evaluation method:

- 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

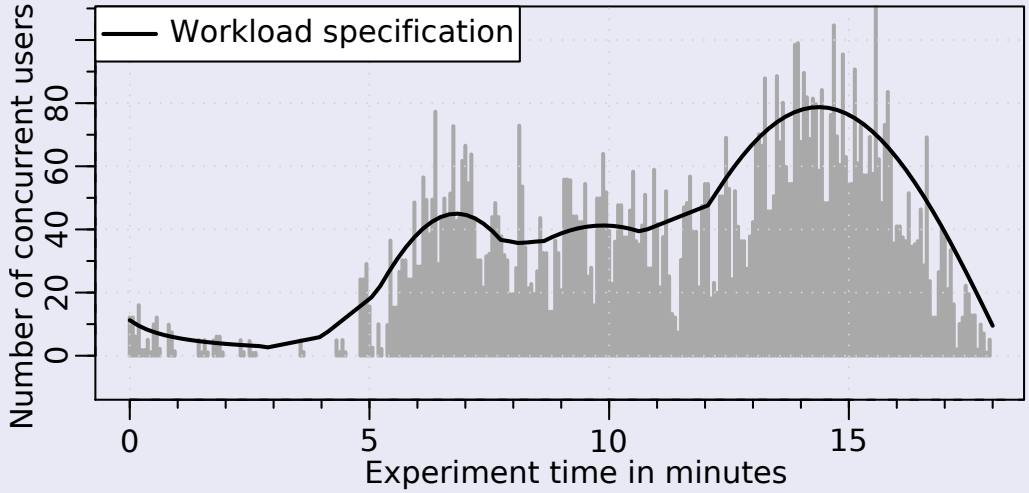
Case study 1/3 - Distributed Web Shop

Setting

- 5-node distributed variant[1] of the iBATIS JPetStore
- 34 software operation instrumented



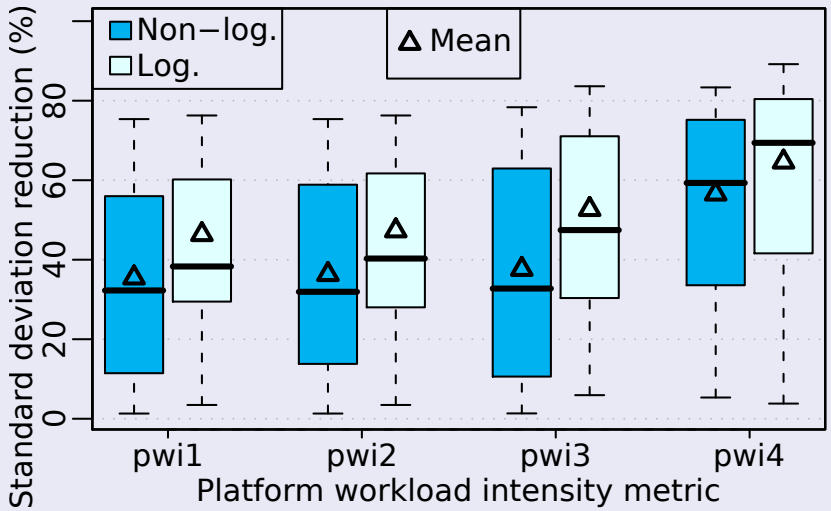
- Probabilistic, multi-user workload using Markov4JMeter
- Real workload intensity curve, scaled to max. 80% capacity utilization



[1] Instrumented sources available at http://sourceforge.net/projects/kieker

Case study 1/3 - Distributed Web Shop

Results



- 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.

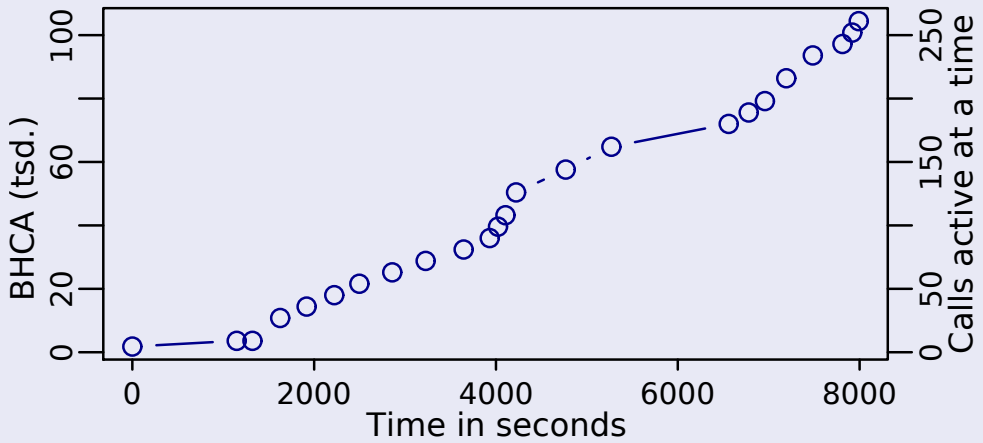
Case study 2/3 - Telecommunication System

Setting

- Telecommunication signaling system of Nokia Siemens Networks
- 8 instrumented operations on two clustered nodes

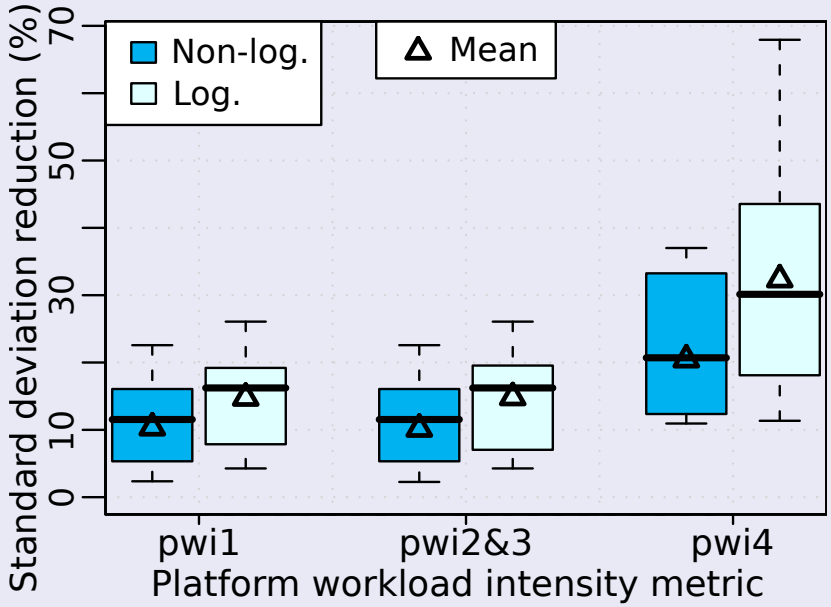
Nokia Siemens Networks

- Test workload using the companies own workload simulator
- Less than 15% of CPU utilization peak



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 $\Rightarrow pwi_2 = pwi_3$.

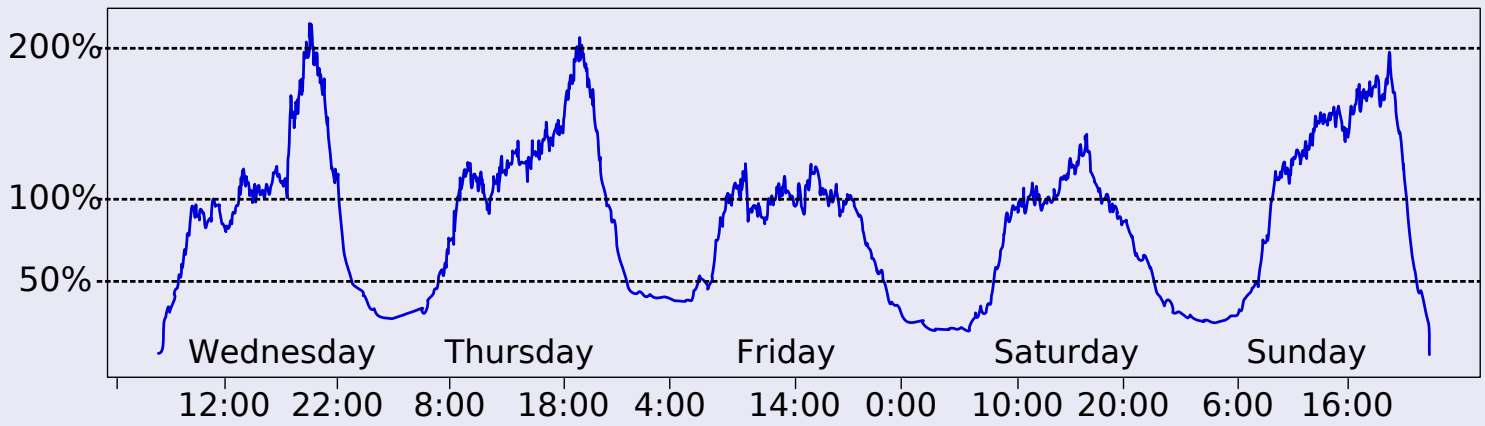
Case study 3/3 - Photo Shopping and Service Portal

Setting

- Customer portal for ordering photo prints and other photo products of CeWe Color AG, Europe's largest digital photo service provider.

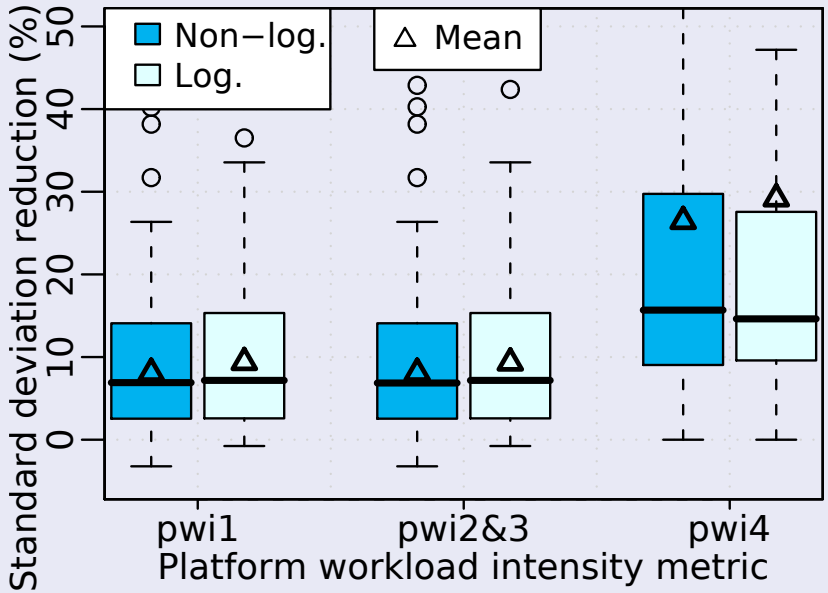
cewe color

- Large number of monitoring points: 161
- Low utilization: CPU utilization (averaged) stays below 15%
- Real workload - Kieker monitoring framework used in production environment:



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 $\Rightarrow$ $pwi_2 = pwi_3$.
- 0% benefit was accounted for several operations with too few observations.

Agenda

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).

Agenda

1. Motivation
2. Foundations
3. Workload-intensity-sensitive timing behavior analysis
4. Empirical evaluation
5. Related work
6. **Conclusions and future work**

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.

References I